POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering - Ingegneria Informatica

Deterministic Policy Optimization:
an Approach to Safe Reinforcement Learning

Supervisor:

PROF. MARCELLO RESTELLI

Assistant Supervisor:

DOTT. MATTEO PAPINI

Master Graduation Thesis by:

PIETRO MELZI

Student Id n. 900217

Academic Year 2018-2019

To everyone who shared this journey with me.

RINGRAZIAMENTI

Desidero ringraziare il Prof. Marcello Restelli per avermi dato la possibilità di realizzare questa tesi. La sua competenza nel guidare il lavoro di ricerca e le sue abilità nel dare i giusti consigli sono state molto preziose per il conseguimento degli obiettivi prefissati.

Fondamentale è stato il contributo del Dott. Matteo Papini, desidero ringraziarlo per i validi spunti e suggerimenti che mi ha fornito in tutte le fasi di ricerca e di scrittura della tesi e per la disponibilità dimostrata in ogni occasione di confronto.

Ringrazio la mia famiglia che ha sempre condiviso le mie scelte ed è stata di grande aiuto in questi anni, Ashia Onlus che ha sostenuto economicamente i miei studi, gli amici e tutte le persone con cui ho condiviso questo percorso per la loro compagnia e il loro importante sostegno morale.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MDP Markov Decision Process

MDPs Markov Decision Processes

RL Reinforcement Learning

DP Dynamic Programming

BMDP Bounded-parameter MDP

PGPE Policy Gradients with Parameter-Based Exploration

DPG Deterministic Policy Gradient

DDPG Deep Deterministic Policy Gradient

MC Monte Carlo

TD Temporal Difference

IVI Interval Value Iteration

LC Lipschitz Continuous

TV Total Variation

PS Policy Search

RBF Radial Basis Functions

PG Policy Gradient

SPG Safe Policy Gradient

TDL Target Distribution Learning

DPO Deterministic Policy Optimization

RMSE Root Mean Square Error

BBO black-box optimization

LQR Linear-Quadratic Regulator

ABSTRACT

In reinforcement learning, policy optimization algorithms normally rely on action randomization to make the learning problem easier and to guarantee a sufficient exploration of all the possible situations in the task. Action randomization allows to execute and evaluate a wide range of actions that otherwise may be neglected by the algorithm. However, this practice may be unacceptable in real-life applications, such as industrial ones, where safety is a concern and deviations from usual behavior are not welcome by stakeholders. There exist multiple and not exclusive definitions of safety in reinforcement learning, hence safety aspects can be modeled and incorporated in the tasks in different ways. We consider the challenging scenario in which a learning agent is deployed in the real world and must be able to improve on-line without performing any random action, to ensure safe exploration throughout the learning process. For the first time, to the best of our knowledge, we propose a truly deterministic policy optimization algorithm for continuous domains. To design this algorithm, we require the validity of some assumptions on the regularity of the environment, which we deem easy to satisfy in the scenarios of interest. We also use state aggregation to build an abstract model of the environment and exploit passive exploration, necessary to allow successful policy optimization. The proposed approach is tested on simulated continuous control tasks, both in the case of learning from scratch and in the case of having some prior knowledge of the problem. The results obtained from the experiments are promising and encourage the future development of the techniques presented in this work.

ESTRATTO

L'apprendimento per rinforzo è un insieme di tecniche di apprendimento automatico che permettono a un agente autonomo che interagisce con un ambiente di imparare il miglior comportamento possibile, valutato rispetto al riscontro che l'agente riceve dall'ambiente. Nell'apprendimento per rinforzo, gli algoritmi che permettono di ottenere il miglior comportamento possibile solitamente richiedono all'agente di eseguire delle azioni casuali per facilitare la risoluzione del problema garantendo una sufficiente esplorazione delle possibili situazioni in cui l'agente si può trovare. Far eseguire azioni casuali all'agente permette di valutare un vasto numero di azioni, che altrimenti verrebbero trascurate dall'algoritmo di apprendimento. Tuttavia, questa tecnica può essere considerata inaccettabile in applicazioni reali dell'apprendimento per rinforzo, ad esempio nel campo industriale dove la sicurezza è un requisito importante ed è consigliabile evitare qualsiasi variazione dal comportamento usuale dell'agente. Ci sono molte possibilità, non esclusive tra loro, per definire il concetto di sicurezza in un problema di apprendimento per rinforzo. Per questo motivo, la letteratura propone numerose tecniche che affrontano la questione della sicurezza con approcci tra loro diversi. Noi consideriamo uno scenario in cui l'agente interagisce con un ambiente del mondo reale e deve imparare un comportamento ottimale senza poter effettuare alcuna azione casuale. Questa limitazione permette all'agente di esplorare l'ambiente in sicurezza durante l'apprendimento del comportamento ottimale. Per la prima volta, al meglio della nostra conoscenza, viene proposto un algoritmo di ottimizzazione del comportamento dell'agente in ambiente continuo che esegue solamente azioni deterministiche. Questo algoritmo richiede che siano valide alcune assunzioni sulla regolarità dell'ambiente, necessarie per avere una stima delle situazioni non più osservabili in mancanza di azioni casuali. Nelle applicazioni per cui l'algoritmo è stato pensato, consideriamo realistiche queste assunzioni. In questo algoritmo, inoltre, viene utilizzata la tecnica dell'aggregazione degli stati per costruire un modello astratto dell'ambiente e sfruttare una forma di esplorazione passiva, necessaria per migliorare il comportamento dell'agente. Abbiamo testato il metodo proposto in simulazioni di problemi di controllo continuo, nei casi di totale assenza di informazioni riguardo al problema e disponibilità di una conoscenza preliminare. Nei problemi considerati, affrontiamo un crescente livello di difficoltà dell'ambiente e della rappresentazione dello stato dell'agente: a partire da un ambiente deterministico monodimensionale, consideriamo prima un am-

biente stocastico bidimensionale e poi un ambiente deterministico a nove dimensioni, rappresentato dai sensori con cui l'agente è equipaggiato. Negli esperimenti effettuati, nonostante i limiti imposti per ragioni di sicurezza sull'esplorazione, l'agente impara un comportamento ottimale che è equiparabile a quello appreso con altri algoritmi esistenti, dove la questione della sicurezza non è considerata. Questo documento contiene un'introduzione al lavoro svolto che presenta le motivazioni alla base di esso, una parte teorica relativa all'apprendimento per rinforzo e allo stato dell'arte, la presentazione dell'algoritmo, la sua giustificazione teorica e la descrizione degli esperimenti con cui l'algoritmo è stato testato. I risultati ottenuti dagli esperimenti sono promettenti e incoraggiano uno sviluppo futuro delle tecniche presentate in questo lavoro.

1 INTRODUCTION

Reinforcement Learning (RL) is a machine learning field that includes techniques that allow an agent to learn the best possible behavior from its interaction with the environment. In RL, the agent explores different situations and evaluates their effectiveness according to the rewards it receives from the environment. RL techniques are applied in numerous real-world domains, such as recommender systems, computer systems, energy, finance, healthcare, robotics, transportation. Details on specific applications are discussed in [Li, 2019]. One of the challenges that arise in RL is the trade-off between exploration and exploitation. To obtain a high reward, an agent must perform actions that it has found to be effective in producing reward. However, to discover such actions, it has to try actions it has not selected before [Sutton and Barto, 2018]. An intuitive strategy to address this issue is to perform random actions at the beginning of the task, so as to test as many actions as possible in a first phase and then exploit the most rewarding actions in the long term. By executing random actions in any state, the agent potentially executes and evaluates the entire range of feasible behaviors it can undertake. Such an approach strongly supports exploration and provides future advantages to the agent. If the agent learns within a simulator there are no concerns related to this strategy. Unfortunately, a simulator is not always available: when knowledge of the environment is lacking or too much complexity is required to faithfully reproduce the real system, the agent has to learn directly in the real world. When the agent interacts with a real-life environment, serious drawbacks may come up. For instance, the randomness of the action selections allows the execution of unwanted actions in some specific situations.

1.1 MOTIVATIONS

A lack of control over the agent's behavior in the real world can lead to dangerous actions that may damage the agent's hardware or, even worse, harm people that operate in the environment. Partly because of this issue raised by safety concerns, RL techniques are scarcely used in fields where they could otherwise provide outstanding benefits as industrial robotics, surgery [Baek et al., 2018] or autonomous driving. Again, another important scenario in which the randomness of actions is unwanted and deemed dangerous is finance: certainly nobody wants to perform stochastic operations involving their own money, which is different than making bets in an informed way. In

all of these fields, the effectiveness of RL techniques is concealed by the willingness to ensure a predictable agent's behavior. Indeed, even a single execution of a random and unsafe action may cause a failure in the task or harm the environment, which is often less acceptable than the consequences of epistemic uncertainty or an aleatory environment. According to [García and Fernández, 2015], the latter can be referred to as *inherent uncertainty* and it is unavoidable, the former is called *parameter uncertainty* and it is due to the lack of knowledge of the environment. Parameter uncertainty can be reduced by performing random actions that explore the environment but introduce a new source of uncertainty: *agent uncertainty*. Several definitions of safety have been proposed in RL (we discuss them in Section 3.2). These definitions involve different aspects of safety, such as the variance in the reward signal received by the agent or the ability to avoid the exploration of dangerous regions of the environment, however, the safety problem due to the randomness of the actions is poorly addressed. Moreover, many techniques in safe RL ensure convergence to a safe behavior but do not set any constraint on intermediate solutions. In RL the stochasticity on actions, widely used for intermediate solutions, appears to be essential for successful learning. Indeed, if the agent always performs the same action in every state, it may not find any better action to improve its behavior. In this work, we want to contradict the idea that stochastic behavior is fundamental to the agent's learning process.

1.2 GOAL

Driven by these motivations, we explore an alternative strategy, which ensures the safeness of the agent-environment interaction in every instant by forcing the explorative behavior of the agent to be deterministic. In particular, exploration in the environment is performed according to the same deterministic policy that the agent is learning, in order to perfectly monitor the agent's behavior. In addition, we opt for risk-averse approaches to implement specific parts of our strategy in order to strengthen the concept of safeness. Several algorithms that learn a deterministic policy have been proposed in the literature, most of which are suitable for environments with a finite number of feasible actions. In this work, we consider environments with continuous (possibly infinite) actions, also for them there are algorithms that learn deterministic policies. Two of these algorithms, Policy Gradients with Parameter-Based Exploration (PGPE) [Sehnke et al. (2008), subsection 3.1.2] and Deterministic Policy Gradient (DPG) [Silver et al. (2014), subsection 3.1.2], have been considered in this work for a comparison with our algorithm, called Deterministic Policy Optimization (DPO). Anyhow, to the best of our knowledge, all existing algorithms in continuous RL involve the execution of random actions.

1.3 CONTRIBUTIONS

The solution we propose is free from this issue and it is suitable for regular environments, i. e. environments where performing the same action in similar states produces similar effects. In these environments, once an action is executed in a certain state, we can evaluate its effect in other (similar) states, without necessarily redoing the action. This assumption provides a sort of passive exploration that the agent can exploit, instead of relying on the stochasticity of its actions. As a result, in any state, the agent performs only a deterministic action and evaluates, within a certain precision, all the actions that have been executed in similar states. Because of this, only a subset of all feasible actions is evaluated in every state, therefore the learning ability of the agent may be significantly reduced in this approach: learning an optimal behavior may no longer be feasible or require an unreasonable amount of time. On the other hand, this approach allows the agent to learn and improve its behavior without the necessity of performing any random action. If the environment is partially observable or we consider multi-agent contexts, the optimal behavior may require stochasticity on actions. However, we do not consider these settings in the work. We tested our algorithm on simulated continuous control tasks obtaining promising results.

1.4 OUTLINE

In this document we present our work according to the following structure: Chapter 2 provides the theory (and notation) related to Reinforcement Learning (RL) and Markov Decision Process (MDP), Chapter 3 illustrates the state of the art in Policy Gradient (PG) and safe RL from which we started to develop our algorithm, Chapter 4 describes the DPO algorithm, providing details on the implementation of its different building blocks, Chapter 5 contains a theoretical analysis of our approach, in Chapter 6 the performed experiments are reported and discussed, at last Chapter 7 concludes the thesis by providing some hints for the future development of this work.

2 PRELIMINARIES ON REINFORCEMENT LEARNING

According to [Sutton and Barto, 2018], Reinforcement Learning (RL) consists in learning what to do so as to maximize a numerical reward signal. The learner must discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and all subsequent rewards. The problem of RL is formalized with mathematical tools coming from dynamical systems theory, specifically with the Markov Decision Processes (MDPs) that we detail in Section 2.1.

Chapter 2 provides a strong theoretical background for our work. After the introduction on MDPs in Section 2.1, Section 2.2 addresses the *Tabular Methods* used to solve MDPs. Even if the application of these methods is limited by computational costs, they provide basic concepts in RL. Section 2.3 explores *Policy Search (PS)*, a class of methods that solve RL problems when *Tabular Methods* are infeasible. PS offers several advantages in the robotic field (described in Section 2.3) and for this reason it is widely used there. Then, we present two special types of MDPs in Section 2.4: *Bounded MDPs*, suitable to model uncertainty in the environment, and *Lipschitz MDPs*, suitable to model regularity in the environment. Finally, Section 2.5 presents some topics related to state discretization that will be useful for the description of our approach.

2.1 MARKOV DECISION PROCESSES

MDPs are a mathematical framework used for modeling RL problems in which an *agent*, by interacting with an *environment*, learns how to achieve a goal. MDPs are a formalization of sequential decision making, where the decision taken by the agent influences immediate and future rewards. The agent interacts with the environment by selecting the actions to perform and receiving rewards and information on the new situation. Rewards are provided by the environment in the form of scalar values and the agent aims to maximize the sum of rewards over time.

Definition 2.1 (MDP)**.** A Markov Decision Process (MDP) is described by a six-tuple $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, p_0 \rangle$, where:

- $\mathcal{S}$ is the state space, with $\mathcal{S} \subseteq \mathbb{R}^N$.
- $\mathcal{A}$ is the action space, with $\mathcal{A} \subseteq \mathbb{R}^D$.

- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, with $P(s'|s,a)$ denoting the probability of reaching state s' from state s by taking action a. $P(\cdot|s,a)$ is a distribution of probability on the arriving state, then for any state-action pair (s,a), the following equality holds:

$$\sum_{s' \in \mathcal{S}} P(s'|s,a) = 1 \tag{2.1}$$

  and $P(s'|s,a) \geqslant 0 \quad \forall s, s' \in \mathcal{S}, a \in \mathcal{A}$.

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, with $R(s,a)$ denoting the expected reward from taking action a in state s. Usually the reward function is bounded with a finite value $\overline{R}$ such that $|R(s,a)| \leqslant \overline{R} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$.

- $\gamma \in [0,1)$ is the discount factor, used to discount the present effect of future rewards.

- $p_0 \in \Delta(\mathcal{S})$ is the initial-state distribution.

The transition function P defines the dynamics of the MDP and satisfies the *Markov property*: the probability of reaching s_t depends only on the immediately previous state and action, s_{t-1} and a_{t-1}, and not on earlier states and actions.

The interaction between agent and environment can be better explained with symbols: at each time step t the agent receives a representation of the state $s_t \in \mathcal{S}$ and on that basis it selects an action $a_t \in \mathcal{A}$. One time step later, the agent receives a reward r_{t+1} and finds itself in a new state $s_{t+1} \in \mathcal{S}$. We have defined the MDP following [Puterman, 2014]. In general, the state space $\mathcal{S}$ and the action space $\mathcal{A}$ can be finite or continuous sets. We focus on continuous-space MDPs as these are the most suitable for modeling continuous control problems.

2.1.1 *Policy and Value Functions*

The behaviour of the agent is modeled with a *policy* $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, i. e. a mapping from states to probabilities of selecting each possible action. The agent should learn a policy according to its goal of maximizing the sum of rewards collected during the task. Specific details on how to learn policies that achieve this goal are given later, in Section 2.2 and Section 2.3. The sum of rewards obtained by the agent, if we consider the time steps $k \in (t, T]$, where the time step T represents the horizon of the task, is called *return* G_t. In general, the return is defined as a sum of discounted rewards:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k, \tag{2.2}$$

where r_k is the reward obtained at step k. The horizon T can be finite or infinite. In the first case, the task is said to be *episodic* and the agent-environment interaction breaks naturally into episodes. In the second case (i.e. $T = \infty$), the task is said to be *continuing* and a discount factor $\gamma < 1$ is required in order to obtain a return $G_t < \infty$.

In this work, we denote with $\pi(a|s)$ the probability of performing the action a in the state s, according to the policy π. Since $\pi(s)$ is a distribution of probability, the following equality holds:

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1 \quad \forall s \in \mathcal{S} \tag{2.3}$$

and $\pi(a|s) \geqslant 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$. If the set of actions $\mathcal{A}$ is finite and for each state $s \in \mathcal{S}$ there exists an action a such that $\pi(a|s) = 1$, the policy is deterministic. If the set of actions $\mathcal{A}$ is continuous and for each state $s \in \mathcal{S}$ the probability distribution $\pi(s)$ is a Dirac delta function, the policy is deterministic. In the case of deterministic policies, $\pi(s)$ identifies the action prescribed by π when the agent is in state s.

Given a policy π, it is possible to compute, according to the policy, the *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$. This is a widely used function in RL that measures how good it is for the agent to be in a given state $s \in \mathcal{S}$, according to the expected return obtainable from that state. Since the rewards that the agent can expect to receive in the future depend on the actions taken in any state, the value function is defined with respect to policies: $V^\pi(s)$ is the expected sum of discounted rewards that the agent collects by starting at state s and following policy π. The value function can be defined recursively via the Bellman equation:

$$V^\pi(s) = \int_{\mathcal{A}} \pi(a|s) \Big(R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) V^\pi(s') \, ds' \Big) \, da. \tag{2.4}$$

For control purposes, we can also define an action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$Q^\pi(s,a) = R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) \int_{\mathcal{A}} \pi(a'|s') Q^\pi(s',a') \, da' \, ds'. \tag{2.5}$$

$Q^\pi(s,a)$ represents the expected return obtained from taking the action a in state s and then following the policy π.

Finally, we denote with

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.6}$$

the advantage function of policy π, that represents the advantage in terms of value functions given by performing action a in state s, instead of the action prescribed by $\pi(s)$.

2.1.2 *Performance Measure*

In order to evaluate how good a policy π is, we consider the expected return obtained starting from a state $s \in \mathcal{S}$ drawn from the initial-state distribution p_0 and following the policy π. The *performance measure* $J(\pi)$ expressed in the form of an expected value is:

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim p_0}[G_0] \quad = (1-\gamma)^{-1} \mathop{\mathbb{E}}_{s \sim \delta^\pi, a \sim \pi}[R(s,a)] \quad = \mathop{\mathbb{E}}_{s \sim p_0}[V^\pi(s)], \tag{2.7}$$

where δ^π is the γ-discounted future-state distribution. This function is defined as:

$$\delta^\pi(s) = (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim p_0} \sum_{t=0}^{\infty} \gamma^t P^\pi(S_t = s | S_0 = s_0) \tag{2.8}$$

and represents the probability of being in a certain state s during the execution of the task, provided that the policy is π and the initial state distribution is p_0.

The performance measure is used to identify the optimal policy π^* that we want to learn in the RL problem as:

$$\pi^* \in \arg\max_\pi J(\pi). \tag{2.9}$$

To be precise, in RL the optimal policy π^* is required to maximize the value function (defined as in equation (2.4)) for each state $s \in \mathcal{S}$. The methods described in subsection 2.2.1 allow to obtain optimal policies that satisfy this property. Instead, the set of optimal policies obtained from (2.9) includes policies whose value function is not maximum in every state. The criterion used to identify optimal policies in equation (2.9) is weaker[1] than the one used in subsection 2.2.1, however it is appropriate in our work.

2.1.3 *Properties of RL techniques*

RL algorithms require exploration to learn the optimal policy π^*. Exploration is provided by the agent that evolves in the MDP according to the current policy π and collects information about states, actions and rewards at each time step t, until the (possibly infinite) horizon T of the episode. All the information gathered by the agent is represented by a tuple called *trajectory*: $\langle s_0, a_0, r_1, s_1, a_1, ..., s_{T-1}, a_{T-1}, r_T \rangle$,

1 The performance measure J depends on the initial state distribution p_0. We consider, for instance, an MDP whose state space $\mathcal{S}$ is composed of two regions such that it is not possible to reach one region from the other. If the probability of being in one of these two regions is equal to zero (according to p_0), the policies maximizing J are the ones that maximize J in the visited states, regardless of their performance in the unvisited region.

where $s_0 \sim p_0$, $a_t \sim \pi(\cdot|s_t)$, $r_t = R(s_t, a_t)$, $s_{t+1} \sim P(\cdot|s_t, a_t)$. If the task is episodic, usually the agent collects a *batch* of N different trajectories with the same policy π before updating it. The number N of trajectories is called *batch size*.

The policy used to generate trajectories can be different from the policy that the RL technique learns. We can distinguish between:

- *On-policy algorithms:* the policy used to interact with the environment is the policy that is being learnt;
- *Off-policy algorithms:* the policy used to interact with the environment is different from the policy that is being learnt. It is called *behavioral policy*.

Another differentiation, important for our work, is between:

- *Model-based algorithms:* the agent knows or estimates the model of the environment, as in the algorithm we present;
- *Model-free algorithms:* the agent has no knowledge on the model of the environment.

2.2 TABULAR SOLUTION METHODS

Solving a RL task means finding a policy that maximizes the return obtained by the agent in the task. First, we consider finite MDPs, a subset of the MDPs defined in 2.1. In order to reason with finite MDPs, we consider the state space $\mathcal{S}$ and the action space $\mathcal{A}$ as finite sets of discrete values and we replace all the integrals appearing in the expressions reported so far with summations. The methods used to solve problems involving finite MDPs are called *tabular* because the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are small enough for the value functions to be represented in a tabular format.

Policies can be partially ordered according to their value function:

$$\pi' \succcurlyeq \pi \iff V^{\pi'}(s) \geqslant V^{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{2.10}$$

In finite MDPs, there always exists a deterministic optimal policy π^* such that $\pi^* \geqslant \pi \quad \forall \pi \in \Pi$, said Π the set of policies. The value function V^* computed according to π^* has the following property:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}. \tag{2.11}$$

Optimal value function V^* and optimal action-value function Q^* can be written with the Bellman optimality equations:

$$V^*(s) = \max_{a} \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right) \tag{2.12}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a'). \tag{2.13}$$

If the agent has a *complete knowledge* of the environment, i. e. the agent knows the transition function P and the reward function R of the MDP M representing the environment, Dynamic Programming (DP) can be used to solve the RL problem. Details are provided in subsection 2.2.1. Instead, if the agent has an *incomplete knowledge* of the environment, the unknown functions P and R of MDP M can be estimated from experience. Details are provided in subsection 2.2.2.

When the problems involve finite MDPs with a larger state space or continuous MDPs, tabular methods are no more suitable because value functions cannot be represented as tables. Two main approaches are possible:

- the value functions can be estimated with *function approximators* and used to solve the task;
- the optimal policies can be directly searched in the space of policies, without the necessity of computing any value function. Details are provided in Section 2.3.

2.2.1 *Dynamic Programming*

Dynamic Programming (DP) is a collection of algorithms that can be used to compute optimal policies, given a model of the environment in the form of an MDP. Since DP is expensive and requires finite MDPs, its utility in solving RL problems is limited. However DP provides strong foundations for understanding the more advanced methods. DP offers two algorithms that compute the optimal value functions: *policy iteration* and *value iteration*.

POLICY ITERATION: In policy iteration, two consecutive operations are performed on a policy π in order to obtain a policy π' that is better according to the ordering rule in (2.10). These operations are called *policy evaluation* and *policy improvement* and the goal of the algorithm is to obtain the optimal value function V^* through the sequence:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V^*, \tag{2.14}$$

where $\pi_i \xrightarrow{E} V^{\pi_i}$ indicates that policy evaluation is performed on policy π_i to calculate the value function V^{π_i} and $V^{\pi_i} \xrightarrow{I} \pi_{i+1}$ indicates that policy improvement is performed from policy π_i and its value function V^{π_i} to obtain the policy π_{i+1}.

Policy evaluation consists in computing the value function V^π for

every state $s \in \mathcal{S}$, according to the current policy π, by iteratively applying the following updating rule:

$$V_{k+1}(s) = \sum_a \pi(a|s)\Big(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')\Big), \tag{2.15}$$

until $V_{k+1}(s) = V_k(s) \quad \forall s \in \mathcal{S}$. When it happens, the convergence to V^π is obtained. The existence and uniqueness of V^π are guaranteed as long as either $\gamma < 1$ or eventual termination is guaranteed from all states under the policy π [Sutton and Barto, 2018]. Formally policy evaluation converges in the limit of infinite iterations, in practice the algorithm stops when:

$$\max_{s \in \mathcal{S}} |V_{k+1}(s) - V_k(s)| \leqslant \epsilon, \tag{2.16}$$

where ϵ is a fixed threshold. The value function V^π is approximated with the computed value function V_{k+1}.

Policy improvement consists in an update of policy π in a new deterministic policy π', according to the value function V^π computed in the previous policy evaluation. For each state $s \in \mathcal{S}$, indeed, the policy π' is computed according to the following rule:

$$\pi'(s) = \arg\max_a \Big(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^\pi(s')\Big). \tag{2.17}$$

By contruction $V^{\pi'}(s) \geqslant V^\pi(s) \quad \forall s \in \mathcal{S}$, then the policy π' must be as good as or better than π [Sutton and Barto, 2018]. When $\pi'(s) = \pi(s) \quad \forall s \in \mathcal{S}$ or π' is as good as π (i.e. $V^\pi = V^{\pi'}$), the algorithm terminates. The policy π' is optimal, hence $\pi^* = \pi'$.

Because a finite MDP has a finite number of policies, the entire process is ensured to converge to an optimal policy π^* and optimal value function V^* in a finite number of iterations.

VALUE ITERATION: The main drawback in policy iteration is that, for each policy evaluation, multiple iterations of the updating rule (2.15) are required. If we truncate the policy evaluation after just one application of the updating rule, we obtain value iteration. In value iteration, the value function at each step is updated for all $s \in \mathcal{S}$ as follows:

$$V_{k+1}(s) = \max_a \Big(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')\Big). \tag{2.18}$$

Differently from policy iteration, we don't compute any intermediate policy π_i. However, the sequence $\{V_k\}$ of the value functions converges to V^* since value iteration is a special case of policy iteration. The optimal policy π^* is obtained applying (2.17), with V^* instead of

V^π.

For computational reasons, value iteration is implemented with the *stopping condition* showed in equation (2.16). V^* is approximated with the last value function computed in the truncated sequence $\{V_k\}$.

2.2.2 *Uncomplete Knowledge of MDPs*

In many cases, however, it is not possible to apply DP. Some alternatives are introduced below for sake of completeness. We don't detail these methods because they are not related to our work.

TABULAR ALTERNATIVES TO DP: If we do not have a complete knowledge of the environment, we can learn the unknown dynamics from experience, i. e. from the samples collected by the agent while interacting with the environment. The samples are tuples $\langle s, a, r, s' \rangle_t$, with $s, s' \in \mathcal{S}, a \in \mathcal{A}$ and $r = R(s, a)$. They contain information related to the agent-environment interaction at time step t. The two main classes of algorithms are Monte Carlo (MC) and Temporal Difference (TD). MC methods involve episodic tasks, TD learning involves continuing tasks.

APPROXIMATE SOLUTION METHODS: The methods presented so far are not suitable to solve RL problems that involve a very large or infinite state space, such as the tasks in which a robot is free to move in a wide area and its state is represented by several (possibly continuous) dimensions. In these problems we cannot expect to obtain the optimal policy, and our goal is to find a good approximate solution using limited computational resources. In an environment with a continuous state space $\mathcal{S}$, the agent will always visit states never seen before. Then, the experience we gather gives information on a subset of states but we need to generalize it to all the state space $\mathcal{S}$. The generalization comes up in the form of *function approximations*. These functions are estimated from samples and approximate value functions over continuous state space $\mathcal{S}$ in place of using tables. Usually a function approximation is a parameterized function $f^w : \mathcal{S} \to \mathbb{R}$, with parameter $w \in \mathbb{R}^d$.

Several methods rely on value function approximation in order to provide the solution of the RL problem. However, function approximation brings new issues. First of all, convergence guarantees are difficult to obtain for *greedy policies*, i. e. policies for which the action to be performed in any state is the action having the highest value according to some value functions[2]. If the policy is greedy, an arbitrary

2 For instance, $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$, with $Q(s, a)$ a function approximation, is a greedy policy.

small change in the estimated value of an action can cause it to be, or not be, selected [Sutton et al., 1999]. Furthermore, approximated value functions can introduce a bias that prevents the method to converge even to a local optimum [Deisenroth, Neumann, and Peters, 2013] and increase the number of parameters to learn.

2.3 POLICY SEARCH

In contrast with value-based methods, Policy Search (PS) methods use parametrized policies π_θ, where $\theta \in \Theta$ and Θ is the parameter space. The set of parameter θ fixes in advance the candidate policies, i. e. the policy class $\Pi_\Theta = \{\pi_\theta | \theta \in \Theta\}$. PS methods directly operate in the parameter space Θ (i. e. in the set of candidate policies) to find the optimal parametrized policy and typically do not need to learn a value function. PS copes with high dimensional state space $\mathcal{S}$ and action space $\mathcal{A}$ and offers better convergence guarantees compared to the methods that involve value-function approximation. PS methods are more robust to noise because they are able to prevent a small variation in the state to produce a completely different action (this can happen in greedy policies, for instance). PS methods can also incorporate domain knowledge in the policy definition (for instance, in the motor primitive policies defined in [Peters and Schaal, 2008]) and can be made safe by design. Because of these reasons, PS methods have been found to be suitable in robotic applications.

Most of the algorithms in PS are *model-free* (i. e. they don't require a model of the environment) because directly learning a policy is often easier than learning an accurate model. These methods update the policy directly exploiting the sampled trajectories, hence it is important to define an exploration strategy that provides variety in trajectories. Exploration can be performed both in the action space and in the parameter space. The former one can be implemented by adding a noise ϵ directly to the executed actions, the noise is generally sampled from a zero-mean Gaussian distribution. The latter consists in a perturbation of the parameter vector θ of π_θ. The magnitude of noise present in any kind of exploration depends on some parameters. These parameters can also be updated by the algorithms: usually the size of exploration is gradually decreased to fine tune the policy parameters. [Deisenroth, Neumann, and Peters, 2013].

The principal class of algorithms in PS is composed by Policy Gradient (PG) methods. We detail this class in Section 3.1.

2.3.1 *Policy Representations*

In our work, we focus on the optimization of deterministic parametric policies of the form $\pi_\theta : \mathcal{S} \to \mathcal{A}$, with $\theta \in \Theta \subseteq \mathbb{R}^{m \times D}$. We will often abbreviate π_θ as θ in subscripts and function arguments, e. g. $V^\theta \equiv V^{\pi_\theta}$, $J(\theta) \equiv J(\pi_\theta)$. The simplest way of parametrizing π_θ is by means of a linear mapping. The linear policy is defined as:

$$\pi_\theta(s) = \theta^T \boldsymbol{\phi}(s), \tag{2.19}$$

where $\theta \in \mathbb{R}^{m \times D}$ and $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^m$ is a feature function. This can be the state itself or, for instance, a set of Radial Basis Functions (RBF). An example of RBF is the Gaussian

$$\phi_i(s; \mu_i, \sigma_i) = \exp\left\{-(s - \mu_i)^2/(2\sigma_i^2)\right\}, \tag{2.20}$$

where μ_i and σ_i are hyperparameters of the feature function ϕ_i, $i = 1, \ldots, m$. According to the parameters learnt by the algorithm, we can distinguish between:

- *Shallow policies:* the hyperparameters included in the feature functions are fixed throughout the algorithm, only the parameter θ from π_θ is learnt by the algorithm;
- *Deep policies:* both the hyperparameters included in the feature functions and the parameter θ from π_θ are learnt by the algorithm.

More complex policy parametrizations include deep neural networks [Duan et al., 2016]. Stochastic policies randomize over actions.

SOFTMAX POLICIES: A possible policy parameterization for PS methods is the one that uses the *softmax function*. The policies defined according to the softmax function are also called *Gibbs policies*. This kind of policies is mostly used when the action set $\mathcal{A}$ is discrete because, in order to define the probability $\pi(a|s)$, it is required to consider all the feasible actions in the state s:

$$\pi_\theta(a|s) = \frac{\exp\left(\theta^T \phi(s, a)\right)}{\sum_{a_k \in \mathcal{A}(s)} \exp\left(\theta^T \phi(s, a_k)\right)}, \tag{2.21}$$

where $\mathcal{A}(s)$ is the set of actions that can be performed in state s and $\phi(s, a)$ is a feature function depending both on state and action.

NORMAL POLICIES: A common parametrization for countinuous actions represented by real numbers, is the normal distribution. The policy can be defined as the normal probability density over a scalar

action, with mean μ and standard deviation σ given by parametric function approximators that depend on the state:

$$\pi_\theta(a|s) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\Big(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}\Big), \tag{2.22}$$

where $\mu(s,\theta) = \theta_\mu^\intercal \phi(s)$ and $\sigma(s,\theta) = \exp\left(\theta_\sigma^\intercal \phi(s)\right)$. The policy's parameter vector is $\theta = [\theta_\mu, \theta_\sigma]^\intercal$. This is only a possible parameterization for the standard deviation σ of the normal distribution, in this configuration the exploration is said to be *heteroscedastic* because the standard deviation changes according to the state. Usually the standard deviation σ of the normal distribution is represented by a single parameter that does not depend on the state.

2.4 SPECIAL MARKOV DECISION PROCESSES

In this section we present two special classes of MDPs that allow us to represent some properties of interest for the MDPs considered in this work.

2.4.1 *Bounded MDPs*

A Bounded-parameter MDP (BMDP) [Givan, Leach, and Dean, 2000] is a five-tuple $\langle \mathcal{S}, \mathcal{A}, P_\updownarrow, R_\updownarrow, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$ and γ are defined as for (finite) MDPs, and $P_\updownarrow, R_\updownarrow$ are analogous to the MDP transition and reward functions, but yield closed real intervals instead of real values: given a lower bound $\underline{P}$ and an upper bound $\overline{P}$, $P_\updownarrow = [\underline{P}; \overline{P}]$. Similarly we can specify the interval $R_\updownarrow$. This can be used to model uncertainty on the true nature of a decision process. To ensure that $P_\updownarrow$ admits only well-formed transition functions, we require that for any action a and state s, the sum of the lower bounds of $P_\updownarrow(s'|s,a)$ over all states s' must be less than or equal to one, while the upper bounds must sum to a value greater than or equal to one.

A BMDP $M_\updownarrow = \langle \mathcal{S}, \mathcal{A}, P_\updownarrow, R_\updownarrow, \gamma \rangle$ defines a set of exact MDPs. For any exact MDP $M = \langle \mathcal{S}', \mathcal{A}', P', R', \gamma' \rangle$, we have $M \in M_\updownarrow$ if $\mathcal{S} = \mathcal{S}', \mathcal{A} = \mathcal{A}', \gamma = \gamma'$, and for any action a and states s, s', $R'(s,a)$ belongs to the interval $R_\updownarrow(s,a)$ and $P'(s'|s,a)$ belongs to the interval $P_\updownarrow(s'|s,a)$. An interval value function $V_\updownarrow$ is a mapping from states to closed real intervals. We use such functions to indicate that the value of a given state for any exact MDP falls within the selected interval. As in the case of (exact) value functions, interval value functions are specified w.r.t. a fixed policy π, i. e. :

$$V_\updownarrow^\pi(s) = \left[\underline{V}^\pi(s), \overline{V}^\pi(s)\right] = \left[\min_{M \in M_\updownarrow} V_M^\pi(s), \max_{M \in M_\updownarrow} V_M^\pi(s)\right]. \tag{2.23}$$

As shown in [Givan, Leach, and Dean, 2000], $M_{\updownarrow}$ includes both an MDP that simultaneously achieves $\underline{V}^{\pi}(s)$ for all $s \in \mathcal{S}$ and another one that achieves $\overline{V}^{\pi}(s)$ for all $s \in \mathcal{S}$.

The notion of optimal value function in BMDPs requires an ordering rule for intervals. We can define two different possible orderings:

$$[l_1, u_1] \leqslant_{pes} [l_2, u_2] \Leftrightarrow \begin{cases} l_1 < l_2, \text{ or} \\ l_1 = l_2 \text{ and } u_1 \leqslant u_2 \end{cases} \tag{2.24}$$

$$[l_1, u_1] \leqslant_{opt} [l_2, u_2] \Leftrightarrow \begin{cases} u_1 < u_2, \text{ or} \\ u_1 = u_2 \text{ and } l_1 \leqslant l_2 \end{cases} \tag{2.25}$$

We use these orderings rules to partially order interval value functions in the following way:

$$V_{1\updownarrow} \leqslant V_{2\updownarrow} \iff V_{1\updownarrow}(s) \leqslant_* V_{2\updownarrow}(s) \quad \forall s \in \mathcal{S}, \tag{2.26}$$

with $\leqslant_*$ defined either as $\leqslant_{pes}$ in (2.24) or $\leqslant_{opt}$ in (2.25).

As stated in [Givan, Leach, and Dean, 2000], there exists at least one optimistically and one pessimistically optimal policy:

$$V^*_{\updownarrow opt} = \max_{\pi \in \Pi} V^{\pi}_{\updownarrow} \text{ using } \leqslant_{opt} \text{ to order interval value functions,}$$
$$V^*_{\updownarrow pes} = \max_{\pi \in \Pi} V^{\pi}_{\updownarrow} \text{ using } \leqslant_{pes} \text{ to order interval value functions.}$$

In order to better understand the meaning of the optimal policies, we consider a game in which we choose a policy π and then a second player chooses an MDP $M \in M_{\updownarrow}$ to evaluate the policy. $\overline{V}^*_{opt}$ is the best value function we can obtain if the second player cooperates in the game, $\underline{V}^*_{pes}$ is the best value function obtainable if the second player is an adversary.

INTERVAL VALUE ITERATION: In [Givan, Leach, and Dean, 2000] it is defined the Interval Value Iteration (IVI) algorithm that computes optimal value intervals, which is similar to the standard value iteration presented in subsection 2.2.1. In IVI the updating rule is:

$$V_{\updownarrow,k+1}(s) = \max_{a \in \mathcal{A}, \leqslant_*} \left[\min_{M \in M_{\updownarrow}} VI_{M,a}(\underline{V}_k)(s), \max_{M \in M_{\updownarrow}} VI_{M,a}(\overline{V}_k)(s) \right], \tag{2.27}$$

where $\leqslant_*$ is $\leqslant_{pes}$ or $\leqslant_{opt}$. Given an MDP M with transition function P and reward function R, an action $a \in \mathcal{A}$ and a value function v, $VI_{M,a}(v)(s)$ is a single iteration of policy evaluation, where the action is fixed and

$$VI_{M,a}(v)(s) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s'). \tag{2.28}$$

In (2.27) it is required to perform two iterations of VI. The two value functions used to perform the iterations (i. e. the functions used instead of v in (2.28)) are obtained from the interval value function $V_{\updownarrow,k}$. For the first iteration, $v = \underline{V}_k$, with $\underline{V}_k$ being the lower bounds in $V_{\updownarrow,k}$; for the second one $v = \overline{V}_k$ with $\overline{V}_k$ being the upper bounds in $V_{\updownarrow,k}$.

Instead of searching in the set $M_{\updownarrow}$, the MDP M that minimizes (maximizes) the expression contained in min(max) operator in (2.27) can be directly obtained by computing an exact transition function P from the interval transition function $P_{\updownarrow}$ of $M_{\updownarrow}$. In order to do that, the arriving states $s' \in \mathcal{S}$ are sorted in increasing (decreasing) order according to their value $\underline{V}(\overline{V})$. Then, for all the state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ and given an ordering of arriving states $s'_1, s'_2, ..., s'_k$, we calculate the index r, with $1 \leqslant r \leqslant k$, that maximizes the following expression without letting it exceed 1:

$$\sum_{i=1}^{r-1} \overline{P}(s'_i|s, a) + \sum_{i=r}^{k} \underline{P}(s'_i|s, a). \tag{2.29}$$

The exact transition function $P(\cdot|s, a)$ is defined by assigning the upper bound $\overline{P}(s'|s, a)$ to the transition probabilities involving states s' with an index lower than r, the lower bound $\underline{P}(s'|s, a)$ to the transition probabilities involving states s' with an index greater than r and the probability that ensures $\sum_{s' \in \mathcal{S}} P(s'|s, a) = 1$ to the state with index r.

According to [Givan, Leach, and Dean, 2000], the IVI algorithm is able to converge to $V^*_{\updownarrow opt}$ or $V^*_{\updownarrow pes}$ in a polynomial number of iterations, where polynomial is relative to the problem size.

2.4.2 *Lipschitz MDPs*

We introduce the notion of *Lipschitz continuity* in order to define some properties of regularity in the MDP. These properties can be exploited in policy gradient algorithms, for instance in [Pirotta, Restelli, and Bascetta, 2015] it is shown how they can ensure a performance improvement at each iteration of policy-parameter updates. In this section we provide basic concepts related to Lipschitz continuity and useful bounds that will be exploited in our work.

Definition 2.2 (Lipschitz continuity)**.** Given two metric spaces (X, d_X) and (Y, d_Y) where d_X and d_Y denote the corresponding metric functions, a function $f : X \to Y$ is called L_f-Lipschitz Continuous (LC) if

$$\forall (x_1, x_2) \in X^2, d_Y(f(x_1), f(x_2)) \leqslant L_f d_X(x_1, x_2). \tag{2.30}$$

The smallest L_f for which (2.30) holds is called *Lipschitz constant* of f. For real-valued functions (e. g. the reward function R), we use the Euclidean distance as metric for the codomain distance. The same metric can be used to measure the distance between states and actions, as long as they can be represented in a numeric (possibly multidimensional) format. For the transition function P and the stochastic policies π we need to introduce a distance between probability distributions. We consider the Kantorovich or L^1-Wasserstein metric on probability measures, defined as follows:

Definition 2.3 (Kantorovich metric)**.** Given two probability measures p and q, the Kantorvich measure $\mathcal{K}(p, q)$ is:

$$\mathcal{K}(p, q) = \sup_f \left\{ \left| \int_x f(x) d\Big(p(x) - q(x)\Big)\, dx \right| : L_f \leqslant 1 \right\}. \tag{2.31}$$

In the case of deterministic policies, the probability distributions p and q are Dirac delta functions. The Kantorovich distance between two Dirac delta functions is equal to the distance of their locations. As an alternative to the Kantorovich metric, we can consider the Total Variation (TV) distance, defined as:

Definition 2.4 (Total Variation distance)**.** Given two probability measures p and q, the TV distance $TV(p, q)$ is:

$$TV(p, q) = \frac{1}{2} \int_x \big|p(x) - q(x)\big|\, dx. \tag{2.32}$$

This metric is more demanding than Kantorovich metric: MDPs that are Lipchitz according to TV are also Lipschitz according to the Kantorovich metric but not vice versa.

In this work, it is also important to consider the *Hausdorff metric* to measure the distance of two subsets of a metric space. We use it in the abstract state space since each abstract state X is a subset of the state space $\mathcal{S}$.

Definition 2.5 (Hausdorff metric)**.** Let X and Y be two non-empy subsets of a metric space (M, d). The Hausdorff distance $d_H(X, Y)$ is defined as:

$$d_H(X, Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y) \right\}. \tag{2.33}$$

In this work, we restrict our attention to Lipschitz-continuous MDPs and policies. We make similar assumptions as in [Pirotta, Restelli, and Bascetta, 2015]. For the MDP, we require both the continuity of the transition model and the reward:

Assumption 1 (Lipschitz MDP)**.** *For all* $s, \widetilde{s} \in \mathcal{S}$ *and* $a, \widetilde{a} \in \mathcal{A}$*:*

$$\mathcal{K}\left(P(\cdot|s, a), P(\cdot|\widetilde{s}, \widetilde{a})\right) \leqslant L_P\, d_{\mathcal{S}\mathcal{A}}\left((s, a), (\widetilde{s}, \widetilde{a})\right), \tag{2.34}$$

$$|R(s,a) - R(\widetilde{s},\widetilde{a})| \leqslant L_R \, d_{\mathcal{SA}}\left((s,a),(\widetilde{s},\widetilde{a})\right), \tag{2.35}$$

for some positive real constants L_P *and* L_R.

where $d_{\mathcal{SA}}\left((s,a),(\widetilde{s},\widetilde{a})\right) = \|s-\widetilde{s}\| + \|a-\widetilde{a}\|$ is the taxicab norm on $\mathcal{S}\times\mathcal{A}$. We also require our policy to be continuous both w.r.t. the input state and its parameters:

Assumption 2 (Lipschitz Policies)**.** *For all* $s,\widetilde{s}\in\mathcal{S}$ *and* $\theta,\widetilde{\theta}\in\Theta$*:*

$$\mathcal{K}\Big(\pi_\theta(\cdot|s),\pi_\theta(\cdot|\widetilde{s})\Big) \leqslant L_{\pi_\theta}\|s-\widetilde{s}\|, \tag{2.36}$$

$$\mathcal{K}\Big(\pi_\theta(\cdot|s),\pi_{\widetilde{\theta}}(\cdot|s)\Big) \leqslant L_\Theta\left\|\theta-\widetilde{\theta}\right\|, \tag{2.37}$$

for some positive real constants $\{L_{\pi_\theta}\}_{\theta\in\Theta}$ *and* L_Θ*. In case* π_θ *is a deterministic policy, (2.36) and (2.37) are replaced with:*

$$\|\pi_\theta(s)-\pi_\theta(\widetilde{s})\| \leqslant L_{\pi_\theta}\|s-\widetilde{s}\|, \tag{2.38}$$

$$\left\|\pi_\theta(s)-\pi_{\widetilde{\theta}}(s)\right\| \leqslant L_\Theta\left\|\theta-\widetilde{\theta}\right\|. \tag{2.39}$$

We use the Euclidean norm to measure distances on $\mathcal{S}$, $\mathcal{A}$ and Θ, but everything works for general metrics. In the following, we will always assume that $L_P(1+L_{\pi_\theta}) < \gamma^{-1}$. These assumptions are enough to guarantee the Lipschitz continuity of the value functions w.r.t. states and actions:

Lemma 2.1 (Rachelson and Lagoudakis, 2010)**.** *Under Assumptions 1 and 2, for all* $s,\widetilde{s}\in\mathcal{S}$*,* $a,\widetilde{a}\in\mathcal{A}$ *and* $\theta\in\Theta$*:*

$$\left|V^\theta(s)-V^\theta(\widetilde{s})\right| \leqslant L_{V^\theta}\|s-\widetilde{s}\|, \tag{2.40}$$

$$\left|Q^\theta(s,a)-Q^\theta(\widetilde{s},\widetilde{a})\right| \leqslant L_{Q^\theta}\, d_{\mathcal{SA}}\left((s,a),(\widetilde{s},\widetilde{a})\right), \tag{2.41}$$

where $L_{Q^\theta} = \frac{L_R}{1-\gamma L_P(1+L_{\pi_\theta})}$ *and* $L_{V^\theta} = L_{Q^\theta}(1+L_{\pi_\theta})$*,*

and also of the future-state distributions w.r.t. policy parameters:

Lemma 2.2 (Pirotta, Restelli, and Bascetta, 2015)**.** *Under Assumptions 1 and 2, for all* $\theta,\widetilde{\theta}\in\Theta$*:*

$$\mathcal{K}\left(\delta^\theta,\delta^{\widetilde{\theta}}\right) \leqslant L_{\delta^\theta}\left\|\theta-\widetilde{\theta}\right\|, \tag{2.42}$$

where $L_{\delta^\theta} = \gamma L_P L_{\pi_\theta}/(1-\gamma L_P(1+L_{\pi_\theta}))$.

2.5 STATE ABSTRACTION

An abstraction is a mapping from one problem representation to a new simpler representation that preserves some properties of interest. State abstractions [Li, Walsh, and Littman, 2006] map MDPs to MDPs with simpler state spaces, typically for computational purposes.

Let $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ be the ground MDP and its abstract version be $\widetilde{M} = \langle \mathcal{X}, \mathcal{A}, \widetilde{P}, \widetilde{R}, \gamma \rangle$. We define the abstraction function as $\Gamma : \mathcal{S} \to \mathcal{X}$, so that $\Gamma(s) \in \mathcal{X}$ is the abstract state corresponding to the ground state s and $\Gamma^{-1}(X) \subseteq \mathcal{S}$ denotes the inverse image of X under Γ. $\widetilde{P}$ is the abstract transition function, with $\widetilde{P}(X'|X, a)$ denoting the probability of reaching X' from X by taking action a, and $\widetilde{R}$ is the reward function with $\widetilde{R}(X, a)$ denoting the expected reward from taking action a in abstract state X.

To guarantee $\widetilde{P}$ and $\widetilde{R}$ are well-defined, we can use a weighting function $w : \mathcal{S} \to [0, 1]$ such that $\int_{\Gamma^{-1}(X)} w(s)\, ds = 1$ for all $X \in \mathcal{X}$. With these weights at hand, we can define the transition and reward functions of the abstract MDP as follows:

$$\widetilde{R}(X, a) = \int_{\Gamma^{-1}(X)} w(s) R(s, a)\, ds,$$
$$\widetilde{P}(X'|X, a) = \int_{\Gamma^{-1}(X)} w(s) \int_{\Gamma^{-1}(X')} P(s'|s, a)\, ds\, ds'.$$

These definitions may be not enough to include in the abstraction definition the notion of state similarity because they are valid regardless of the abstraction function Γ and the weighting function w considered. Another drawback is that the abstract states $X \in \mathcal{X}$ are often imperfect because representing the current environment in terms of abstract states necessarily neglects information. However, finding solution in the abstract state space $\mathcal{X}$ is typically faster than in the ground state space $\mathcal{S}$ because groups of states are treated as a unit. State abstraction allows to apply learning techniques in large, real-world environments. A central issue in the theory of abstraction is to distinguish between relevant and irrelevant information, in order to use the former to outline how the state abstraction is performed and discard the latter.

Several state abstractions schemes have been proposed, all of them aggregating the ground states s that are similar according to some measure into the same abstract state X. For instance, in [Ferns, Panangaden, and Precup, 2012] a measure of similarity $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ between states is defined as follows:

$$d(s, \widetilde{s}) = \max_{a \in \mathcal{A}} \Big(c_R d_R(R(s, a), R(\widetilde{s}, a)) + c_P d_P(P(\cdot|s, a), P(\cdot|\widetilde{s}, a)) \Big), \tag{2.43}$$

where c_R and c_P are two constants such that $c_R + c_P = 1$, d_R is a distance measure (usually the Euclidean distance) and d_P is a distance measure for probability distributions (for instance, the Kantorovich distance defined in (2.31)). We can choose some seed states and cluster all the remaining states according to the distance metric in (2.43).

In this work, we focus on *state aggregation* where $\mathcal{X}$ is a partition of $\mathcal{S}$ and $s \in \Gamma(s)$. Hence, we will often identify $\Gamma^{-1}(X)$ with X (as a set) itself. When it is possible, state abstraction focuses on the preservation of those properties that allow an agent to perform optimal behaviours.

2.5.1 *Irrelevance Abstractions*

In [Li, Walsh, and Littman, 2006], five different types of abstractions are presented, all of which preserve some information that is critical for solving the original MDP. They can be ordered from the method that prescribes the finest abstraction to the method that prescribes the coarsest one. Here we report the five types of abstraction, called *irrelevance abstractions*:

- A *model-irrelevance* abstraction Γ_{model} is such that for any action a and ay avstract state X, $\Gamma_{model}(s_1) = \Gamma_{model}(s_2)$ implies $R(s_1, a) = R(s_2, a)$ and $\int_{X'} P(s'|s_1, a)\, ds' = \int_{X'} P(s'|s_2, a)\, ds'$.
- A Q^π-irrelevance abstraction Γ_{Q^π} is such that for any policy π and any action a, $\Gamma_{Q^\pi}(s_1) = \Gamma_{Q^\pi}(s_2)$ implies $Q^\pi(s_1, a) = Q^\pi(s_2, a)$.
- A Q^*-irrelevance abstraction Γ_{Q^*} is such that for any action a, $\Gamma_{Q^*}(s_1) = \Gamma_{Q^*}(s_2)$ implies $Q^*(s_1, a) = Q^*(s_2, a)$.
- An a^*-irrelevance abstraction Γ_{a^*} is such that every abstract state X has an action a^* that is optimal for all the states $s \in X$, and $\Gamma_{a^*}(s_1) = \Gamma_{a^*}(s_2)$ implies that $Q^*(s_1, a^*) = Q^*(s_2, a^*)$.
- A π^*-irrelevance abstraction Γ_{π^*} is such that every abstract state X has an action a^* that is optimal for all the states $s \in X$, and $\Gamma_{\pi^*}(s_1) = \Gamma_{\pi^*}(s_2)$ implies that $Q^*(s_1, a^*) = \max_a Q^*(s_1, a)$ and $Q^*(s_2, a^*) = \max_a Q^*(s_2, a)$.

However, even the coarsest irrelevance abstraction is impossible to be implemented in most of the MDPs, obviously if we leave out the naive state abstraction in which the set of abstract states $\mathcal{X}$ corresponds to the set of ground states $\mathcal{S}$.

Γ_{model} is a special case of state abstraction, called *bisimulation*, where states with the same transition and reward functions are aggregated. The rule for aggregation is very strict and not applicable in most of the cases. Because of this, in approximated bisimulation the requirements of exact equivalence for the transition and reward functions are replaced with bounds that allow to aggregate in the same abstract state X all the states s for which the difference in the functions is below a certain value. From these bounds, it is possible to build a BMDP and solve it with the method discussed in subsection 2.4.1

[Dean, Givan, and Leach, 2013].

The policies obtained by solving the abstract MDP are a (possibly stochastic) mapping from abstract states to actions. Given an abstract policy ρ, it can be translated in the ground state space $\mathcal{S}$ as:

$$\pi(a|s) = \rho(a|\Gamma(s)) \tag{2.44}$$

The intuitive explanation for equation (2.44) is that the action prescribed by ρ in the abstract state X is copied into all the ground states $s \in X$. However, this translation does not ensure to preserve the performance of ρ in $\widetilde{M}$ for π in M.

Coarser abstraction methods provide a better computational efficiency and generalization w.r.t. finer abstractions but have looser performance loss bounds. Indeed, in [Li, Walsh, and Littman, 2006] it is stated that in Γ_{model}, Γ_{Q^π}, Γ_{Q}^{*} and Γ_{a^*} abstractions, the optimal abstract policy ρ^* is optimal also in the ground MDP, while it can be suboptimal in Γ_{π^*} abstractions. The optimal policy in the ground MDP could require information that is not available in the abstract MDP. For instance, when the agent is in the abstract state X, it does not know the exact ground state $s \in X$ in which it is. The agent in the abstract space $\mathcal{X}$ can only partially observe the ground states s. A non-markovian abstract solution that keeps track of the entire agent's history may address the problem of finding an optimal policy in both the abstract and the ground state spaces, however it is more complicated to learn this kind of solution.

3

STATE OF THE ART

In this chapter we present two topics from RL that are crucial in our work, namely Policy Gradient (PG) (Section 3.1) and Safe Reinforcement Learning (Section 3.2). We also provide the description of some existing algorithm or implementation strategies, some of them representing the state of the art, that we consider in this work in order to make a comparison with our algorithm.

3.1 POLICY GRADIENT METHODS

Most of the algorithms in PS learn the parameterized policy π_θ, with $\theta \in \Theta \subseteq \mathbb{R}^d$, according to the gradient of some scalar performance measure $J(\theta)$ with respect to the policy parameter θ. These methods seek to maximize performance, so they perform gradient ascent updating the parameter θ in the direction of $\widehat{\nabla} J(\theta)$:

$$\theta \leftarrow \theta + \alpha \widehat{\nabla} J(\theta), \tag{3.1}$$

where $\widehat{\nabla} J(\theta) \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ and $\alpha > 0$ is the step size, a scalar that controls the size of each update and can change through time. In PG methods, the policy $\pi(a|s, \theta)$ has to be differentiable with respect to its parameters $\theta \in \mathbb{R}^d \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

With appropriate policy parametrization the action probabilities change smoothly as a function of the learned parameter, whereas for greedy policies the action probabilities may change enormously for a small change in the estimated action values. Stronger convergence guarantees are available for policy-gradient methods and approximate gradient ascent can be performed as in equation (3.1). According to [Peters, 2010], if the gradient estimate is unbiased[1] and learning rates fulfill the following conditions:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{3.2}$$

the learning process is guaranteed to converge to at least a local optimum.

1 In [Williams, 1992a] it is proved that an unbiased estimate of the gradient can be obtained from samples without the assistance of approximate functions.

Some methods also learn approximations to value-functions and are called *actor-critic methods*, where "actor" is a reference to the learned policy, and "critic" refers to the learned value function. In these methods, the function approximation for value function introduces bias but reduces variance and accelerates learning.

3.1.1 *Policy Gradient Theorem*

It may seem challenging to change the policy parameter in a way that ensures improvement. The problem is that the performance $J(\theta)$, i. e. the measure to maximize, depends on both the action selections and the distribution of states in which those selctions are made, and both of these are affected by the policy parameter θ. We have to estimate the performance gradient with respect to θ but the gradient is affected by the unknown effect of policy changes. Fortunately, there is a theoretical answer to this challenge in the form of the *policy gradient theorem*, which provides an analytic expression for $\nabla_\theta J(\theta)$ that does not involve the derivative of the state distribution $\delta^\theta(s)$.

The theorem is from [Sutton et al., 1999] and we enunciate it in the case of continuous state space $\mathcal{S}$ and action space $\mathcal{A}$:

Theorem 3.1 (Policy Gradient Theorem)**.** Given a stochastic policy π_θ differentiable w.r.t. its parameter, the gradient of performance measure $J(\theta)$ with respect to the policy parameter θ can be written as:

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \delta^\theta(s) \int_{\mathcal{A}} \nabla_\theta \pi(a|s) Q^\theta(s,a)\, da\, ds. \tag{3.3}$$

Starting from (3.3), the performance gradient $\nabla_\theta J(\theta)$ can be estimated from samples in different ways. In subsection 3.1.2 we present some PG algorithms. Generally, a batch of N trajectories is sampled and N single estimates of the gradient are averaged to obtain the final estimate. This estimate is the $\widehat{\nabla} J(\theta)$ used in (3.1) to perform a gradient ascent iteration that updates the policy parameter θ.

3.1.2 *Policy Gradient Algorithms*

In this section we present some PG algorithms that have some interesting properties for our work. We compare these existing algorithms and their properties with the algorithm proposed in our work. The first algorithm we present is REINFORCE, a method directly derived from the Policy Gradient Theorem that shows how the gradient of performance can be easily estimated from samples. Then, we present two methods that learn deterministic policies. The difference between these two methods is the way in which exploration is performed: in Policy Gradients with Parameter-Based Exploration (PGPE) exploration is performed in the policy space while in Deterministic Policy

Gradient (DPG) exploration is performed in action space by means of a behavioural policy (i. e. a policy that collects samples, different from the policy that is being learnt).

REINFORCE: In REINFORCE the gradient of performance $\nabla_\theta J(\theta)$ can be estimated from a single trajectory $\langle s_0, a_0, r_1, s_1, a_1, ..., s_{T-1}, a_{T-1}, r_T \rangle$ without performing any sort of perturbation on parameters θ, according to [Williams, 1992b]. Starting from (3.3), we can write the gradient as an expected value depending on the state distribution δ^θ and the parameterized policy π_θ. Then, the samples collected following π_θ can be used for the estimation of $\nabla_\theta J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_s \delta^\theta(s) \int_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} Q^\theta(s,a)\, da\, ds \\ &= \mathop{\mathbb{E}}_{s\sim\delta^\theta, a\sim\pi_\theta} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} Q^\theta(s,a) \right] \\ &= \mathop{\mathbb{E}}_{s\sim\delta^\theta, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\theta(s,a) \right] \qquad (3.4)\end{aligned}$$

where (3.4) is obtained by considering $\nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$ and $Q^\theta(s,a)$, usually unknown, can be replaced by an action-value function approximation.

In episodic tasks, the return G_t can be computed from every time step t once the episode is terminated. Then, $Q^\theta(s,a)$ can be approximated with G_t for every state-action pair (s_t, a_t) sampled in the trajectory. This approximation is unbiased, because $\mathbb{E}_{\pi_\theta}[G_t|s_t, a_t] = Q^\theta(s_t, a_t)$. Given a batch of N trajectories $\langle s_0, a_0, r_1, s_1, a_1, ..., s_{T-1}, a_{T-1}, r_T \rangle$ from an episodic task, the gradient of performance can be estimated as:

$$\widehat{\nabla}_\theta J(\theta) = \left\langle \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right\rangle_N, \qquad (3.5)$$

where $\langle \cdot \rangle_N$ denotes the average of the inner expression over the N trajectories.

REINFORCE has good theoretical convergence properties: by construction, an improvement in expected performance and convergence to a local optimum under standard stochastic approximation conditions are ensured [Sutton and Barto, 2018]. However, it presents an high variance that slows the convergence process. In order to mitigate this problem, we consider *baseline functions* $b : S \to \mathbb{R}$, i. e. functions that don't depend on the action and can reduce variance in the estimate without introducing bias. Indeed, we observe that if $b(s)$ does not depend on a we can modify (3.3) in the following way:

$$\nabla_\theta J(\theta) = \int_s \delta^\theta(s) \int_a \nabla_\theta \pi_\theta(a|s) \Big(Q^\theta(s,a) - b(s) \Big)\, da\, ds. \qquad (3.6)$$

The new term appearing in (3.6) doesn't affect the value of gradient:

$$\int_a b(s)\nabla_\theta \pi_\theta(a|s)\,da = b(s)\nabla_\theta \int_a \pi_\theta(a|s)\,da = b(s)\nabla_\theta 1 = 0. \quad (3.7)$$

An intuitive baseline $b(s)$ that can be used is the value function $V^\theta(s)$. In some tasks, $Q^\theta(s,a)$ can assume high values and the use of $V^\theta(s)$ as baseline provides an effect of normalization. The difference in expression (3.6) becomes the advantage function $A^\theta(s,a)$ when V is used as a baseline:

$$\nabla_\theta J(\theta) = \int_s \delta^\theta(s) \int_a \nabla_\theta \pi_\theta(a|s)\Big(Q^\theta(s,a) - V^\theta(s)\Big)\,da\,ds. \quad (3.8)$$

In [Peters and Schaal, 2008] a different formulation for the estimation of gradient is provided. It is used in the *GPOMDP* algorithm and reduces variance in the estimation of gradient because it prevents to sum, for each time-step, the rewards obtained from the current time-step until the end of the episode:

$$\widehat{\nabla}_\theta^{\text{GPOMDP}} J(\theta) = \Big\langle \sum_{l=0}^{T-1} \Big(\sum_{k=0}^{l} \nabla_\theta \log \pi_\theta(a_k|s_k) \Big) \Big(\gamma^l r_{l+1} - b_l \Big) \Big\rangle_N. \quad (3.9)$$

Apart from the baseline b_l, this expression is equivalent to (3.5). In this work, we consider the GPOMDP estimation of gradient with the variance-minimizing baseline provided by [Peters and Schaal, 2008]:

$$b_l = \frac{\Big\langle \Big(\sum_{k=0}^{l} \nabla_\theta \log \pi_\theta(a_k|s_k) \Big)^2 \gamma^l r_{l+1} \Big\rangle_N}{\Big\langle \Big(\sum_{k=0}^{l} \nabla_\theta \log \pi_\theta(a_k|s_k) \Big)^2 \Big\rangle_N}, \quad (3.10)$$

where the square operation is performed in an element-wise manner on the vector. This is one of the most common policy gradient algorithms and we use it in this work to compare our algorithm with a standard algorithm in policy search.

POLICY GRADIENT WITH PARAMETER-BASED EXPLORATION (PGPE): PGPE [Sehnke et al., 2008] is a method that estimates a gradient by directly sampling in parameter space. In PGPE the policy is defined by a distribution over the parameters of deterministic controllers that we indicate with the function $\mu_\theta : \mathcal{S} \to \mathcal{A}$. At the beginning of each step, the parameter θ of the controller is sampled and then, the deterministic policy μ_θ generated from θ is followed for all the episode length. Even if samples are collected with a detrministic policy, the choice of the policy is stochastic and then unexpected behaviour could arise in the task.

The variance of the estimates obtained with PGPE is lower than

the variance of the estimates obtained with REINFORCE [Zhao et al., 2013]. This is due to the fact that, in REINFORCE, a repetitive sampling from a stochastic policy injects noise in the gradient estimate at every time-step. Furthemore, the variance increases linearly with the length of the history since each state depends on the entire sequence of previous samples.

As we said, in PGPE the stochasticity that in REINFORCE was given by a stochastic policy π_θ is replaced by a probability distribution over the parameters θ themselves. In turn, this probability distribution is parameterized with parameter ρ, independent from θ. Given a parameter space Θ for (deterministic) controllers $\mu_\theta, \theta \in \Theta$, the policy considered to perform exploration in the task is:

$$\pi_\rho(a|s) = \int_\Theta p_\rho(\theta)\delta_{\mu_\theta(s)}\, d\theta, \tag{3.11}$$

where $\delta_{\mu_\theta(s)}$ is the Dirac delta function corresponding to the deterministic controller μ_θ depending on parameter θ sampled from a distribution with probability $p_\rho(\theta)$.

Given a trajectory $h \in \mathbb{H}$, where $\mathbb{H}$ is the set of possible trajectories, a probability $p_\rho(h, \theta)$ of sampling parameter θ and trajectory h and defined $G(h)$ the return of trajectory h, we can define a suitable performance measure $J(\rho)$ as:

$$J(\rho) = \int_\Theta \int_{\mathbb{H}} p_\rho(h, \theta) G(h)\, dh\, d\theta. \tag{3.12}$$

From 3.12, we can write the expected value of the gradient of J w.r.t. the distribution parameter ρ as:

$$\widehat{\nabla}_\rho J(\rho) \approx \nabla_\rho \log p_\rho(\theta) G(h), \tag{3.13}$$

where θ is sampled at the beginning of the episode and h is resulting from the deterministic controller μ_θ. This is a black-box optimization (BBO) approach because it uses a constant policy perturbation during policy execution and it stores only scalar return in the trajectory [Stulp and Sigaud, 2012]. Differently from REINFORCE, for instance, PGPE does not use information fron single time steps and, then, it does not exploit the temporal structure of RL problems.

Usually, we can consider the parameter ρ learnt by the algorithm as $\rho = \Big(\{\mu_i\}, \{\sigma_i\}\Big)$, where μ_i and σ_i are the parameters that determine an indipendent normal distribution.

DETERMINISTIC POLICY GRADIENT (DPG): DPG [Silver et al., 2014] is a class of algorithms that learn a parametric deterministic policy μ_θ defined as $\mu_\theta : S \to A$. The approach in DPG is similar to

the one used in PG methods that learn stochastic policies π_θ, where the gradient of performance is estimated according to the Policy Gradient Theorem (3.1.1). However, for deterministic policies the gradient of performance is slightly different than the one computed by the theorem (3.3).

Indeed, by considering a deterministic policy μ_θ with parameter vector $\theta \in \mathbb{R}^n$ and performance objective $J(\mu_\theta) = \mathbb{E}_{s\sim\delta^\mu}[R(s, \mu_\theta(s))]$, the expression for *Deterministic Policy Gradient* derived by the theorem is the following:

Theorem 3.2 (Deterministic Policy Gradient Theorem)**.** Suppose that $P(s'|s,a), \nabla_a P(s'|s,a), \mu_\theta(s), \nabla_\theta \mu_\theta(s), R(s,a), \nabla_a R(s,a), p_0(s)$ are continuous in all parameters θ and variables s, a, s'. Then,

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\delta^\mu}\Big[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\Big]. \tag{3.14}$$

In order to ensure exploration while learning a deterministic policy μ_θ, an *off-policy* algorithm is introduced. Both in our work and in DPG algorithms a deteministic policy is learnt. However, during the learning phase DPG algorithms collect samples from a stochastic behavioral policy, differently from what our goal aims: to avoid the execution of random actions.

More precisely, relying on Theorem 3.2 an off-policy actor-critic algorithm can be derived. It learns a deterministic target policy $\mu_\theta(s)$ from trajectories generated by an arbitrary stochastic behavioral policy $\pi(a|s)$. A critic estimates the action-value function $Q^w(s,a) \approx Q_\mu(s,a)$, where w is the parameter of the function approximator $Q^w(s,a)$. In [Silver et al., 2014] the conditions required for a function approximator to be compatible (i. e. avoid to introduce any bias in gradient update) are specified.

Deep Deterministic Policy Gradient (DDPG) is an algorithm of the DPG class that follows an off-policy actor-critic scheme. It uses four neural networks: an actor μ_θ, a critic Q_ϕ, a target actor $\mu_{\theta_{targ}}$ and a target critic $Q_{\phi_{targ}}$, where $\theta, \phi, \theta_{targ}$ and ϕ_{targ} are the parameters of the neural networks. The behavioral policy is obtained from μ_θ by adding a Gaussian noise to the action prescribed. From the collected samples, a batch B of samples $i = (s, a, r, s')$ is generated in order to perform a parameter update.

The critic parameter ϕ is updated in a way such that the following *loss measure* L is minimized:

$$L = \frac{1}{|B|} \sum_{i\in B} \Big(y_i - Q_\phi(s_i, a_i)\Big)^2, \tag{3.15}$$

where $y_i = r_i + \gamma Q_{\phi_{targ}}\left(s', \mu_{\theta_{targ}}(s')\right)$. The actor parameter μ is updated by one step of gradient ascent using an expression derived from Theorem 3.2 and applying the inverse of *chain rule*:

$$\frac{1}{|B|}\sum_{i\in B}\nabla_\theta Q_\phi\left(s, \mu_\theta(s)\right) \tag{3.16}$$

Finally, the target policy are updated as follows, in order to ensure stability in learning according to hyperparameter ρ:

$$\phi_{targ} = \rho\phi_{targ} + (1-\rho)\phi \tag{3.17}$$

$$\theta_{targ} = \rho\theta_{targ} + (1-\rho)\theta \tag{3.18}$$

3.2 SAFE REINFORCEMENT LEARNING

In RL several definitions of safety have been proposed and also methods that ensure a certain degree of safety according to some measures of risk. These definitions refer to different facets of safety, hence they are not conflicting to each other. In this section, firstly we provide an overview of the different methods that face the problem of safe learning, using the same classification proposed by García and Fernández, then we focus on the more specific issue related to *safe exploration*.

According to [García and Fernández, 2015], there exist two main trends for Safe RL. The first one is based on the modification of the optimality criterion to introduce the concept of risk. The second is based on the modification of the exploration process, so as to avoid the exploration of actions that may lead the system to undesirable or catastrophic situations. Regarding the first trend, there are several alternatives to quantify risk:

- *Worst Case Criterion*: a policy is considered to be optimal if it has the maximum return w.r.t. the worst-case scenario, namely the worst possible trajectory in terms of return:

$$\max_{\pi\in\Pi}\min_{h\in H^\pi}\mathbb{E}_{\pi,h}[G_0], \tag{3.19}$$

  where H^π is a set of trajectories that occurs under the policy π and the quantity to maximize-minimize is an expected value w.r.t. the policy π and the trajectory h [Heger, 1994]. This criterion is used to mitigate the effects of variability induced by a given policy. It is also possible to use this kind of criterion when the transition function P is uncertain:

$$\max_{\pi\in\Pi}\min_{P}\mathbb{E}_{\pi,P}[G_0], \tag{3.20}$$

- *Risk-Sensitive Criterion*: the optimization criterion balances return and risk by means of a scalar parameter that is included in the objective function and allows the sensitivity to the risk to be controlled. Risk can be defined, for instance, as the variance of return or as the probability of entering into an error state [Geibel and Wysotzki, 2005].

- *Constrained Criterion*: the method maximizes the expectation of return subject to one or more constraints. It is defined as:

$$\max_{\pi \in \Pi} \mathbb{E}_{\pi}[G] \text{ subject to } c_i \in C, \tag{3.21}$$

  where G is the return and C is the set of constraints c_i, with $c_i = f_i \leqslant t_i$. f_i is a function related with the return and t_i is its threshold [Moldovan and Abbeel, 2012]. The set of constraints C limits the space of allowable policies and makes this optimization criterion suitable for risky domains. The constraints, for instance, can ensure that the expectation of return exceeds a minimum threshold or that the variance of the return does not exceed a maximum threshold.

Regarding the modification of the exploration process, it can be modified through the incorporation of external knowledge in three different ways:

- *Providing Initial Knowledge*: examples gathered from a teacher or previous information on the task can be used to provide an initial knowledge for the learning algorithm. From the ealier steps of the algorithm, the most relevant regions of the state and action spaces are visited. It considerably reduces random exploration providing guidance in the form of reasonable policies [Driessens and Džeroski, 2004].

- *Deriving a policy from a finite set of demonstrations*: a set of examples provided by a teacher, that replace examples provided by exploration, can be used to learn a model from which to derive a policy in an off-line and, hence, safe manner. An example of this approach, related to autonomous helicopter flight, is provided in [Abbeel, Coates, and Ng, 2010].

- *Providing Teach Advice*: a human or a simple controller assists the exploration during the learning process and provides advice. In some approaches, the teacher can provide advice only when the agent explicitly asks for, as in [Clouse, 1997], in others whenever it feels it is necessary.

Alternatively, the exploration can be managed through the use of a risk measure that favors the execution of low-risk actions, this approach is called *Risk-directed Exploration*. An example of it is

provided in [Gehring and Precup, 2013], where the notion of *state controllability* is defined. This notion tells the agent in which states the effects of actions are easier to predict.

Accidents in learning systems may emerge from a poor design of real-world systems. According to [Amodei et al., 2016], the system could behave undesirably if the designers wrongly carry out one of the following:

- assign a wrong objective function: the function may not consider the negative side effects of some actions. Moreover, the function may allow the agent to hack the reward. Indeed, the agent may find a way to obtain an high reward with an unintended behavior;
- define an objective function that is too expensive to evaluate frequently;
- enable undesirable behavior during the learning phase: the agent may perform dangerous exploratory moves (subsection 3.2.1) or it may have difficulties to recognize environments that are different from the training one.

Another approach to safe RL is presented in [Thomas et al., 2019]. Here it is proposed a framework that simplifies the problem of specifying and regulating undesirable behavior. The framework allows to easily constrain the behavior of the algorithm, without requiring extensive domain knowledge or additional data analysis, by setting the maximum admissible probability of undesirable behavior.

3.2.1 *Safe Exploration*

In our work, it is fundamental to ensure that the agent learns in complete safety (i. e. the agent doesn't perform undesired actions throughout the learning phase). The second class of safe methods is suitable for this. However, it requires external knowledge, in the form of experience samples or teacher advice, that is not always available. Instead, the methods of the first class can learn an optimal policy (according to a definition of return that includes risk) but they trade-off short term loss in performance for a long term gain. This is not accettable in the case of safety-critical applications, regardless of whether the return contains a measure of risk. Moreover, when the risk is encoded in the reward function, any sort of performance oscillation points out a dangerous situation that can cause failure or harm the environment. In order to avoid this source of risk, a monotonic improvement in performance can be required while learning the policy. Here we describe two state of the art approaches to safe exploration, the first one for finite MDPs and the second one for policy gradient methods. Both of

the methods rely on a statistical confidence ϵ in order to define safety constraints, then the complete avoidance of unwanted actions is only ensured with high probability.

SAFE EXPLORATION IN FINITE MDPS: [Turchetta, Berkenkamp, and Krause, 2016] address the problem of safely exploring finite MDPs, where safety is defined in terms of a constraint that satisfies regularity conditions. The algorithm, called SafeMDP, cautiously explores safe states and actions, obtains noisy observations and gains knowledge on the safety of unvisited state-action pairs. Starting from a set of states and actions that are known to be safe, the regularity assumptions are exploited in order to evaluate only state-action pairs known to fulfill the safety constraint. The reward (encoding a measure of safety) is unknown and drawn from a Gaussian distribution. At each iteration of the algorithm a posterior distribution is computed from the sampled rewards, affected by an additive noise drawn from a zero-mean Gaussian distribution. The reward function $R(s)$ is Lipschitz continuous, with Lipschitz constant L_R. Since only noisy measurements are observed, $R(s)$ is known up to some statistical confidence ϵ. By considering the Lipschitz continuity of $R(s)$ and the confidence ϵ and starting from some safe set S_s, the resulting set of safe states is:

$$R_\epsilon^{safe}(S_s) = S_s \cup \{s \in S | \exists s' \in S_s : R(s') - \epsilon - L_R d(s, s') \geqslant h\}, \quad (3.22)$$

where h represents a safety threshold. The obtained set is then restricted by considering only the safe states that are reachable and from which we can move to other safe states. By iteratively applying the operator R_ϵ^{safe} in (3.22) to the restricted set of safe states S_s and updating the Gaussian distribution of $R(s)$ from samples, we obtain the largest set of states that can be safely reached by the exploring agent.

SAFE POLICY GRADIENTS: Safe Policy Gradient (SPG) is an algorithm from [Papini, Pirotta, and Restelli, 2019] in which the learning agent is constrained to never worsen its performance during learning. According to the classification of safe RL methods, the algorithm is included in the *constrained criterion* class as oscillating performances may violate the constraint, called *Monotonic Improvement*. It can be considered a safety constraint only when the risk is perfectly encoded in the reward. This work involves actor-only PG from a stochastic optimization perspective and *smoothing policies*, i. e. twice-differentiable parametric policies π_θ, with $\theta \in \Theta$, for which properties[2] that induce the smoothness of performance are valid. In [Papini, Pirotta, and

2 The parameter space Θ is convex. For every state s and in expectation over actions $a \sim \pi_\theta(\cdot|s)$, the Euclidean norm $\|\nabla \log \pi_\theta(a|s)\|$, the squared Euclidean norm $\|\nabla \log \pi_\theta(a|s)\|^2$ and the spectral norm $\left\|\nabla \nabla^\intercal \log \pi_\theta(a|s)\right\|$ are upper-bounded by non-negative constants.

Restelli, 2019] it is shown that Gaussian and softmax policies are smoothing. From upper bounds on the variance of policy gradient estimators, a lower bound on the performance improvement (with a certain confidence ϵ) provided by gradient-based updates is calculated and expressed as a function of some meta-parameters. These meta-parameters are the step size α of the parameter updates and the batch size N of the gradient estimators. Then, the adaptive meta-parameters that guarantee monotonic improvement with high probability are identified.

PG methods may suffer from the explosion of gradients when the current policy is close to be deterministic, leading to unstable training process. Target Distribution Learning (TDL) [Zhang, Li, and Li, 2019] addresses this problem, alternating between proposing a target distribution and training the policy to approach the target distribution. The target policy is the solution of a constrained optimization problem where the constraints involve the difference between updated policies, so that the algorithm leads to more stable improvements.

4

DETERMINISTIC POLICY OPTIMIZATION

This is the most important chapter in our work, where we present the algorithm we designed, called Deterministic Policy Optimization (DPO). The chapter is divided in four sections. In Section 4.1 we provide an explanation of the approach that we followed, motivated by the aim of this work (presented in Chapter 1 and recalled in Section 4.1), in Section 4.2 we describe the algorithm according to high-level pseudocode of it. Details on the implementation of the different parts of the algorithm are provided in Section 4.3, while in Section 4.4 we explain how to address the concrete issues that arise in the algorithm, related to the necessity of function estimations.

4.1 PASSIVE EXPLORATION VIA δ-MDPS

As mentioned in Chapter 1, the goal of this work is to ensure that the agent is able to learn a deterministic policy avoiding the execution of dangerous actions throughout the learning phase (in this way the agent does not harm itself or the environment in which it evolves). We address this requirement on exploration by forcing the agent to gather the samples needed for improving the parametric policy *without performing any random action*. A way to satisfy this strict constraint is to learn a deterministic policy and use it to collect samples so that no random actions can be performed at all. By doing this, the exact action that will be performed by the agent is known in every moment. This choice, unfortunately, prevents any form of active exploration and the possibility to update the parametric policy with existing algorithms. The fundamental reason is that, by observing a single action for each (continuous) state and without any probability measure over actions, the agent has no way of preferring one action over another for that particular state, which makes policy optimization impossible.

However, if the environment is sufficiently regular, we can exploit a form of *passive* exploration. The key idea is to aggregate states in such a way as to observe a variety of actions within these larger, abstract states. This variety is obtained by considering, for every abstract state, the set of actions chosen by our deterministic policy in the different states belonging to the same aggregate. In a sense, aggregation allows transferring the diversity of the states visited by the agent (both over different time-steps of the same episode and over independent episodes) into actions. Variety over states is encoded by the future-state distribution δ^{θ}, depending on the

deterministic policy π^θ, and is fueled by stochastic transitions and random restarts that allow to ideally cover the entire state space $\mathcal{S}$. The price to pay in order to consider state aggregation is basically a discretization error, that could be, in general, unbounded. However, in Lipschitz MDPs we can keep this error under control, as we will show in Chapter 5.

In a Lipschitz MDP (see subsection 2.4.2 for details on it), if we know the effect of performing an action a in a state s, we can exploit this knowledge to obtain information about the effect of performing the same action a on a different state $\widetilde{s}$. As an example we can consider a deterministic environment[1] in which we collect a sample (s, a, s'), where the next-state s' is obtained from a state-action pair (s, a). In deterministic environments, the next state s' is computed as $s' = f(s, a)$ according to the function $f\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ that conducts the state transition from s to s'. For a deterministic environment, the assumption of Lipschitz regularity on the transition function P defined in (2.34) can be rewritten as:

$$\|f(s, a) - f(\widetilde{s}, \widetilde{a})\| \leqslant L_P \, d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right), \tag{4.1}$$

for some positive real constant L_P. From this inequality, the next-state $\widetilde{s'}$ resulting from an initial state-action pair $(\widetilde{s}, \widetilde{a})$ can be estimated even if the agent never executes the action $\widetilde{a}$ in the state $\widetilde{s}$. The estimation of $\widetilde{s'}$ comes in the form of an interval, we show how it can be derived in the case of one-dimensional states and action spaces:

$$\begin{array}{rcccl}
-L_P \, d_{\mathcal{SA}}(\cdot) & \leqslant & f(s, a) - f(\widetilde{s}, \widetilde{a}) & \leqslant & L_P \, d_{\mathcal{SA}}(\cdot) \\
f(s, a) - L_P \, d_{\mathcal{SA}}(\cdot) & \leqslant & f(\widetilde{s}, \widetilde{a}) & \leqslant & f(s, a) + L_P \, d_{\mathcal{SA}}(\cdot) \\
s - L_P \, d_{\mathcal{SA}}(\cdot) & \leqslant & \widetilde{s'} & \leqslant & s + L_P \, d_{\mathcal{SA}}(\cdot),
\end{array} \tag{4.2}$$

where $d_{\mathcal{SA}}(\cdot) = d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right)$ is the taxicab norm on $\mathcal{S} \times \mathcal{A}$ considered in (2.34). Details on how to derive the intervals in the case of multi-dimensional states are provided in Section 4.4. The tightness of the estimated intervals, and then their precision, increases according to the similarity between the state-action pairs (s, a) and $(\widetilde{s}, \widetilde{a})$ used to build the intervals. It appears to be a good idea, starting from an observed state-action pair (s, a), to estimate the effect of executing the action a in all the states $\widetilde{s}$ visited by the agent that are considered similar to s. In this way, the agent only performs the actions suggested by the deterministic policy and estimates intervals that can be deemed sufficiently accurate. For a more detailed explanation of how these simple considerations are used to estimate the unknown information required by the algorithm, refer to Section 4.4.

1 An environment in which the transition function P is a Dirac delta function.

4.1.1 δ-*MDPs*

We are going to exploit the ideas derived above by building an abstract MDP in which the state space is composed by aggregates of states. We divide the original state space $\mathcal{S}$ into a partition $\mathcal{X}$. To take advantage of the Lipschitz continuity of our environment, we need to aggregate states taking their mutual distance into account. For this reason, we only consider *tessellations* of the state space, such as grids. We denote as $D(X) = \sup_{s,\widetilde{s}\in X} \|s - \widetilde{s}\|$ the diameter of an abstract state $X \in \mathcal{X}$. We want that the transition and the reward functions of our abstract MDP are able to model the overall effect of actions on the abstract states, so that the optimization in the abstract MDP corresponds to the resolution of the original policy optimization problem. Doing this exactly would require additional assumptions, like bisimulation (details in Section 2.5), which we deem too restrictive. Indeed, bisimulation is an *equivalence relation* that allows to obtain a partition of the state space composed by *equivalence classes* where the optimal values and the optimal policies of the original MDP are preserved [Givan, Dean, and Greig, 2003], however this assumption is too stringent and difficult to obtain. [Ferns, Panangaden, and Precup, 2012] shows with an example that in a simple MDP with only fours states, the constraints on the transition function P needed to ensure bisimulation are already strict. Hence, we design a solution that approximates the resolution of the original problem. To achieve this purpose, we consider the future-state distribution δ^θ in order to define the weighting function for the state abstraction (details in Section 2.5), obtaining:

$$\widetilde{R}_\theta(X, a) = \int_X \frac{\delta^\theta(s)}{Z_\theta(X)} R(s, a)\, ds, \tag{4.3}$$

$$\widetilde{P}_\theta(X'|X, a) = \int_X \frac{\delta^\theta(s)}{Z_\theta(X)} \int_{X'} P(s'|s, a)\, ds\, ds', \tag{4.4}$$

with weighting function $w(s) = \delta^\theta(s)/Z_\theta(X)$, where $Z_\theta(X) = \int_X \delta^\theta(s)\, ds$ is a normalization factor necessary to ensure that the weights sum to one. This completes the definition of our abstract MDP, called δ-MDP in the following to stress the fundamental role of the future-state distribution δ^θ in its definition:

Definition 4.1. Given an MDP $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, a policy π_θ, and a partition $\mathcal{X}$ of $\mathcal{S}$, the corresponding δ^θ-MDP is $\langle \mathcal{X}, \mathcal{A}, \widetilde{P}_\theta, \widetilde{R}_\theta, \gamma \rangle$, where $\widetilde{R}_\theta$ is defined as in Equation (4.3) and $\widetilde{P}_\theta$ as in Equation (4.4) for all $X, X' \in \mathcal{X}$ and $a \in \mathcal{A}$.

An initial-state distribution for the δ-MDP can be defined as $\widetilde{p}_0(X) = \int_X p_0(s)\, ds \quad \forall X \in \mathcal{X}$, where p_0 is the initial-state distribution for the original MDP. A (stationary[2], Markovian) abstract policy $\rho : \mathcal{X} \to \Delta(\mathcal{A})$ is a (possibly stochastic) mapping from abstract states to actions. In

2 The policy does not changes over time.

order not to confuse them with concrete policies π, we denote them with the letter ρ. Fixed the abstract MDP, we can define value functions V^ρ and Q^ρ, future-state distribution δ^ρ (over $\mathcal{X}$) and optimal policy ρ^*, as in any MDP. Note that there is no direct correspondence, in general, between concrete and abstract policies, since the behavior of a concrete policy may appear non-Markovian in the abstract MDP [Li, Walsh, and Littman, 2006]. However, we introduce a simple "concretization" operator $\mathcal{C} : \rho \mapsto \pi$ from abstract policies to concrete policies, defined as $(\mathcal{C}\rho)(s) = \rho(\Gamma(s))$ for any $\rho : \mathcal{X} \to \mathcal{A}$ and $s \in \mathcal{S}$. The resulting concrete policy $\pi = \mathcal{C}\rho$ performs the same action $\rho(X)$ for all states $s \in X$, hence it is a piece-wise constant function.

4.2 DETERMINISTIC POLICY OPTIMIZATION

In this section, we show how to use the δ-MDP defined in Section 4.1 to approximately solve the policy optimization problem, under Lipschitz conditions but without access to random actions. The proposed methodology is as follows: given the original MDP and a deterministic parametric policy π_θ, we build the corresponding δ^θ-MDP using only the data collected with π_θ itself. This requires estimating the abstract reward and transition functions. To do so, we can actively exploit the Lipschitz conditions, as discussed in the next sections (Section 4.3, Section 4.4).

The solution of the δ^θ-MDP is a deterministic optimal abstract policy ρ^*, which is guaranteed to exist and maximizes $V^\rho(X)$ for all $X \in \mathcal{X}$. This abstract policy can be emulated in the original MDP by $\mathcal{C}\rho^*$, the piecewise-constant policy, even if in general $\mathcal{C}\rho^*$ is not optimal for the original MDP and may not belong to the original parametric policy space. Hence, we need to project it back as $\pi_{\theta'} = \mathcal{P}_{\Pi_\Theta}(\mathcal{C}\rho^*)$, where $\mathcal{P}$ is a projection operator. The new parameter vector θ' identifies the novel policy that we are going to run in the environment. The procedure can then be repeated. We call this general method Deterministic Policy Optimization (DPO for short), and we outline (at high level) the different phases of it in Algorithm 1. Details on the implementation of the single phases are described in Section 4.3. We provide a theoretical justification of this approach in Chapter 5.

4.3 ALGORITHMIC DETAILS

In this section, we provide further details on our implementation of the different phases of DPO, outlined in Algorithm 1: state aggregation (or state partition), estimation of $\widetilde{R}_\theta$ and $\widetilde{P}_\theta$ for the creation of the δ-MDP, solution of it and projection of $\mathcal{C}\rho^*$ into Π_Θ.

Algorithm 1 DPO

Input: policy class Π_Θ, initial policy parameter θ, batch size N
Partition the state space into $\mathcal{X}$
for $t = 0, 1, \ldots$ **do**
  Collect N samples with π_θ
  Estimate $\widetilde{R}_\theta$ and $\widetilde{P}_\theta$ over $\mathcal{X}$
  Solve the δ^θ-MDP to find optimal abstract policy ρ^*
  Project $\mathcal{C}\rho^*$ back into Π_Θ to find θ'
  $\theta \leftarrow \theta'$
end for

4.3.1 *State aggregation*

We discretize the state space $\mathcal{S} \subseteq \mathbb{R}^N$ into a regular grid $\mathcal{X}$, in which each hyperrectangular cell is an abstract state. We assume that the domain of each of the N state variables is a continuous interval. If a variable is unbounded or the bounds are unknown, they are set equal to the minimum and maximum value on the corresponding dimension, observed among the collected samples. For each dimension $i \in N$, it is provided in input to the algorithm a scalar k that indicates the number of subsets in which the $i-th$ dimension of the state space $\mathcal{S}$ has to be divided. We consider each dimension i separately from the others and we divide the one-dimensional space in a partition of k convex subsets with constant diameter. For instance, we consider a two-dimensional space in the domain $[2;4] \times [-4;4] \subset \mathbb{R}^2$ and a vector $\mathbf{k} = [2,4]$ representing the number of subsets in which each dimension has to be divided. The first dimension is divided in two subsets having domains $[2;3)$ and $[3;4]$, the second dimension is divided in four subsets having domains $[-4;-2)$, $[-2;0)$, $[0;2)$ and $[2;4]$. Hence, the original state space $\mathcal{S}$ is divided in 8 subsets $X \in \mathcal{X}$, the domain of each subset X is the combination of one intervals from the first dimension and one from the second dimensions. By partitioning in this way, the measure of the subsets'edges can vary a lot among the different state-dimensions. If the original state space $\mathcal{S}$ contains an absorbing state, we add an absorbing abstract state to $\mathcal{X}$. This can be useful for modeling indefinite-horizon tasks.

The state space partition is not required to be the same across the different iterations of the algorithm, since a new δ-MDP is built at each iteration. For instance, if some state-dimensions are unbounded and we establish the minimum and maximum possible values for them based on the collected samples at each iteration, the size of the subsets in the partition changes between iterations, according to the minimum and maximum values considered. An adaptive discretization of the state space, i.e. a discretization that changes between iterations based on the evolving situation, can

provide benefits such as more efficient use of the collected samples. However, for computational reasons we can avoid to compute a new partition of the state space at the beginning of each iteration and keep a fixed one across the multiple iterations. In the latter case, if we collect a samples whose starting state is outside from the grid, the sample can be assigned to the closest cell in the partition. If naively implemented, this aggregation strategy is clearly subject to the *curse of dimensionality*. However, only the abstract states that are actually visited by π_θ are considered in the next steps of the algorithm, so the size of the δ-MDP is polynomial in the number of collected samples[3]. More details on how the abstract state space $\mathcal{X}$ is built in the experiments and how the chosen aggregation strategy affects the performances of the algorithm are provided in Chapter 6.

4.3.2 *Abstract MDP estimation*

A fundamental aspect in the creation of the δ-MDP is the estimation of the γ-discounted future-state distribution $\delta^\theta(s)$, which we avoid to explicitly perform because it would require too many samples to obtain an accurate estimation. To construct the δ^θ-MDP, given the current policy π_θ, we need to estimate the abstract reward function $\widetilde{R}_\theta$ and the abstract transition function $\widetilde{P}_\theta$. As we previously said, the weighting function $w(s)$ that weighs the abstract functions $\widetilde{R}_\theta(X, a)$ and $\widetilde{P}_\theta(X'|X, a)$ of the δ-MDP is defined according to $\delta^\theta(s)$. The exact computation of the abstract functions would require knowledge of the future state distribution δ^θ, which is out of reach. However, we can write the abstract functions as expected values:

$$\widetilde{R}_\theta(X, a) = \mathop{\mathbb{E}}_{s\sim p(s|X)} R(s, a) \tag{4.5}$$

$$\widetilde{P}_\theta(X'|X, a) = \mathop{\mathbb{E}}_{s\sim p(s|X)} \int_{X'} P(s'|s, a)\, ds' \tag{4.6}$$

where $p(s|X) = Z_\theta(X)^{-1}\delta^\theta(s)\mathbb{1}_X(s)$ is the probability of visiting state s conditioned by $s \in X$, and $\mathbb{1}_X(\cdot)$ is the indicator function for set X. In this way, we can simply estimate the abstract function via Monte Carlo estimation using the samples collected with π_θ, since the visited states are distributed as δ^θ. While this is enough for $\widetilde{R}_\theta(X, a)$, since $R(s, a)$ is usually known for all the state-action pairs (s, a) or designed by humans, estimating $\widetilde{P}_\theta(X'|X, a)$ for all the abstract states and actions without any knowledge of $P(\cdot|s, a)$ requires more effort. All we get from the agent-environment interactions is a finite set of samples (s, a, s'). This means that, for almost all the actions in $\mathcal{A}$, we do not have any samples. Fortunately, we can

3 Efficiently exploring high-dimensional state spaces is still a difficult problem: considering only the visited abstract states allows to contain the computational complexity of the algorithm but does not reduce the difficulty of covering an high-dimensional state space with exploration.

leverage our Lipschitz assumptions to fill in this missing information.

We first consider deterministic environments, where $P(\cdot|s,a)$ is a Dirac delta function that exactly identifies the next state s' resulting from a state-action pair (s,a), to be more precise:

$$P(s'|s,a) = \begin{cases} 1 & \text{if } s' = f(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

For simplicity, we denote this deterministic mapping as $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and write $s' = f(s,a)$. Let $\Delta(s,a) \coloneqq f(s,a) - s$ be the state difference obtained transitioning from s to s'. From Assumption 1 on Lipschitz MDPs and fixed an action $a \in \mathcal{A}$, we obtain:

$$\|\Delta(s,a) - \Delta(\widetilde{s},a)\| \leqslant L_\Delta \|s - \widetilde{s}\|, \tag{4.8}$$

for all $s, \widetilde{s} \in \mathcal{S}$, hence the state difference Δ is also Lipschitz continuous. A valid Lipschitz constant L_Δ that can be obtained from Assumption 1 is $(1 + L_P)$, as we show below. Since the Kantorovich distance between two Dirac delta functions is equal to the distance of their locations, we can use Assumption 1 to obtain a Lipschitz constant L_Δ for state-difference function Δ. For every $a \in \mathcal{A}$ and $s, \widetilde{s} \in \mathcal{S}$:

$$\begin{aligned} \|\Delta(s,a) - \Delta(\widetilde{s},a)\| &= \|f(s,a) - s - f(\widetilde{s},a) + \widetilde{s}\| \\ &\leqslant \|f(s,a) - f(\widetilde{s},a)\| + \|s - \widetilde{s}\| && (4.9) \\ &= \mathcal{K}\left(P(\cdot|s,a), P(\cdot|\widetilde{s},a)\right) + \|s - \widetilde{s}\| \\ &\leqslant L_P \|s - \widetilde{s}\| + \|s - \widetilde{s}\| && (4.10) \\ &\leqslant L_\Delta \|s - \widetilde{s}\| \qquad \text{with } L_\Delta = (1 + L_P), && (4.11) \end{aligned}$$

where (4.9) is from the triangle inequality and (4.10) is from Assumption 1.

Under additional assumptions, such as the linearity of the transition function $f(s,a)$, we can obtain constants lower than one or even equal to zero. The latter happens when the state difference depends only on the action, which can be realistic, or at least a good approximation, for some continuous control problems. We discuss these different settings and provide detailed computations of the Lipschitz constants in Section 4.4. When $L_\Delta = 0$, we can easily generate extra "fictitious" samples. For instance, given sample (s,a,s'), we can generate $(\widetilde{s},a,\widetilde{s}')$ for any other state $\widetilde{s} \in \mathcal{S}$ by setting $\widetilde{s}' = \widetilde{s} + (s' - s)$. Indeed, from the inequality in (4.11) and considering $L_\Delta = 0$, we obtain $\Delta(s,a) = \Delta(\widetilde{s},a)$. In this way, for each action a performed by policy π_θ in abstract state X, we can have several (fictitious) samples (one for each visited state $s \in X$). With these samples, we can simply estimate $\widetilde{P}_\theta(X'|X,a)$ as the ratio of the next states that fall into X' when we consider samples and fictitious

samples involving any state $s \in X$ and the action a. To be more precise, given a batch $\mathcal{D}$ of samples collected with π_θ and fictitious samples, the abstract transition function $\widetilde{P}_\theta$ is estimated as:

$$\widetilde{P}_\theta(X'|X,a) = \frac{\left|(s,a,s') \in \mathcal{D} : s \in X, s' \in X'\right|}{\left|(s,a,s') \in \mathcal{D} : s \in X\right|}. \tag{4.12}$$

When $L_\Delta \neq 0$, we can still use (4.8) to get a region containing $\widetilde{s}'$. From this, we can estimate lower and upper bounds on the transition function $\widetilde{P}$. As a result, the δ-MDP will be a BMDP [Givan, Leach, and Dean, 2000] instead of a regular one. In the case of stochastic environments, we follow a different approach for estimating the abstract transition kernel. We define a maximum-likelihood problem using data sampled from π_θ *and* fictitious samples. We then add special constraints to this problem as to enforce the Lipschitz continuity of $\widetilde{P}_\theta(X'|X,a)$ w.r.t. actions. The resulting problem is still convex and can be solved with standard optimization tools. Details on BMDP are provided in subsection 2.4.1, details on the algorithm involving BMDPs and the constrained optimization approach are provided in Section 4.4.

4.3.3 *Solving the abstract MDP*

In the previous paragraph, we have proposed a way to estimate the abstract transition function for state-action pairs not experienced by the agent, by exploiting the assumptions about the regularity of the environment. However, this still applies only to the actions actually performed by the agent. As a result, our approximation of the δ-MDP has a finite action set. This introduces a further model bias, but allows us to employ dynamic programming to find the optimal abstract policy. This kind of error can be reduced by increasing the number of collected episodes, or the control frequency. Furthermore, under our Lipschitz assumptions, the finite actions actually performed by the agent are supposedly good representatives for the other ones.

We solve this approximate δ-MDP via value iteration (subsection 2.2.1), with a stopping condition (introduced for computational reasons) on the max-norm distance between consecutive state-value functions. The value iteration on the δ-MDP is stopped when, said $V_k(X)$ the value function of state X at the $k-th$ iteration and ϵ the constant used as a threshold in the algorithm,

$$|V_k(X) - V_{k-1}(X)| < \epsilon \quad \forall X \in \mathcal{X}. \tag{4.13}$$

In the case of multiple optimal policies, we keep track of the entire set of optimal actions for each abstract state X so that we can set, for every state $s \in X$, a possibly different optimal action of X as target in the regression problem described in subsection 4.3.4. We do this

to minimize the difference between the deterministic policy π_θ before and after the update of its parameter θ, while we consider for the update only actions that are optimal in $\mathcal{X}$. We may need to evaluate abstract states for which no samples have been collected by the agent. We propose a risk-averse solution, which consists of considering these "unknown" abstract states as absorbing and set their value to the lowest known value.

4.3.4 *Projection*

Given the optimal abstract policy ρ^*, its concretization $\mathcal{C}\rho^*$ is easily obtained by copying the action selected by ρ^* for all the states of the same abstract state. We then need to represent $\mathcal{C}\rho^*$ in the space Π_Θ of the original policy π_θ. This projection phase can be approached as a regression problem, where the target for $\pi_{\theta'}(s)$ is $\mathcal{C}\rho^*(s)$. The problem consists in finding the parameter θ' that minimizes the following expression:

$$\|\mathcal{C}\rho - \pi_{\theta'}\|_{\delta^\theta} + \lambda \left\|\theta' - \theta\right\|, \tag{4.14}$$

where the expression is evaluated for all the collected samples (s, a, s'). When the abstract optimal policy ρ^* prescribes multiple optimal actions for a certain abstract state $X \in \mathcal{X}$, in order to facilitate the projection, we consider for each sample (s, a, s'), with $s \in X$, that $\mathcal{C}\rho(s)$ is the action with the smallest Euclidean distance from the action prescribed by $\pi_\theta(s)$, among the optimal actions related to the abstract state X.

The first term from (4.14) is the Root Mean Square Error (RMSE) loss, weighted by the future state distribution δ^θ. For differentiable policies (including neural networks), this loss can be minimized via gradient descent from the samples collected with π_θ. The second term from (4.14) penalizes solutions that are too far from the current policy parameters and it can be used as a regularization term in the regression problem. In practice, the regularization coefficient λ can be tuned as a meta-parameter. Note that, without this projection step, there would be no room for further policy improvement, as the next δ-MDP would have just one action per abstract state. The projection error actually allows the agent to evaluate new actions and visit new regions of the state space.

4.4 ABSTRACT TRANSITION FUNCTION CONSTRUCTION

In this section, we provide additional details on the estimation of the δ-MDP outlined in Section 4.3.2 and used to exploit passive exploration and update the deterministic policy of the agent. We describe the different approaches that we propose to estimate the abstract tran-

sition function $\widetilde{P}$, these approaches depend on the nature of the environment considered in the task. We inspect environments with an increasing level of difficulty: in subsection 4.4.1 we derive two Lipschitz constants that can be useful to estimate the arriving state in deterministic linear environments, in subsection 4.4.2 we propose to estimate the δ-MDP with a BMDP when the environment is deterministic but non-linear, in subsection 4.4.3 we build a constrained optimization problem to ensure a property of $\widetilde{P}$ in the case the environment is stochastic.

4.4.1 *Deterministic linear environments*

We call *linear* a deterministic environment in which $f(s, a)$ is linear w.r.t. the input variables s and a. For these environments, we can write $f(s, a) = As + Ba$, where A and B are (possibly unknown and then estimated) constant matrices. With $f(s, a)$ defined in this way, we can obtain a Lipschitz constant for Δ possibly smaller than the one derived in (4.11):

$$\begin{aligned} \|\Delta(s, a) - \Delta(\widetilde{s}, a)\| &= \|As + Ba - s - (A\widetilde{s} + Ba - \widetilde{s})\| \\ &\leqslant L_\Delta \|s - \widetilde{s}\| \qquad \text{with } L_\Delta = \|A - \mathbb{I}\|, \end{aligned} \tag{4.15}$$

where $\|A - \mathbb{I}\|$ is the induced matrix norm of $A - \mathbb{I}$ compatible with the vector norm considered. In the case the latter is the Euclidean norm, the induced matrix norm is the *spectral norm*. In general, the constant obtained in (4.15) can be smaller than $(1 + L_P)$ and even equal to zero. When $\Delta(s, a)$ depends only on the performed action a and not on the starting state s, $A = \mathbb{I}$ and $L_\Delta = 0$. From (4.11), if we consider $L_\Delta = 0$ instead of $L_\Delta = (1 + L_P)$, we obtain the equality $\Delta(s, a) = \Delta(\widetilde{s}, a)$ that allows to exactly predict the arriving state $\widetilde{s}' = f(\widetilde{s}, a)$ for an un-sampled pair $(\widetilde{s}, a)$ as $\widetilde{s}' = \widetilde{s} + \Delta(s, a)$, from a sampled pair (s, a).

When $\Delta(s, a)$ also depends on the starting state s, the matrix A is different from the identity matrix $\mathbb{I}$. If we know the matrix A or we have high confidence on its estimation, we can use the matrix to calculate the arriving state $\widetilde{s}'$ for an un-sampled pair $(\widetilde{s}, a)$. Reasoning on a generic dimension i of the state space, we obtain:

$$\begin{aligned} \Delta(s, a)_i &= (A - \mathbb{I})_i \cdot s + (B)_i \cdot a \\ \big(\Delta(s, a) - \Delta(\widetilde{s}, a)\big)_i &= (A - \mathbb{I})_i \cdot (s - \widetilde{s}), \end{aligned} \tag{4.16}$$

where $(A - \mathbb{I})_i$ and $(B)_i$ are the $i-th$ rows of matrices $(A - \mathbb{I})$ and B and $\cdot$ is the scalar product between vectors. We can derive the i-th dimension of the arriving state $\widetilde{s}'$ as:

$$\widetilde{s}'_i = \widetilde{s}_i + \big(s'_i - s_i\big) - (A - \mathbb{I})_i \cdot (s - \widetilde{s}). \tag{4.17}$$

Unfortunately, when we are not able to estimate with precision the matrix A we cannot generate "fictitious" samples exactly. However, we exploit the inequality in (4.11) to obtain an interval (or a higher-dimensional region) including the unknown $\widetilde{s'} = f(\widetilde{s}, a)$, instead of the exact point $\widetilde{s'}$. Here we show how to obtain the interval in the simple case of one-dimensional states:

$$\begin{array}{rcccl} -L_\Delta |s - \widetilde{s}| & \leqslant & \Delta s(s,a) - \Delta s(\widetilde{s},a) & \leqslant & L_\Delta |s - \widetilde{s}| \\ \Delta(s,a) - L_\Delta |s - \widetilde{s}| & \leqslant & \Delta(\widetilde{s},a) & \leqslant & \Delta(s,a) + L_\Delta |s - \widetilde{s}| \\ \Delta(s,a) - L_\Delta |s - \widetilde{s}| & \leqslant & f(\widetilde{s},a) - \widetilde{s} & \leqslant & \Delta(s,a) + L_\Delta |s - \widetilde{s}| \,. \end{array} \tag{4.18}$$

From (4.18) we derive:

$$\widetilde{s'} = f(\widetilde{s}, a) \in \Big[\widetilde{s} + \Delta(s,a) - L_\Delta |s - \widetilde{s}| \,; \quad \widetilde{s} + \Delta(s,a) + L_\Delta |s - \widetilde{s}|\Big], \tag{4.19}$$

where L_Δ is the exact (if known) or an estimated Lipschitz constant and $|\cdot|$ is the absolute value operator. When the state space is $N-$dimensional with $N \geqslant 2$, the interval that we estimate is an hyperrectangle. Since $|\Delta(s,a) - \Delta(\widetilde{s},a)|_i \leqslant \|\Delta(s,a) - \Delta(\widetilde{s},a)\|$, for each dimension $i \in [1; N]$ we apply inequalities from (4.11) and (4.18) to derive the $i-th$ bound of the hyperrectangle as:

$$\widetilde{s'}_i \in \Big[\widetilde{s}_i + \Delta(s,a)_i - L_\Delta \|s - \widetilde{s}\| \,; \quad \widetilde{s}_i + \Delta(s,a)_i + L_\Delta \|s - \widetilde{s}\| \Big], \tag{4.20}$$

where L_Δ is an estimation of the Lipschitz constant in (4.15) and $\|\cdot\|$ is the Euclidean norm operator.

In tasks where a lot of precision is required in order to estimate $\widetilde{s'}$, the computed intervals could be not tight enough to represent $\widetilde{s'}$ with the required accuracy. We can increase the precision by computing a second interval on $\widetilde{s'}$ and considering the intersection of the two intervals on $\widetilde{s'}$ as the most precise estimate. The second interval is calculated by considering two state-action pairs having the *same* state and different *actions*: the sampled pair $(\widetilde{s}, \widetilde{a})$ and the un-sampled pair $(\widetilde{s}, a)$.

$$\begin{aligned} \|\Delta(\widetilde{s},a) - \Delta(\widetilde{s},\widetilde{a})\| &= \|A\widetilde{s} + Ba - \widetilde{s} - (A\widetilde{s} + B\widetilde{a} - \widetilde{s})\| \\ &\leqslant L_{\Delta(a)} \|a - \widetilde{a}\| \qquad \text{with } L_{\Delta(a)} = \|B\| \,. \end{aligned} \tag{4.21}$$

Similarly to (4.18), we obtain the new interval containing $\widetilde{s'} = f(\widetilde{s}, a)$ in the case of one-dimensional states:

$$\widetilde{s'} \in \Big[\widetilde{s} + \Delta(\widetilde{s},\widetilde{a}) - L_{\Delta(a)} \|a - \widetilde{a}\| \,; \quad \widetilde{s} + \Delta(\widetilde{s},\widetilde{a}) + L_{\Delta(a)} \|a - \widetilde{a}\| \Big], \tag{4.22}$$

with $L_{\Delta(a)} = \|B\|$. For the N-dimensional case, we can apply the same reasoning used previously.

Hence, given (real) samples (s_1, a_1, s'_1) and (s_2, a_2, s'_2), we can generate fictitious next states for both (s_1, a_2) and (s_2, a_1), intersecting two intervals for each of them.

4.4.2 *Deterministic non-linear environments*

When the (deterministic) transition function is non-linear, we are not able to exactly predict the next state of un-sampled state-action pairs. However, we can still exploit the Lipschitz assumptions to obtain upper and lower bounds (or multi-dimensional regions) for the next-states. To do so, we can consider $L_\Delta = (1 + L_P)$, as shown in (4.11) for general deterministic transitions, or we can estimate a smaller L_Δ from data. The result is an abstract BMDP (see subsection 2.4.1 for details) instead of a regular one.

If we have $L_\Delta \neq 0$ and we estimate with (4.11) the fictitious arriving state $\widetilde{s'} = f(\widetilde{s}, a)$, where $(\widetilde{s}, a)$ is an unseen state-action pair, we obtain a region including $\widetilde{s'}$ instead of the exact point $\widetilde{s'}$. The information provided by the estimation of the fictitious arriving states is not sufficient to compute an exact $\widetilde{P}_\theta(X'|X, a)$ for each abstract state-action pair (X, a) of the δ-MDP. The value of each $\widetilde{P}_\theta(X'|X, a)$ is a range of probabilities, whose lower and upper bounds are computed by evaluating all the estimated intervals related to some $\widetilde{s'} = f(\widetilde{s}, a)$, where $\widetilde{s} \in X$. The intervals estimated for fictitious arriving states can be entirely contained in a single abstract state $X' \in \mathcal{X}$ or they can overlay multiple abstract states of $\mathcal{X}$. We estimate the lower bound $\underline{\widetilde{P}}_\theta(X'|X, a)$ as the ratio of the intervals entirely contained into X' and the upper bound $\overline{\widetilde{P}}_\theta(X'|X, a)$ as the ratio of the intervals that overlay X'. To be more precise, given a batch $\mathcal{D}$ of samples collected with π_θ and fictitious samples $(s, \widetilde{a}, X'_\updownarrow)$, with $X'_\updownarrow$ representing the possible arriving abstract states X' according to the collected samples or the estimation, the bounds of the abstract transition function $\widetilde{P}_{\updownarrow,\theta}$ are estimated as:

$$\underline{\widetilde{P}}_\theta(X'|X, a) = \frac{\left|(s, a, X'_\updownarrow) \in \mathcal{D} : s \in X, X'_\updownarrow = \{X'\}\right|}{\left|(s, a, X'_\updownarrow) \in \mathcal{D} : s \in X\right|}, \tag{4.23}$$

$$\overline{\widetilde{P}}_\theta(X'|X, a) = \frac{\left|(s, a, X'_\updownarrow) \in \mathcal{D} : s \in X, X' \in X'_\updownarrow\right|}{\left|(s, a, X'_\updownarrow) \in \mathcal{D} : s \in X\right|}. \tag{4.24}$$

Once we have $\widetilde{P}_{\updownarrow,\theta}(X'|X, a)$, we can build the δ-MDP as a BMDP. According to [Givan, Leach, and Dean, 2000], the BMDP can be solved following an optimistic or a pessimistic criterion. Motivated by risk-aversion, we choose the second one, so as to obtain the solution of

the exact MDP whose transition function is the $\widetilde{P}_\theta(X'|X,a)$ that maximizes the probability of transitioning to the abstract states with the lowest value function.

4.4.3 *Stochastic environments*

For stochastic environments, we propose a different approach for estimating the abstract transition kernel from the collected data while exploiting regularities of the environment. To do so, we require a stronger assumption on the transition function P w.r.t. actions:

Assumption 3. *For all $s \in \mathcal{S}$ and $a, \widetilde{a} \in \mathcal{A}$:*

$$\mathrm{TV}\left(P(\cdot|s,a), P(\cdot|s,\widetilde{a})\right) \leqslant L_{\mathrm{TV}} \left\|a - \widetilde{a}\right\|, \tag{4.25}$$

for some positive real constant L_{TV}, where $\mathrm{TV}(\cdot,\cdot)$ denotes the Total Variation (TV) distance defined in (2.32).

We can use this assumption to derive a Lipschitz constant for the abstract transition function:

$$\begin{aligned}
\left|\widetilde{P}(X'|X,a) - \widetilde{P}(X'|X,\widetilde{a})\right| &\leqslant \int_X w(s) \int_{X'} \left|P(s'|s,a) - P(s'|s,\widetilde{a})\right| ds'\, ds \\
&\leqslant \int_X w(s) \int_{\mathcal{S}} \left|P(s'|s,a) - P(s'|s,\widetilde{a})\right| ds'\, ds \qquad (4.26) \\
&= 2\int_X w(s)\, \mathrm{TV}\left(P(\cdot|s,a), P(\cdot|s,\widetilde{a})\right) ds \qquad (4.27) \\
&\leqslant L_{\widetilde{P}} \left\|a - \widetilde{a}\right\| \qquad \text{with } L_{\widetilde{P}} = 2L_{\mathrm{TV}}, \qquad (4.28)
\end{aligned}$$

where (4.26) extends the inner integral to the entire state space $\mathcal{S}$ (the integrands are all non-negative) and (4.27) is from the definition of the total variation distance. We formulate the estimation of $\widetilde{P}$ as a maximum likelihood problem, with additional constraints obtained from (4.28). Let $\mathcal{D}$ be the set of available (real or fictitious) samples, in the form of (s, a, s') tuples. Let $\mathcal{X}_\mathcal{D}$ denote the set of visited abstract states, i. e. for which there exists at least an $s \in \mathcal{X}$ that appears in $\mathcal{D}$. Moreover, let $\mathcal{A}_X$ be the set of actions from the dataset that were performed in a state of X, for $X \in \mathcal{X}_\mathcal{D}$. The objective of our optimization problem is to maximize the likelihood of the samples:

$$\max_{\widetilde{P} \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{A}|\times|\mathcal{X}|}} \prod_{X,X' \in \mathcal{X}_\mathcal{D}, a \in \mathcal{A}_X} \widetilde{P}(X'|X,a). \tag{4.29}$$

This can be reformulated as a convex program as follows:

$$\begin{aligned}
\min_{\widetilde{P} \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{A}|\times|\mathcal{X}|}} \quad & -\sum_{X,X' \in \mathcal{X}_\mathcal{D}, a \in \mathcal{A}_X} \log \widetilde{P}(X'|X,a) \\
\text{subject to} \quad & \widetilde{P}(X'|X,a) \geqslant 0 \qquad \forall X, X' \in \mathcal{X}_\mathcal{D}, a \in \mathcal{A}_X
\end{aligned}$$

$$\sum_{X' \in \mathcal{X}_{\mathcal{D}}} \widetilde{P}(X'|X, a) = 1 \qquad \forall X \in \mathcal{X}_{\mathcal{D}}, a \in \mathcal{A}_X. \tag{4.30}$$

We then add additional constraints that enforce in our estimate of $\widetilde{P}$ the regularity property we know to be true from (4.28):

$$\begin{aligned}
\min_{\widetilde{P} \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{A}| \times |\mathcal{X}|}} \quad & - \sum_{X, X' \in \mathcal{X}_{\mathcal{D}}, a \in \mathcal{A}_X} \log \widetilde{P}(X'|X, a) \\
\text{subject to} \quad & \widetilde{P}(X'|X, a) \geqslant 0 && \forall X, X' \in \mathcal{X}_{\mathcal{D}}, a \in \mathcal{A}_X \\
& \sum_{X' \in \mathcal{X}_{\mathcal{D}}} \widetilde{P}(X'|X, a) = 1 && \forall X \in \mathcal{X}_{\mathcal{D}}, a \in \mathcal{A}_X \\
& \left| \widetilde{P}(X'|X, a) - \widetilde{P}(X'|X, \widetilde{a}) \right| \\
& \qquad \leqslant L_{\widetilde{P}} \, \|a - \widetilde{a}\| && \forall X, X' \in \mathcal{X}_{\mathcal{D}}, a \in \mathcal{A}_X.
\end{aligned} \tag{4.31}$$

This is still a convex program and can be solved with standard optimization tools. In many cases, we can still generate fictitious samples as in the deterministic setting (see Section 6.2). These are simply added to the dataset $\mathcal{D}$.

5

THEORETICAL PROOFS

In this chapter we provide some theoretical proofs that sustain the goodness of the solution proposed with the DPO algorithm and provide further insights that justify the approach we used in the specific phases of the algorithm. Firstly, we enunciate some theorems in Section 5.1, together with some considerations that confirm the consistency of DPO. The proofs of these theorems are provided in Section 5.2.

5.1 THEOREMS

The overall goal that we consider in each iteration of the algorithm is the maximization of performance improvement $J(\theta') - J(\theta)$. From the following lower bound we obtain a justification, under Lipschitz assumptions, for the approach we propose in DPO:

Theorem 5.1. For any deterministic parametric policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$, state partition $\mathcal{X}$ and deterministic abstract policy $\rho : \mathcal{X} \to \mathcal{A}$, let $\pi_{\theta'} = \mathcal{P}_{\Pi_\Theta}(\mathcal{C}\rho)$, where $\mathcal{P}_{\Pi_\Theta}$ is the projection operator that projects policies in the policy space Π_Θ and $\mathcal{C}$ is the "concretization" operator that converts abstract policies in concrete policies. Then, under Assumptions 1 and 2:

$$J(\theta') - J(\theta) \geqslant \frac{1}{1-\gamma} \sum_{X \in \mathcal{X}} \int_X \delta^\theta(s) A^\theta(s, \rho(X))\, ds$$
$$- \frac{L_{Q^\theta}}{1-\gamma} \|\mathcal{C}\rho - \pi_{\theta'}\|_{\delta^\theta} - \frac{L_{\text{shift}}}{1-\gamma} \left\|\theta' - \theta\right\|,$$

where $L_{\text{shift}} = L_{\delta^\theta}\left(L_{Q^\theta}\left(1 + L_{\pi_{\theta'}}\right) + L_{V^\theta}\right)$, L_{δ^θ} is from Lemma 2.2, and L_{Q^θ}, L_{V^θ} are from Lemma 2.1.

This lower bound on performance improvement can serve as a surrogate optimization objective. In particular, the first term provides a criterion for selecting the abstract policy ρ, while the remaining terms provide a principled way to project it back into the original policy space. We perform these two tasks separately, as we said in Section 4.2 and more specifically in subsection 4.3.3 and subsection 4.3.4. This is clearly a simplification, as the surrogate objective should be optimized jointly for ρ and θ'. First, let us consider the projection part:

$$\min_{\theta' \in \Theta} \|\mathcal{C}\rho - \pi_{\theta'}\|_{\delta^\theta} + \lambda \left\|\theta' - \theta\right\|, \tag{5.1}$$

where $\lambda = L_{\text{shift}}/L_{Q^\theta}$. The first term accounts for the difference between the piecewise-constant policy and the updated parametric policy. It would be minimized by an exact projection. Since the error is weighted by the future-state distribution, we can use the samples collected with the original deterministic policy to approximately perform the projection, which is a regression problem. The second term accounts for the distributional mismatch between the future-state distributions of the two parametric policies. It can be added as a regularization term in the regression (subsection 4.3.4). Now we focus on the problem of selecting the best abstract policy:

$$\max_{\rho:\mathcal{X}\to\mathcal{A}} \sum_{X\in\mathcal{X}} \int_X \delta^\theta(s) A^\theta(s, \rho(X))\, ds, \tag{5.2}$$

and show that the optimal policy of the δ-MDP is indeed a reasonable choice. This problem has no trivial solution, as we do not know the advantage function A^θ, and estimating it for all (infinite) states and actions with our limited samples is out of reach. Let us first define the equivalent maximization problem:

$$\max_{\rho:\mathcal{X}\to\mathcal{A}} \sum_{X\in\mathcal{X}} W^\rho(X), \tag{5.3}$$

where $W^\rho(X) = Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds$. $W^\rho(X)$ is obtained from (5.2), we observe that we can maximize the expression by independently maximize each term of the sum:

$$\max_{\rho:\mathcal{X}\to\mathcal{A}} \sum_{X\in\mathcal{X}} Z_\theta(X)^{-1} \int_X \delta^\theta(s) A^\theta(s, \rho(X))\, ds \tag{5.4}$$

$$\max_{\rho:\mathcal{X}\to\mathcal{A}} \sum_{X\in\mathcal{X}} Z_\theta(X)^{-1} \int_X \delta^\theta(s) \big(Q^\theta(s, \rho(X)) - V^\theta(s)\big)\, ds \tag{5.5}$$

$$\max_{\rho:\mathcal{X}\to\mathcal{A}} \sum_{X\in\mathcal{X}} Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds, \tag{5.6}$$

where in (5.4) $Z_\theta(X)^{-1}$ does not depend on ρ, (5.5) is from the definition of advantage function and (5.6) by observing that V^θ does not depend on ρ. This is equivalent to finding, for each abstract state $X \in \mathcal{X}$, the action that maximizes $W^\rho(X)$. In DPO, we solve the δ-MDP for an optimal (deterministic) abstract policy ρ^* (subsection 4.3.3), which maximizes the value function $V^\rho(X)$ for all $X \in \mathcal{X}$. The following Theorem establishes the similarity of W^ρ with V^ρ under the Lipschitz assumptions, justifying our approach:

Theorem 5.2. Fixed a deterministic parametric policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$ and a state partition $\mathcal{X}$, for any deterministic abstract policy $\rho : \mathcal{X} \to \mathcal{A}$, let $W^\rho(X) = Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds$. Then, under Assumptions 1 and 2, for all $X \in \mathcal{X}$, provided $\rho(X) \in \pi_\theta(X)$:

$$|W^\rho(X) - V^\rho(X)| \leqslant \frac{\gamma L_{V^\theta} D_{\max}}{1-\gamma},$$

where $\pi_\theta(X) \subseteq \mathcal{A}$ denotes the image of X under π_θ, i.e. the set of actions performed in the states $s \in X$ according to π_θ, L_{V^θ} is from Lemma 2.1 and $D_{max} = \max_{X \in \mathcal{X}} \{D(X)\}$.

An immediate consequence is that:

$$W^{\rho^*}(X) \geqslant V^{\rho^*}(X) - \frac{\gamma L_{V^\theta} D_{max}}{1-\gamma}, \tag{5.7}$$

i.e. for a sufficiently fine discretization (depending on the environment regularity) we can use the optimal policy ρ^* of the δ-MDP as an approximate maximizer of (5.3). This is also an approximately optimal solution, for ρ alone (fixed the projection operator), to our surrogate objective from Theorem 5.1. These theoretical bounds justify our definition of δ-MDP and how we use the optimal abstract policy to update the original parameters.

In addition, this analysis provides some insight into how choosing the partition $\mathcal{X}$ affects the optimization problem. From (5.7), it would seem that a finer discretization is always better. However, an important assumption of Theorem 5.2 suggests that this is not the case: for the error bound to hold, ρ must be chosen so that, for all $X \in \mathcal{X}$, $\rho(X)$ belongs to $\pi_\theta(X)$, the subset of actions performed by the original policy π_θ in the states of X. This additional constraint on the abstract policy is necessary to keep the discretization error under control, but it introduces a further *model bias* in the solution of the policy optimization problem, since the optimum over the complete action space may no longer be attainable. This bias is larger when the restricted action sets are smaller, which may result from very fine discretizations. Intuitively, we must guarantee a sufficient amount of variety among the candidate actions of each abstract state. This can be seen as a trade-off between (passive) exploration and precision. The state partition should be chosen so as to optimize this trade-off, which may require an adaptive discretization schedule. We leave the theoretical study of this problem as future work, instead providing a sensitivity analysis to grid resolution in Section 6.2.

5.2 PROOFS

In this section, we provide complete proofs for the formal statements presented in Section 5.1. In order to do that, we also introduce and prove some auxiliary lemmas that involve the advantage function $A^\theta(s,a) \coloneqq Q^\theta(s,a) - V^\theta(s)$, defined according to a policy π_θ. This function is Lipschitz w.r.t. states and actions:

Lemma 5.3. *Under Assumptions 1 and 2, for all $\theta \in \Theta$, $s, \widetilde{s} \in \mathcal{S}$ and $a, \widetilde{a} \in \mathcal{A}$:*

$$\left| A^\theta(s,a) - A^\theta(\widetilde{s},\widetilde{a}) \right| \leqslant L_{Q^\theta} \|a - \widetilde{a}\| + \left(L_{Q^\theta} + L_{V^\theta}\right) \|s - \widetilde{s}\|,$$

where L_{Q^θ} *and* L_{V^θ} *are from Lemma 2.1.*

Proof.

$$\begin{aligned}\left|A^\theta(s,a) - A^\theta(\widetilde{s},\widetilde{a})\right| &\leqslant \left|Q^\theta(s,a) - V^\theta(s) - Q^\theta(\widetilde{s},\widetilde{a}) + V^\theta(\widetilde{s})\right| \\ &\leqslant \left|Q^\theta(s,a) - Q^\theta(\widetilde{s},\widetilde{a})\right| + \left|V^\theta(s) - V^\theta(\widetilde{s})\right| \\ &\leqslant L_{Q^\theta}\left(\|s-\widetilde{s}\| + \|a-\widetilde{a}\|\right) + L_{V^\theta}\|s-\widetilde{s}\| \quad (5.8) \\ &= L_{Q^\theta}\|a-\widetilde{a}\| + \left(L_{Q^\theta} + L_{V^\theta}\right)\|s-\widetilde{s}\|,\end{aligned}$$

where (5.8) is from Lemma 2.1. ■

The following special case will be useful:

Lemma 5.4. *Under Assumptions 1 and 2, for all* $\theta, \theta' \in \Theta$ *and* $s \in \mathcal{S}$*:*

$$|A^\theta(s,\pi_{\theta'}(s)) - A^\theta(\widetilde{s},\pi_{\theta'}(\widetilde{s}))| \leqslant L_{A^\theta_{\theta'}}\|s-\widetilde{s}\|,$$

where $L_{A^\theta_{\theta'}} = \left(L_{Q^\theta}(1 + L_{\pi_{\theta'}}) + L_{V^\theta}\right)$*, and* L_{Q^θ}*,* L_{V^θ} *are from Lemma 2.1.*

Proof. From Lemma 5.3:

$$\begin{aligned}|A^\theta(s,\pi_{\theta'}(s)) - A^\theta(\widetilde{s},\pi_{\theta'}(\widetilde{s}))| &\leqslant L_{Q^\theta}\|\pi_{\theta'}(s) - \pi_{\theta'}(\widetilde{s})\| + \\ &\quad + \left(L_{Q^\theta} + L_{V^\theta}\right)\|s-\widetilde{s}\| \\ &\leqslant \left(L_{Q^\theta}(1 + L_{\pi_{\theta'}}) + L_{V^\theta}\right)\|s-\widetilde{s}\|, \quad (5.9)\end{aligned}$$

where the last inequality is from Assumption 2. ■

We can now prove the main Theorem 5.1:

Theorem 5.1. For any deterministic parametric policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$, state partition $\mathcal{X}$ and deterministic abstract policy $\rho : \mathcal{X} \to \mathcal{A}$, let $\pi_{\theta'} = \mathcal{P}_{\Pi_\Theta}(\mathcal{C}\rho)$, where $\mathcal{P}_{\Pi_\Theta}$ is the projection operator that projects policies in the policy space Π_Θ and $\mathcal{C}$ is the "concretization" operator that converts abstract policies in concrete policies. Then, under Assumptions 1 and 2:

$$\begin{aligned}J(\theta') - J(\theta) \geqslant \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_X \delta^\theta(s)A^\theta(s,\rho(X))\,ds \\ - \frac{L_{Q^\theta}}{1-\gamma}\|\mathcal{C}\rho - \pi_{\theta'}\|_{\delta^\theta} - \frac{L_{\text{shift}}}{1-\gamma}\|\theta'-\theta\|,\end{aligned}$$

where $L_{\text{shift}} = L_{\delta^\theta}\left(L_{Q^\theta}\left(1 + L_{\pi_{\theta'}}\right) + L_{V^\theta}\right)$, L_{δ^θ} is from Lemma 2.2, and L_{Q^θ}, L_{V^θ} are from Lemma 2.1.

Proof. From the Performance Improvement Lemma [Kakade and Langford, 2002]:

$$J(\theta') - J(\theta) = \frac{1}{1-\gamma}\int_{\mathcal{S}} \delta^{\theta'}(s)A^\theta(s,\pi_{\theta'}(s))\,ds$$

$$
\begin{aligned}
&= \frac{1}{1-\gamma}\int_{\mathcal{S}} \delta^{\theta}(s) A^{\theta}(s, \mathcal{C}\rho(s))\,\mathrm{d}s \\
&\quad + \frac{1}{1-\gamma}\int_{\mathcal{S}} \delta^{\theta}(s)\left(A^{\theta}(s, \pi_{\theta'}(s)) - A^{\theta}(s, \mathcal{C}\rho(s))\right)\,\mathrm{d}s \\
&\quad + \frac{1}{1-\gamma}\int_{\mathcal{S}} \left(\delta^{\theta'}(s) - \delta^{\theta}(s)\right) A^{\theta}(s, \pi_{\theta'}(s))\,\mathrm{d}s \\
&= \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_{X} \delta^{\theta}(s) A^{\theta}(s, \rho(X))\,\mathrm{d}s \\
&\quad + \frac{1}{1-\gamma}\int_{\mathcal{S}} \delta^{\theta}(s)\left(A^{\theta}(s, \pi_{\theta'}(s) - A^{\theta}(s, \mathcal{C}\rho(s))\right)\,\mathrm{d}s \\
&\quad + \frac{1}{1-\gamma}\int_{\mathcal{S}} \left(\delta^{\theta'}(s) - \delta^{\theta}(s)\right) A^{\theta}(s, \pi_{\theta'}(s))\,\mathrm{d}s \qquad (5.10) \\
&\geqslant \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_{X} \delta^{\theta}(s) A^{\theta}(s, \rho(X))\,\mathrm{d}s \\
&\quad - \frac{1}{1-\gamma} L_{Q^{\theta}} \int_{\mathcal{S}} \delta^{\theta}(s)\,\|\pi_{\theta'}(s) - \mathcal{C}\rho(s)\|\,\mathrm{d}s \\
&\quad + \frac{1}{1-\gamma}\int_{\mathcal{S}} \left(\delta^{\theta'}(s) - \delta^{\theta}(s)\right) A^{\theta}(s, \pi_{\theta'}(s))\,\mathrm{d}s \qquad (5.11) \\
&\geqslant \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_{X} \delta^{\theta}(s) A^{\theta}(s, \rho(X))\,\mathrm{d}s \\
&\quad - \frac{L_{Q^{\theta}}}{1-\gamma}\int_{\mathcal{S}} \delta^{\theta}(s)\,\|\pi_{\theta'}(s) - \mathcal{C}\rho(s)\|\,\mathrm{d}s \\
&\quad - \frac{L_{A^{\theta}_{\theta'}}}{1-\gamma}\mathcal{K}\left(\delta^{\theta'}, \delta^{\theta}\right) \qquad (5.12) \\
&= \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_{X} \delta^{\theta}(s) A^{\theta}(s, \rho(X))\,\mathrm{d}s \\
&\quad - \frac{L_{Q^{\theta}}}{1-\gamma}\|\pi_{\theta'}(s) - \mathcal{C}\rho(s)\|_{\delta^{\theta}} - \frac{L_{A^{\theta}_{\theta'}}}{1-\gamma}\mathcal{K}\left(\delta^{\theta'}, \delta^{\theta}\right) \\
&\geqslant \frac{1}{1-\gamma}\sum_{X\in\mathcal{X}}\int_{X} \delta^{\theta}(s) A^{\theta}(s, \rho(X))\,\mathrm{d}s \\
&\quad - \frac{L_{Q^{\theta}}}{1-\gamma}\|\pi_{\theta'}(s) - \mathcal{C}\rho(s)\|_{\delta^{\theta}} - \frac{L_{\delta} L_{A^{\theta}_{\theta'}}}{1-\gamma}\left\|\theta' - \theta\right\|,
\end{aligned}
$$

where (5.10) is from the fact that $\mathcal{X}$ is a partition of $\mathcal{S}$ and $\mathcal{C}\rho(s) = \rho(X)$ for all $s \in X$; (5.11) is from Lemma 5.3 with $\widetilde{s} = s$, $a = \pi_{\theta'}(s)$ and $\widetilde{a} = \mathcal{C}\rho(s)$; (5.12) is from Lemma 5.4 and the definition of the Kantorovich distance; and the last inequality is from Lemma 2.2. ■

We now establish a relationship between the surrogate objective $W^{\rho}(X)$ and the value function of the current policy in the original MDP:

Lemma 5.5. *Fixed a deterministic parametric policy $\pi_{\theta} : \mathcal{S} \to \mathcal{A}$ and a state partition $\mathcal{X}$, for any deterministic abstract policy $\rho : \mathcal{X} \to \mathcal{A}$, let*

$W^\rho(X) = Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds$. *Then, under Assumptions 1 and 2, for all* $X \in \mathcal{X}$, *provided* $\rho(X) \in \pi(X)$, *and for all* $s \in X$:

$$|V^\theta(s) - W^\rho(X)| \leqslant L_{V^\theta} D(X),$$

where L_{V^θ} *is from Lemma 2.1.*

Proof.

$$|V^\theta(s) - W^\rho(X)| = Z_\theta(X)^{-1} \left| \int_X \delta^\theta(s') \left(V^\theta(s) - Q^\theta(s', \rho(X)) \right) ds' \right| \tag{5.13}$$

$$= Z_\theta(X)^{-1} \left| \int_X \delta^\theta(s') \left(Q^\theta(s, \pi_\theta(s)) - Q^\theta(s', \rho(X)) \right) ds' \right|$$

$$\leqslant Z_\theta(X)^{-1} \int_X \delta^\theta(s') \left| Q^\theta(s, \pi_\theta(s)) - Q^\theta(s', \rho(X)) \right| ds'$$

$$\leqslant \left| Q^\theta(s, \pi_\theta(s)) - Q^\theta(s, \rho(X)) \right| + Z_\theta(X)^{-1} \int_X \delta^\theta(s') \left| Q^\theta(s, \rho(X)) - Q^\theta(s', \rho(X)) \right| ds' \tag{5.14}$$

$$\leqslant L_{Q^\theta} D(\pi_\theta(X)) + L_{Q^\theta} D(X) \tag{5.15}$$

$$\leqslant L_{Q^\theta}(1 + L_{\pi_\theta}) D(X) \tag{5.16}$$

$$= L_{V^\theta} D(X), \tag{5.17}$$

where (5.13) is from the definition of $W^\rho(X)$; (5.14) is obtained by applying the triangular inequality after adding and subtracting $Q^\theta(s, \rho(X))$; (5.15) is from Lemma 2.1 and the fact that both $\pi_\theta(s)$ and $\rho(X)$ belong to $\pi_\theta(X)$, and all the considered states belong to X; (5.16) if from Assumption 2 since:

$$\begin{aligned} D(\pi_\theta(X)) &= \sup_{a, \widetilde{a} \in \pi_\theta(X)} \|a - \widetilde{a}\| \\ &= \sup_{s, \widetilde{s} \in X} \|\pi_\theta(s) - \pi_\theta(\widetilde{s})\| \\ &\leqslant L_{\pi_\theta} \sup_{s, \widetilde{s} \in X} \|s - \widetilde{s}\| \\ &= L_{\pi_\theta} D(X); \end{aligned} \tag{5.18}$$

and (5.17) is from the definition of L_{V^θ} in Lemma 2.1. ∎

Finally, we can relate W^ρ to the value function of abstract policy ρ in the abstract MDP and prove Theorem 5.2.

Theorem 5.2. Fixed a deterministic parametric policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$ and a state partition $\mathcal{X}$, for any deterministic abstract policy $\rho : \mathcal{X} \to \mathcal{A}$, let $W^\rho(X) = Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds$. Then, under Assumptions 1 and 2, for all $X \in \mathcal{X}$, provided $\rho(X) \in \pi_\theta(X)$:

$$|W^\rho(X) - V^\rho(X)| \leqslant \frac{\gamma L_{V^\theta} D_{\max}}{1 - \gamma},$$

where $\pi_\theta(X) \subseteq \mathcal{A}$ denotes the image of X under π_θ, i.e. the set of actions performed in the states $s \in X$ according to π_θ, L_{V^θ} is from Lemma 2.1 and $D_{\max} = \max_{X \in \mathcal{X}} \{D(X)\}$.

Proof. First, by Bellman's equation:

$$\begin{aligned}
W^\rho(X) &= Z_\theta(X)^{-1} \int_X \delta^\theta(s) Q^\theta(s, \rho(X))\, ds \\
&= Z_\theta(X)^{-1} \int_X \delta^\theta(s) R(s, \rho(X))\, ds \\
&\quad + \gamma Z_\theta(X)^{-1} \int_X \delta^\theta(s) \int_S p(s'|s, \rho(X)) V^\theta(s')\, ds'\, ds \qquad (5.19) \\
&= \widetilde{R}_\theta(s, \rho(X)) \\
&\quad + \gamma Z_\theta(X)^{-1} \int_X \delta^\theta(s) \int_S p(s'|s, \rho(X)) V^\theta(s')\, ds'\, ds \qquad (5.20) \\
&= \widetilde{R}_\theta(s, \rho(X)) \\
&\quad + \gamma \sum_{X' \in \mathcal{X}} Z_\theta(X)^{-1} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X)) V^\theta(s')\, ds'\, ds, \\
& \qquad (5.21)
\end{aligned}$$

where (5.19) is from Bellman's equation for Q^θ, (5.20) is from the definition of $\widetilde{R}_\theta$, and (5.21) is from the fact that $\mathcal{X}$ is a partition of S. Hence:

$$\begin{aligned}
|W^\rho(X) - V^\rho(X)| &= \Bigg| \gamma \sum_{X' \in \mathcal{X}} Z_\theta(X)^{-1} \int_X \delta^\theta(s) \\
&\quad \int_{X'} p(s'|s, \rho(X)) V^\theta(s')\, ds'\, ds - \gamma \sum_{X' \in \mathcal{X}} \widetilde{P}_\theta(X'|X, \rho(X)) V^\rho(X') \Bigg| \qquad (5.22) \\
&= \gamma Z_\theta(X)^{-1} \Bigg| \sum_{X' \in \mathcal{X}} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X)) V^\theta(s')\, ds'\, ds \\
&\quad - \sum_{X' \in \mathcal{X}} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X))\, ds'\, ds V^\rho(X') \Bigg| \qquad (5.23) \\
&\leqslant \gamma Z_\theta(X)^{-1} \sum_{X' \in \mathcal{X}} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X)) \left| V^\theta(s') - V^\rho(X') \right| ds'\, ds \\
&\leqslant \gamma Z_\theta(X)^{-1} \sum_{X' \in \mathcal{X}} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X)) \left| V^\theta(s') - W^\rho(X') \right| ds'\, ds \\
&\quad + \gamma Z_\theta(X)^{-1} \sum_{X' \in \mathcal{X}} \int_X \delta^\theta(s) \int_{X'} p(s'|s, \rho(X)) \left| W^\rho(X') - V^\rho(X') \right| ds'\, ds \\
& \qquad (5.24) \\
&\leqslant \gamma L_{V^\theta} \sum_{X' \in \mathcal{X}} \widetilde{P}_\theta(X'|X, \rho(X)) D(X') \\
&\quad + \gamma \sum_{X' \in \mathcal{X}} \widetilde{P}_\theta(X'|X, \rho(X)) \left| W^\rho(X') - V^\rho(X') \right| \qquad (5.25)
\end{aligned}$$

$$\begin{aligned} &\leqslant \frac{\gamma L_{V^\theta} \sum_{X' \in \mathcal{X}} \widetilde{P}_\theta(X'|X, \rho(X)) D(X')}{1-\gamma} \\ &\leqslant \frac{\gamma L_{V^\theta} D_{\max}}{1-\gamma}, \end{aligned} \tag{5.26}$$

where (5.22) is from (5.21) and Bellman's equation for V^ρ (the abstract reward terms cancel out); (5.23) is from the definition of $\widetilde{P}_\theta$; (5.24) is obtained by applying the triangular inequality after adding and subtracting $W^\rho(X)$; (5.25) is from Lemma 5.5 and the definition of $\widetilde{P}_\theta$; (5.26) is by recursion on $|W^\rho(X) - V^\rho(X)|$; and the last inequality is from the definition of $D_{\max}$. ■

6 EXPERIMENTS

In this chapter, we test DPO on simulated continuous control tasks. Details on task and policy definitions are provided for each experiment: in Section 6.1 for the *Minigolf* experiment, in Section 6.2 for the *Double Integrator* experiment and in Section 6.3 for the *Robot Adaptation* experiment. In Section 6.4 we provide details on how experiments were performed, we also report the results of the tuning phase (previous to the experiments) performed on some hyperparameters. Hyperparameters for all the algorithms are selected via grid search, for DPO the regularization coefficient λ and the projection step size α are tuned in this way. Additional results that help to better understand the behavior of DPO are provided in Section 6.5.

6.1 MINIGOLF

We begin the experimental phase of this work with a one-dimensional task, adapted from [D'Oro et al., 2019], where the agent has to throw a ball into a hole by hitting it with a putter as few times as possible. The policy is an RBF network mapping the scalar state (the ball-hole distance) to the scalar action (the applied force). The agent receives a reward of -1 for every intermediate hit, and receives a penalty of -100 for overshooting, i. e. for throwing the ball over the hole. We refer to the latter event as a *failure* in the following. The ball is initialized at a random position at each episode. If not for the initialization, the environment is deterministic and has nonlinear dynamics, complicated by the variable friction of the course, which is proportional to the distance of the ball from the hole. We compare DPO with REINFORCE (see subsection 3.1.2 for more details on REINFORCE). The latter learns the mean and standard deviation of a Gaussian (or normal) policy, described in subsection 2.3.1. For DPO, we discretize the state space into 12 equally sized intervals. We use the BMDP approach with Lipschitz constant[1] $L_\Delta = 0.3$. We also evaluate DPO under the simplifying assumption $L_\Delta = 0$, obtaining better results in practice. The initial policy standard deviation and the learning rate for REINFORCE are tuned before running the final experiment. Details on hyperparameter selection are reported in Section 6.4.

TASK SPECIFICATIONS The state space is one-dimensional and represented by the interval $(0, 20]$. States whose value is greater than

1 We assume to know the Lipschitz constant, but it should be estimated from data or hand-tuned in practice.

20 are set to 20, states whose value is lower or equal to 0 cause the end of the episode. The action space is one-dimensional and represented by the interval $[10^{-5}, 5]$. Actions whose value is outside from the interval are clipped. The environment is deterministic, with a function $f(s, a)$ that exactly computes the arriving state s' from every pair (s, a):

$$s' = f(s, a) = s - at + 0.5 \times dt^2,$$

where $t = \frac{a}{d}$ and $d = \frac{5}{7} \times \text{friction}(s) \times 9.81$. Friction is computed as:

$$\text{frict}(s) = \text{friction\_low} + \frac{\text{friction\_high} - \text{friction\_low}}{s_{max} - s_{min}} \times s,$$

where $\text{friction\_low} = 0.131$, $\text{friction\_high} = 0.19$, $s_{max} = 20$ and $s_{min} = 0$. The Lipschitz constant L_Δ of the environment is computed as follows:

$$\Delta(s, a) = -0.5 \times \frac{a^2}{d(s)}; \quad \text{friction}(s) = 0.131 + 2.95 \times 10^{-3} \times s,$$

$$\begin{aligned} \|\Delta(s, a) - \Delta(\widetilde{s}, a)\| &= \left\| -0.5 \times a^2 \left(\frac{1}{7 \times \text{frict}(s)} - \frac{1}{7 \times \text{frict}(\widetilde{s})} \right) \right\| \\ &= \left\| -0.0714 \times a^2 \right\| \left\| \frac{2.95 \times 10^{-3} \times (\widetilde{s} - s)}{\text{frict}(s) \times \text{frict}(\widetilde{s})} \right\| \qquad (6.1) \\ &= L_\Delta \left\| s - \widetilde{s} \right\| \qquad \text{with } L_\Delta = 0.3, \qquad (6.2) \end{aligned}$$

where (6.2) is obtained from (6.1) considering the maximum possible value for actions and the minimum possible value for states, so as to maximize L_Δ. At any time step, the reward function depends only on the current state: the effect of every action performed is observable in the reward of the next step of the episode:

$$R(s) = \begin{cases} 0 & \text{if } s \in [-4, 0], \\ -100 & \text{if } s < -4, \\ -1 & \text{otherwise.} \end{cases}$$

The policy used for this task is a radial-basis network composed of four Gaussian functions ϕ_i, with constant hyper-parameters μ_i and σ_i. The action prescribed by the policy is computed as $a = \boldsymbol{\phi}(s)^\intercal \theta$, where $\theta \in \mathbb{R}^4$, the learned parameters, are the weights given to each Gaussian function. The initial values for θ are $[1, 1, 1, 1]$. The RBF hyperparameters are as follows:

$$\begin{gathered} \phi_i(s; \mu_i, \sigma_i) = \exp\left\{ -(s - \mu_i)^2 / (2\sigma_i^2) \right\}, \\ \mu_i \in (4, 8, 12, 16), \\ \sigma_i \in (4, 4, 4, 4). \end{gathered}$$

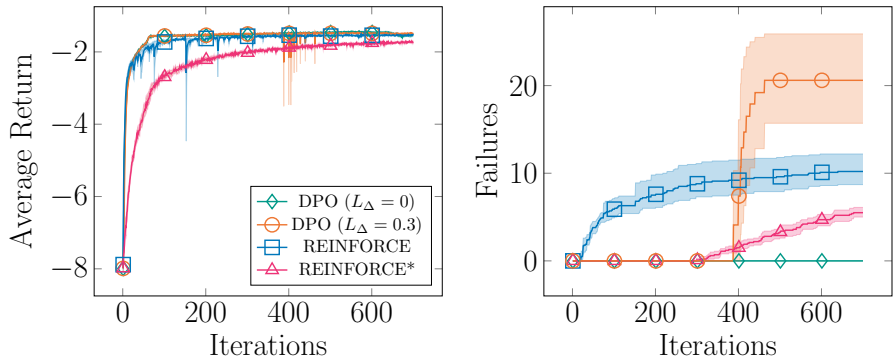

Figure 6.1: Minigolf results, averaged over 10 random seeds with 95% bootstrapped confidence intervals. On the left: average return ($\gamma = 0.99$) per iteration. On the right: number of failing episodes in a batch of 500 (the same legend applies).

In the case of REINFORCE, a Gaussian noise $\eta \sim \mathcal{N}(0, \sigma^2)$ is added to the action, where $\sigma = e^{\omega}$ and $\omega \in \mathbb{R}$ is an additional learned parameter. The initial value for σ is selected as a hyper-parameter. We set a discount factor of $\gamma = 0.99$ and a maximum task horizon of 20 time steps.

RESULTS Figure 6.1 shows average return and total failures over iterations (all algorithms use a constant batch size of 500 episodes per iteration). The performance of DPO is comparable with that of REINFORCE. However, the $L_\Delta = 0$ version is able to solve the task without any failure. This is not the case for REINFORCE, not even when the hyperparameters are explicitly selected as to minimize the number of failures (REINFORCE* in the figure). In fact, in this task, an almost-optimal stochastic policy has non-zero probability of overshooting. This source of risk is removed by DPO, which is entirely deterministic. However, without explicit safety constraints, a failing policy can still be learned, as happens in the $L_\Delta = 0.3$ case.

6.2 DOUBLE INTEGRATOR

Next, we consider a two-dimensional stochastic environment. In the Double Integrator task [Recht, 2018], a mass on a line must be brought to a target point by applying a force. The state is two-dimensional and includes distance from the goal and speed of the mass, both randomly initialized. The task is modeled as a special case of two-dimensional Linear-Quadratic Regulator (LQR) [Peters, 2002], which means the transition function is linear plus an additive zero-mean Gaussian noise and the optimal policy is also linear. The LQR can be applied in several situations, for instance [Recht, 2018] shows how to balance battery life versus speed for a quadrotor. A Gaussian noise on the output of the transition function makes the task stochastic. Hence,

we use the constrained-max-likelihood approach for DPO presented in subsection 4.4.3. We can still exploit the underlying linear dynamics of the task to easily generate fictitious samples. For each abstract state X, we compute a fictitious sample for each unseen pair $(\widetilde{s}, a)$, where $\widetilde{s} \in X$ and a is sampled from a state $s \in X$. The next states for fictitious samples are computed from the equation (4.15) using $L_\Delta = 0$, as if the noise was not present. In Double Integrator, the Gaussian noise is added to each dimension of the state independently from the other one. Then, we can define a separate constraint (similar to (4.31)) for each of the two dimensions of the state space, possibly with a different constant $L_{\widetilde{P}}$ for each dimension, to increase the precision of the abstract transition function (see Section A.1 for details).

TASK SPECIFICATIONS The environment has a two-dimensional state and a scalar action. Both the state dimensions and the actions are represented with real values in the interval $[-1, 1]$. The arriving state s' is computed with the following equation:

$$s' = As + Ba + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \nu^2),$$

where:

$$A = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ \frac{\tau}{mass} \end{bmatrix},$$

$\nu = 0.1$, $mass = 1$ and $\tau = 1$. Since the term ϵ is a Gaussian noise, $P(s'|s, a)$ will be a Gaussian distribution. The reward function is:

$$R(s) = -(s^T Q s + r a^2), \tag{6.3}$$

where:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \tag{6.4}$$

and $r = 0.1$. The policy used for this task is linear in the state, with initial parameters = $[-0.3, -0.3]$. Hence, the optimal policy is included among the achievable policies because we are considering a LQR task. For PGPE, a factored Gaussian hyper-policy $\mathcal{N}(\boldsymbol{\rho}, diag(\boldsymbol{\sigma}^2))$ is defined over the two-dimensional parameter space, where $\boldsymbol{\sigma} = e^{\boldsymbol{\omega}}$, $\boldsymbol{\rho}, \boldsymbol{\omega} \in \mathbb{R}^2$ are learned parameters, and $\boldsymbol{\rho}$ is always initialized to $[-0.3, -0.3]$. Both the components of $\boldsymbol{\sigma}$ are initialized to the same scalar σ, selected as a hyper-parameter. We set a discount factor of $\gamma = 0.95$ and a maximum task horizon of 20 time steps.

RESULTS We study the performance of DPO under different grid sizes. We also report the performance of PGPE [Sehnke et al., 2008] as

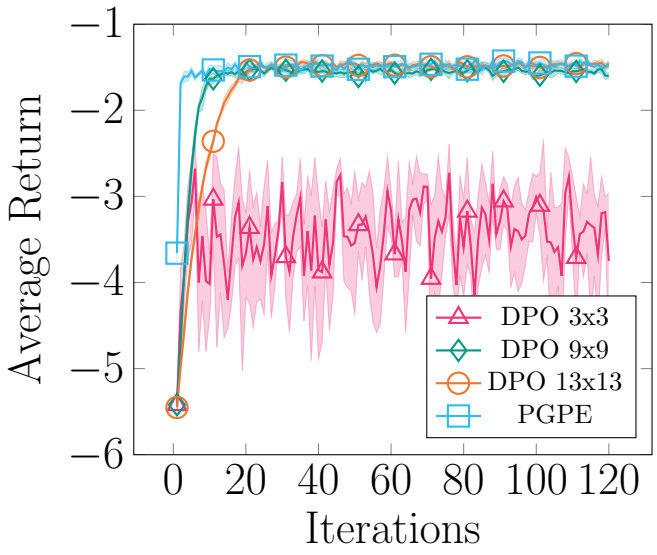

Figure 6.2: Double Integrator: average return ($\gamma = 0.95$) per iteration, averaged over 5 random seeds with 95% bootstrapped confidence intervals.

a reference. As shown in Figure 6.2, as expected, a very coarse discretization (3×3) leads to large errors and instabilities, while a very fine one (13×13) limits exploration, resulting in slower convergence than 9×9. PGPE is very fast to find the optimal deterministic policy, but its randomization over policy parameters results in stochastic behavior.

6.3 ROBOT ADAPTATION

Finally, we consider a more realistic scenario. A mobile robot on a flat surface has to reach goal areas specified by the user. The task is adapted from Safety Gym [Ray, Achiam, and Amodei, 2019] and based on MuJoCo [Todorov, Erez, and Tassa, 2012]. The robot has a speed-based actuator for turning and a force-based one for going forward and backward, resulting in a two-dimensional action. Its state is composed of typical sensor observations (accelerometer, velocimeter, gyroscope, magnetometer) and a compass pointing to the current goal, for a total of 9 state variables. The goal is a randomly placed circular area, and the robot is rewarded for approaching it through a dense reward signal. Whenever reached, the goal is placed at a new random position. We initialize the robot with a good deterministic linear policy, learned with PGPE, whose performance (averaged over 1000 test episodes) is reported as a black dashed line in Figure 6.3. Imagine this policy has been learned in a controlled environment, then the robot has been deployed in a facility where random actions are not permitted.

Then, a fault occurs, in the form of a fixed offset (20°) on the angle measured by the goal compass. The lower performance of the original policy after the fault is reported as a red dotted line in Figure 6.3. A corresponding difference in the agent's behavior can be appreciated from the animation feature provided by MuJoCo to observe the simulation (see subsection 6.5.3). The environment is deterministic. We also make the simplifying assumption $L_\Delta = 0$ and show the results for a hand-tuned regularization parameter λ. For each policy update, a batch of 10 episodes of 200 steps each is collected, and we show the average return over each batch.

TASK SPECIFICATIONS This is a custom environment built using the Safety Gym library [Ray, Achiam, and Amodei, 2019], which relies on the MuJoCo simulator [Todorov, Erez, and Tassa, 2012]. The task is composed by a mobile robot that moves on a flat surface with the aim of reaching a randomly placed goal area. The state space considered in the task is composed by 9 variables that represent the measurements of some sensors mounted on the robot. The robot is equipped with four standard sensors (accelerometer, velocimeter, gyroscope, magnetometer) and a compass that detects the orientation of the robot with regard to the goal area. Since the environment is flat, some of the data coming from the sensor (e. g. vertical acceleration) are unnecessary and are simply discarded in order to reduce the dimensionality of the state space. The reduced state is composed by the following variables, reported here in the order in which they are considered in the implementation. All measurements are in the robot's frame of reference:

- Linear acceleration in the plane (coordinates of a 2D vector).
- Cosine and sine of the angle observed by the compass, which indicates the direction of the goal.
- Angular velocity of the robot (w.r.t. yaw);
- Magnetic flux observed by the magnetometer (coordinates of a 2D vector).
- Linear velocity in the plane (coordinates of a 2D vector).

Note that the robot has no information regarding its position in the world's frame of reference, which is not necessary for this task. The reward function is defined as:

$$R(s_t, a_t) = r \times \Delta distance\_from\_goal(t),$$

where $r = 1$, $\Delta distance\_from\_goal = distance(t) - distance(t-1)$ and $distance(t)$ measures the distance of the robot from the goal at time step t. To compute $\Delta distance\_from\_goal$, the environment keeps a memory of previous distances. This, together with the fact

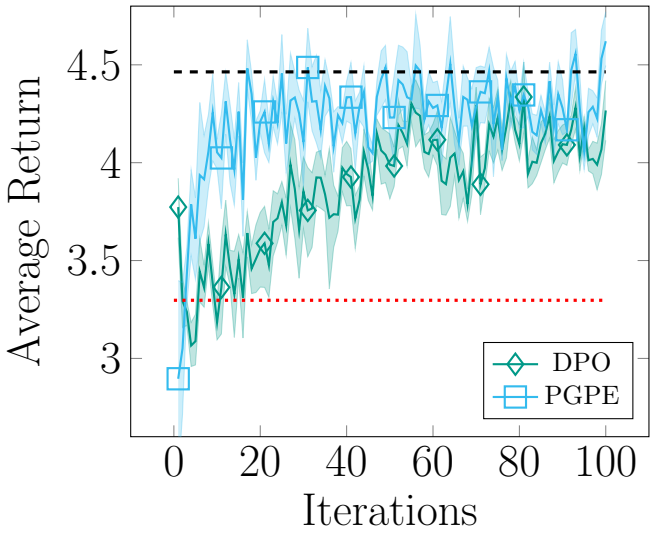

Figure 6.3: Robot Adaptation: average return (undiscounted) per iteration, averaged over 10 random seeds with 95% bootstrapped confidence intervals.

that $\mathtt{distance}(t)$ cannot be computed from the available observations, prevents us to compute $R(s, a)$ for all states and actions, as assumed in subsection 4.3.2. However, since $R(s_t, a_t)$ is independent from a_t, we can estimate $R(X, a)$ for any abstract state X and action a simply by averaging the rewards obtained in the states of X, regardless of the performed action. The delayed effect of actions on rewards is captured by also estimating the abstract transition function anyway. For this task, we use a linear policy. The initial parameters, learned with PGPE and representing an almost optimal policy before the compass fault, are reported here:

$$\theta_0 = \begin{bmatrix} 0.11 & 0.02 & 4.30 & 0.11 & 0.02 & 0.11 & 0.02 & -0.18 & -0.04 \\ -0.01 & 0.49 & 0.11 & -12.07 & 1.07 & -0.05 & -0.22 & 0.04 & -0.40 \end{bmatrix}.$$

For PGPE, a factored Gaussian hyper-policy $\mathcal{N}(\rho, \mathtt{diag}(\sigma^2))$ is defined over the 18-dimensional parameter space, where $\sigma = e^{\omega}$, $\rho, \omega \in \mathbb{R}^{18}$ are learned parameters, and ρ is initialized as θ_0. All the components of σ are initialized to the same scalar σ, selected as a hyper-parameter. We set a discount factor of 1 and a maximum task horizon of 200 (2000 in some experiments) time steps.

RESULTS Using DPO, the agent can adapt the policy parameters to the environmental change, as shown in Figure 6.3. The performance of PGPE is also reported as a reference. Both algorithms are able to fine-tune the policy back to its original performance. However, DPO does that without action randomization.

6.4 EXPERIMENTAL DETAILS

In this section we provide further details on how the experiments presented in Section 6.1, Section 6.2 and Section 6.3 were conducted.

6.4.1 *Hyper-parameter tuning*

For each task, we have to tune some hyper-parameters before applying the different algorithms. We do so via *grid search*. We define a set of reasonable values for these parameters and we fix a criterion for choosing the best parameters to be used in the algorithm. We compute the sum over n iterations of the performance measure (estimated from the collected samples at each iteration of the algorithm) and average it over m independent runs, performed with different random seeds. The values of n and m for the different experiments, together with other details, are reported in Table 6.1. The results of the grid search are reported in Tables 6.2-6.7. The bold values in the tables represent the combination of parameters used to perform the final experiments. For REINFORCE [Williams, 1992a], the selected hyper-parameters are the step size α and the policy initial standard deviation σ. For DPO, the learning rate α of the projection phase and the regularization coefficient λ (see Section 4.3). For PGPE [Sehnke et al., 2008], the step size α and the hyper-policy initial standard deviation σ.

Task	**Algorithm**	n	m	N	H	γ	$\lvert\mathcal{X}\rvert$
Minigolf	DPO	300	5	500	20	0.99	12
	REINFORCE	300	5	500	20	0.99	–
Double Integrator	DPO	60	3	500	20	0.95	9^2
	PGPE	100	5	500	20	0.95	–
Robot Adaptation	DPO	500	5	1	2000	1	5^9
	PGPE	100	5	10	200	1	–

Table 6.1: Configurations used for hyper-parameter tuning. We denote with n the number of iterations (policy updates), with m the number of independent runs, with N the batch size, with H the task horizon, with γ the discount factor and with $\lvert\mathcal{X}\rvert$ the number of abstract states.

MINIGOLF For REINFORCE (Table 6.3) we performed the final experiment with two different combinations of parameters: $(\alpha = 0.05, \log\sigma = -2)$ selected according to the criterion used in the tuning phase and $(\alpha = 0.005, \log\sigma = -3)$, that provided the lowest number of *failures* in the tuning phase.

α \ λ	0.0005	0.001	0.005
0.001	**-524.69**	-526.51	-538.16
0.005	-529.29	-530.28	-533.53
0.01	-572.35	-573.54	-566.87

Table 6.2: Grid search for DPO on Minigolf.

α \ $\log \sigma$	-4	-3	-2
0.005	-897.95	**-883.72**	-876.55
0.01	-1011.65	-744.81	-734.89
0.05	-4971.86	-2965.25	**-551.86**

Table 6.3: Grid search for REINFORCE on Minigolf.

DOUBLE INTEGRATOR Hyper-parameter selection was performed for the 9×9 discretization, and kept fixed in the final experiments for other discretizations. We used a fixed α (also for the next task) since its effects on performance are negligible.

α \ λ	0.0001	0.0005	0.001
0.025	**-107.52**	-108.55	-110.13

Table 6.4: Grid search for DPO on Double Integrator.

α \ σ	0.1	0.5	1
0.1	-187.86	-184.57	-190.18
0.5	-174.69	-158.39	-161.64
1	-169.29	**-153.93**	-159.28

Table 6.5: Grid search for PGPE on Double Integrator.

ROBOT ADAPTATION To verify the convergence of DPO, we run it on a longer number of iterations than PGPE. We also used a single, longer episode per iteration, which is more realistic. However, these settings were aligned for the final experiments, to make the comparison fair.

α \ λ	0.001	0.01	0.05
0.005	20658.70	23163.38	**23553.37**

Table 6.6: Grid search for DPO on Robot Adaptation.

α \ σ	0.1	0.5	1
0.1	337.33	378.27	358.08
0.5	372.23	413.78	402.46
1	395.88	**419.37**	405.55

Table 6.7: Grid search for PGPE on Robot Adaptation.

6.4.2 *Final Experiments*

The results of the final experiments (averaged over multiple runs performed with separate random seeds) are reported in the figures in Section 6.1, Section 6.2 and Section 6.3. We recap the final configurations in Table 6.8 for the sake of reproducibility.

6.5 QUALITATIVE RESULTS

In this section, we provide additional results on DPO, with the purpose of better understanding its workings in practice.

6.5.1 *Abstract state-space visualization in Robot Adaptation*

The Robot Adaptation task has a 9-dimensional state space, discretized into a Cartesian grid of 5 buckets per dimension. The total number of abstract states is very large ($\sim 10^6$) and could cause computational issues. However, as mentioned, only a subset of the abstract state space is actually visited by the agent and needs to be considered in the computations of DPO. The colormaps of Figures 6.4-6.7 show the visitation frequencies, for different iterations of the algorithm, of the abstract states, projected onto two of the most relevant state variables (the goal compass measurements) for visualization purposes. Axes labels are the bucket indexes, and the values are the ratios of total visits over 2000 time steps, averaged over 10 episodes. We can see that some abstract states are not visited at all. These are ignored in the estimation of the abstract transition and reward functions, in the value iteration phase and in the projection phase. Note how the

Task	Alg	α	λ	σ	n	m
Minigolf	DPO	0.001	0.0005	–	700	10
	REINFORCE	0.05	–	e^{-2}	700	10
	REINFORCE*	0.005	–	e^{-3}	700	10
Double Integrator	DPO	0.025	0.0001	–	120	5
	PGPE	1	–	0.5	120	5
Robot Adaptation	DPO	0.005	0.05	–	100	10
	PGPE	1	–	0.5	100	10

Task	Alg	N	H	γ	$\lvert\mathcal{X}\rvert$
Minigolf	DPO	500	20	0.99	12
	REINFORCE	500	20	0.99	–
	REINFORCE*	500	20	0.99	–
Double Integrator	DPO	500	20	0.95	$\{3^2, 9^2, 13^2\}$
	PGPE	500	20	0.95	–
Robot Adaptation	DPO	10	200	1	5^9
	PGPE	10	200	1	–

Table 6.8: Configurations used for hyper-parameter tuning, including hyper-parameters α, λ and σ. We denote with n the number of iterations (policy updates), with m the number of independent runs, with N the batch size, with H the task horizon, with γ the discount factor and with $\lvert\mathcal{X}\rvert$ the number of abstract states.

visits change across iterations, and some previously ignored abstract states must be added to the abstract MDP. The central states are never visited, as they correspond to unfeasible configurations (the two variables are the sine and cosine of an angle and their squares must always sum to one). We can also see an improvement in the agent's behavior: abstract state $(0, 2)$ (the most visited one) corresponds to the agent facing the goal. After 99 iterations, the visitation frequency of this state has significantly increased, meaning that the agent is able to point to the goal very quickly.

6.5.2 *Abstract policy visualization in Minigolf*

In Figures 6.8-6.15 we visualize the policies (abstract and concrete) learned by DPO, for different iterations of the Minigolf experiment. The plots have the original state space on the horizontal axis and the continuous action space on the vertical one (both are one-dimensional in this task). The red line represents the deterministic policy, which is

linear in a vector of Gaussian radial-basis features (hence non-linear in the state). White boxes correspond to the 12 abstract states and represent the restricted action space considered in the value iteration phase. Blue boxes represent the range of optimal actions obtained via value iteration. Note that, in this task, several optimal abstract policies exist. These are all considered as valid targets in the projection phase, in order to make the projection to the original policy space easier. The dotted line denotes the maximum action allowed by the environment (larger actions are clipped). When the box is placed above the dotted line, then, all the actions should be considered as optimal. This is not the case of the rightmost box in Figures 6.14 and 6.15, where only a subset of the actions above the dotted line is considered as optimal. Indeed, by solving the task with $L_\Delta = 0$, we assume that the friction has the same effect everywhere in the abstract state, which is not true in general. As a result, actions sampled where the friction is lower are evaluated better than they are.

6.5.3 *Agent behavior visualization in Robot Adaptation*

Visualizing the robot's behavior in different phases of the algorithm is useful to understand both the effect of the simulated fault and the nature of the performance drop it introduces (highlighted with horizontal lines in Figure 6.3). We uploaded four videos here[2] that show how the robot behaves in different situations. In the first video, we observe the behavior of the robot when the actions are prescribed according to an almost-optimal policy (learned with PGPE) and no fault is introduced in the task. The robot easily turns in the direction of the goal and then goes straight to it. In the second video, we can see the behavior of the robot, equipped with the same policy as before, once a damage in the goal compass (a constant offset, see Section 6.3 for further details) is simulated in the task. In the video we can see that the robot easily moves toward the goal but, in proximity of it, it performs a loop around the goal before reaching it. In the third video we observe the behavior of the robot at the 8^{th} iteration of the DPO algorithm. The performance sensibly improves but the behavior is still similar to the previous one. In the fourth video, we observe the behavior of the robot equipped with the policy learned after 100 iterations of DPO. The robot is now able to reach the goal without performing any loop around it, as he was doing before the compass damage was introduced in the task.

2 https://www.youtube.com/playlist?list=PLT0QhH0oHUgVWxikBgc9be78LCNgyqlWO

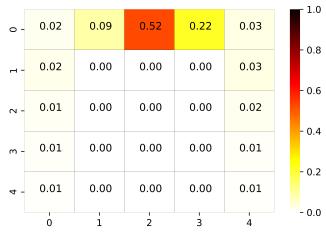

Figure 6.4: Robot Adaptation: abstract state visitation frequencies at the initial iteration of DPO.

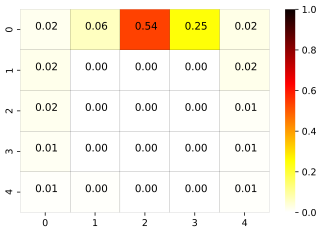

Figure 6.5: Robot Adaptation: abstract state visitation frequencies at the fifth iteration of DPO.

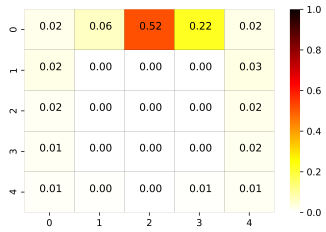

Figure 6.6: Robot Adaptation: abstract state visitation frequencies at the eighth iteration of DPO.

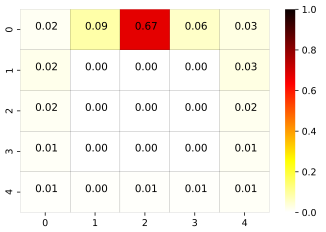

Figure 6.7: Robot Adaptation: abstract state visitation frequencies at the 99-th (final) iteration of DPO.

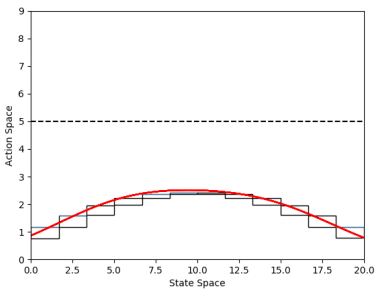

Figure 6.8: Minigolf: initial policy for DPO.

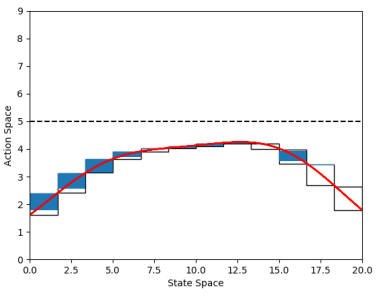

Figure 6.9: Minigolf: policy after 33 iterations of DPO.

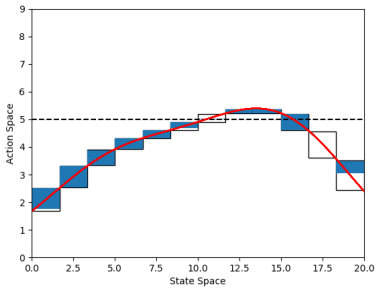

Figure 6.10: Minigolf: policy after 66 iterations of DPO.

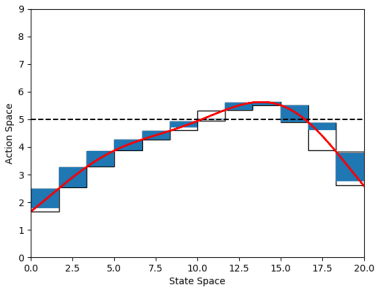

Figure 6.11: Minigolf: policy after 99 iterations of DPO.

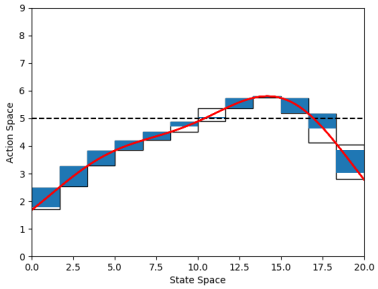

Figure 6.12: Minigolf: policy after 199 iterations of DPO.

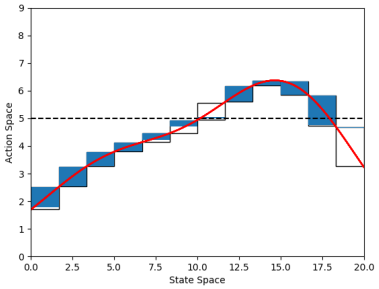

Figure 6.13: Minigolf: policy after 299 iterations of DPO.

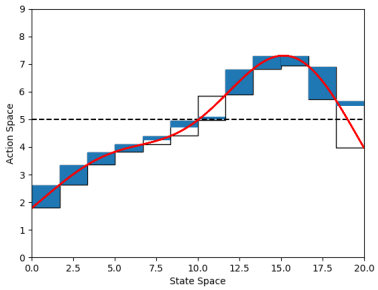

Figure 6.14: Minigolf: policy after 399 iterations of DPO.

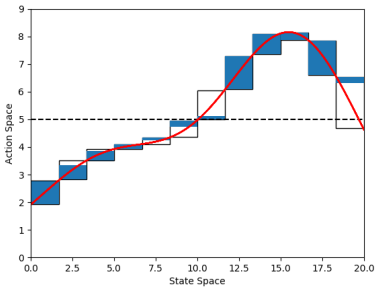

Figure 6.15: Minigolf: final policy after 499 iterations of DPO.

7

CONCLUSIONS

We proposed a policy optimization algorithm (DPO) that is completely deterministic for the whole learning process and allows to learn a satisfactory deterministic policy in tasks where the environment meets some Lipschitz properties of regularity. We tested the validity of the proposed approach on simulated control tasks and we provided the results of the performed experiments in Chapter 6. In Chapter 5, we also provided theoretical support to this approach under these regularity assumptions on the environment, which we deem realistic for continuous control problems of practical interest [Kober, Bagnell, and Peters, 2013]. Indeed, in many continuous problems from industrial robotics, for instance, the effects of actions vary smoothly with the observed states and with the actions themselves. Driven by the goal of finding a learning approach able to ensure safe exploration, we proposed the construction of an abstract MDP (δ-MDP) where the exploration usually obtained with random actions is replaced with a passive exploration, ensured by the regularity assumptions on the environment. The support provided by the theory on δ-MDP combined with the advantages provided by PS methods, such as the robustness to noise or the possibility of encoding domain knowledge in the policy definition, motivated us to follow the strategy presented in this work.

We have shown, empirically, that DPO is competitive with policy gradient methods, at least in the fine-tuning of parametric controllers. We think that the latter experiment, that we discussed in Section 6.3, is an important example for the deploying of lifelong-learning agents in the real world, where changes in the environment and in the goal can occur. This kind of setting may represent a natural application for RL in the near future. In this setting, DPO can remove the unnecessary source of risk deriving from random actions. As a consequence, several problems related to the safety of the agent's hardware and of the environment are avoided. Random exploration may still play an indispensable role in learning challenging tasks *from scratch*, which should be done in a controlled environment. Indeed, the limitation of the feasible actions for each abstract state of the δ-MDP prevents, in many cases, to evaluate the optimal actions and an excessive number of iterations may be necessary before that these actions can be evaluated.

Being the first algorithm of his kind, DPO leaves room for sev-

eral improvements that we present here as open questions and future research directions of this work. First of all, state aggregation could be performed in a more informed way, in order to guarantee a more *efficient exploration*. In this work, we analyzed some theoretical aspects of state abstraction (Section 2.5) and we decided to consider only partitions of the state space in which all the subsets have the same size. However, we captured an important trade-off between precision and exploration in the algorithm that can be addressed by selecting a finer or coarser aggregation of states (Figure 6.2). Starting from this trade-off, new solutions for the state aggregation can be evaluated: partitions of the state space with finer subsets covering the most visited regions and coarser subsets covering the less visited ones, adaptive discretization through algorithm's iterations or soft aggregation [Singh, Jaakkola, and Jordan, 1995]. Moreover, we could make a more efficient use of the collected data, by re-using samples collected from previous policies once they are put in relationship with the new policy. Establishing a relationship between consecutive δ-MDPs may also prove necessary to provide convergence guarantees for DPO. Since DPO only removes the risk deriving from action stochasticity, a further extension of this work that would be of great practical relevance is the combination of DPO with other safe RL approaches. Removing the algorithm's hyperparameters is also important, since a deployed agent cannot perform grid-search. Finally, when available, prior knowledge on the controlled system should be integrated with collected samples in order to better estimate the abstract model of the δ-MDP.

BIBLIOGRAPHY

Abbeel, Pieter, Adam Coates, and Andrew Ng (Nov. 2010). "Autonomous Helicopter Aerobatics through Apprenticeship Learning." In: *I. J. Robotic Res.* 29, pp. 1608–1639 (cit. on p. 30).

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (June 2016). "Concrete Problems in AI Safety." In: (cit. on p. 31).

Baek, Donghoon, Minho Hwang, Hansoul Kim, and Dong-Soo Kwon (June 2018). "Path Planning for Automation of Surgery Robot based on Probabilistic Roadmap and Reinforcement Learning." In: pp. 342–347 (cit. on p. 1).

Clouse, J. (1997). *On Integrating Apprentice Learning and Reinforcement Learning TITLE2:* tech. rep. USA (cit. on p. 30).

Dean, Thomas, Robert Givan, and Sonia Leach (Feb. 2013). "Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes." In: *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)* (cit. on p. 22).

Deisenroth, Marc, Gerhard Neumann, and Jan Peters (Aug. 2013). *A Survey on Policy Search for Robotics*. Vol. 2 (cit. on p. 13).

D'Oro, Pierluca, Alberto Maria Metelli, Andrea Tirinzoni, Matteo Papini, and Marcello Restelli (2019). "Gradient-Aware Model-based Policy Search." In: *CoRR* abs/1909.04115 (cit. on p. 57).

Driessens, Kurt and Sašo Džeroski (Dec. 2004). "Integrating Guidance into Relational Reinforcement Learning." In: *Machine Learning* 57, pp. 271–304 (cit. on p. 30).

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (2016). "Benchmarking Deep Reinforcement Learning for Continuous Control." In: *ICML*. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1329–1338 (cit. on p. 14).

Ferns, Norman, Prakash Panangaden, and Doina Precup (July 2012). "Metrics for Finite Markov Decision Processes." In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence* (cit. on pp. 20, 37).

García, J. and F. Fernández (Aug. 2015). "A comprehensive survey on safe reinforcement learning." In: 16, pp. 1437–1480 (cit. on pp. 2, 29).

Gehring, Clement and Doina Precup (May 2013). "Smart exploration in reinforcement learning using absolute temporal difference errors." In: vol. 2, pp. 1037–1044 (cit. on p. 31).

Geibel, Peter and Fritz Wysotzki (2005). "Risk-Sensitive Reinforcement Learning Applied to Control under Constraints." In: *ArXiv* abs/1109.2147 (cit. on p. 30).

Givan, Robert, Thomas L. Dean, and Matthew Greig (2003). "Equivalence notions and model minimization in Markov decision processes." In: *Artif. Intell.* 147.1-2, pp. 163–223 (cit. on p. 37).

Givan, Robert, Sonia M. Leach, and Thomas L. Dean (2000). "Bounded-parameter Markov decision processes." In: *Artif. Intell.* 122.1-2, pp. 71–109 (cit. on pp. 15–17, 42, 46).

Heger, Matthias (1994). "Consideration of Risk in Reinforcement Learning." In: *ICML* (cit. on p. 29).

Kakade, Sham M. and John Langford (2002). "Approximately Optimal Approximate Reinforcement Learning." In: *ICML*. Morgan Kaufmann, pp. 267–274 (cit. on p. 52).

Kober, Jens, J. Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey." In: *I. J. Robotics Res.* 32.11, pp. 1238–1274 (cit. on p. 71).

Li, Lihong, Thomas J. Walsh, and Michael L. Littman (2006). "Towards a Unified Theory of State Abstraction for MDPs." In: *ISAIM* (cit. on pp. 19, 21, 22, 38).

Li, Yuxi (Aug. 2019). *Reinforcement Learning Applications* (cit. on p. 1).

Moldovan, Teodor and Pieter Abbeel (May 2012). "Safe Exploration in Markov Decision Processes." In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 2 (cit. on p. 30).

Papini, Matteo, Matteo Pirotta, and Marcello Restelli (May 2019). *Smoothing Policies and Safe Policy Gradients* (cit. on p. 32).

Peters, Jan (2002). *Policy gradient methods for control applications*. Tech. rep. (cit. on p. 59).

– (Jan. 2010). "Policy gradient methods." In: *Scholarpedia* 5, p. 3698 (cit. on p. 23).

Peters, Jan and Stefan Schaal (2008). "Reinforcement learning of motor skills with policy gradients." In: *Neural networks : the official journal of the International Neural Network Society* 21 4, pp. 682–97 (cit. on pp. 13, 26).

Pirotta, Matteo, Marcello Restelli, and Luca Bascetta (2015). "Policy gradient in Lipschitz Markov Decision Processes." In: *Machine Learning* 100.2-3, pp. 255–283 (cit. on pp. 17–19).

Puterman, Martin L (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons (cit. on p. 6).

Rachelson, Emmanuel and Michail G. Lagoudakis (2010). "On the locality of action domination in sequential decision making." In: *ISAIM* (cit. on p. 19).

Ray, Alex, Joshua Achiam, and Dario Amodei (2019). "Benchmarking Safe Exploration in Deep Reinforcement Learning." In: (cit. on pp. 61, 62).

Recht, Benjamin (2018). "A Tour of Reinforcement Learning: The View from Continuous Control." In: *CoRR* abs/1806.09460 (cit. on p. 59).

Sehnke, Frank, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber (2008). "Policy Gradients with Parameter-Based Exploration for Control." In: *ICANN (1)*. Vol. 5163. Lecture Notes in Computer Science. Springer, pp. 387–396 (cit. on pp. 2, 26, 60, 64).

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (June 2014). "Deterministic Policy Gradient Algorithms." In: *31st International Conference on Machine Learning, ICML 2014* 1 (cit. on pp. 2, 27, 28).

Singh, Satinder P., Tommi Jaakkola, and Michael I. Jordan (1995). "Reinforcement Learning with Soft State Aggregation." In: *Advances in Neural Information Processing Systems 7*. Ed. by G. Tesauro, D. S. Touretzky, and T. K. Leen. MIT Press, pp. 361–368 (cit. on p. 72).

Stulp, Freek and Olivier Sigaud (Oct. 2012). "Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning." In: (cit. on p. 27).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on pp. 1, 5, 11, 25).

Sutton, Richard S., David A. McAllester, Satinder P. Singh, and Yishay Mansour (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *NIPS* (cit. on pp. 13, 24).

Thomas, Philip, Bruno da Silva, Andrew Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill (Nov. 2019). "Preventing undesirable behavior of intelligent machines." In: *Science (New York, N.Y.)* 366, pp. 999–1004 (cit. on p. 31).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control." In: *IROS*. IEEE, pp. 5026–5033 (cit. on pp. 61, 62).

Turchetta, Matteo, Felix Berkenkamp, and Andreas Krause (2016). "Safe Exploration in Finite Markov Decision Processes with Gaussian Processes." In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 4312–4320. ISBN: 9781510838819 (cit. on p. 32).

Williams, Ronald J. (1992a). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: *Machine Learning* 8, pp. 229–256 (cit. on pp. 23, 64).

– (1992b). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: *Machine Learning* 8, pp. 229–256 (cit. on p. 25).

Zhang, Chuheng, Yuanqi Li, and Jian Li (2019). "Policy Search by Target Distribution Learning for Continuous Control." In: *CoRR* abs/1905.11041. arXiv: 1905.11041 (cit. on p. 33).

Zhao, Tingting, Hirotaka Hachiya, Voot Tangkaratt, Jun Morimoto, and Masashi Sugiyama (Mar. 2013). "Efficient Sample Reuse in Policy Gradients with Parameter-Based Exploration." In: *Neural computation* 25 (cit. on p. 27).

APPENDIX

A.1 MAXIMUM LIKELIHOOD PROBLEM IN DOUBLE INTEGRATOR

In this section we present the maximum likelihood problem considered in the *Double Integrator* task to estimate the abstract transition function of the δ-MDP. Starting from the convex program defined in subsection 4.4.3 and motivated by the nature of noise in the task, described in Section 6.2, we solve the following problem:

$$
\begin{aligned}
\min_{\widetilde{P}\in\mathbb{R}^{|\mathcal{X}|\times|\mathcal{A}|\times|\mathcal{X}|}} \quad & -\sum_{X,X'\in\mathcal{X}_{\mathcal{D}},a\in\mathcal{A}_X} \log \widetilde{P}(X'|X,a) && \\
\text{subject to} \quad & \widetilde{P}(X'|X,a) \geqslant 0 && \forall X,X'\in\mathcal{X}_{\mathcal{D}}, a\in\mathcal{A}_X \\
& \sum_{X'\in\mathcal{X}_{\mathcal{D}}} \widetilde{P}(X'|X,a) = 1 && \forall X\in\mathcal{X}_{\mathcal{D}}, a\in\mathcal{A}_X \\
& \left|\sum_{j\in J} \widetilde{P}(X'_{ij}|X,a) - \sum_{j\in J} \widetilde{P}(X'_{ij}|X,\widetilde{a})\right| && \\
& \qquad \leqslant L_{\widetilde{P}_i}|a-\widetilde{a}| && \forall i\in I, X'\in\mathcal{X}_{\mathcal{D}}, a\in\mathcal{A}_X \qquad \text{(A.1)} \\
& \left|\sum_{i\in I} \widetilde{P}(X'_{ij}|X,a) - \sum_{i\in I} \widetilde{P}(X'_{ij}|X,\widetilde{a})\right| && \\
& \qquad \leqslant L_{\widetilde{P}_j}|a-\widetilde{a}| && \forall j\in J, X'\in\mathcal{X}_{\mathcal{D}}, a\in\mathcal{A}_X. \qquad \text{(A.2)}
\end{aligned}
$$

In the Double Integrator task, the state is two-dimensional and the state abstraction is performed by considering an $I\times J$ grid. We represent the single dimensions with the indexes $i\in I$ and $j\in J$, $X_{ij}\in\mathcal{X}$ is the abstract state corresponding to the combination of i-th (w.r.t. I) and j-th (w.r.t. J) dimensions. The action is one-dimensional, hence we consider the absolute value $|a-\widetilde{a}|$ as measure of the action distance. In the problem we can use two (possibly) different Lipschitz constants $L_{\widetilde{P}_i}$ and $L_{\widetilde{P}_j}$, one for each dimension.

The Lipschitz constant $L_{\widetilde{P}_i}$ for a generic dimension can be derived starting from the Assumption 3. We use the *Pinsker's inequality* to bound the TV distance:

$$
\mathrm{TV}\left(P(\cdot|s_i,a), P(\cdot|s_i,\widetilde{a})\right) \leqslant \sqrt{\frac{1}{2} D_{KL}\left(P(\cdot|s_i,a), P(\cdot|s_i,\widetilde{a})\right)}, \qquad \text{(A.3)}
$$

where D_{KL} is the *Kullback–Leibler divergence*. In the case of two Gaussian distributions $\mathcal{N}_1$ and $\mathcal{N}_2$ with the same standard deviation σ and different means μ_1 and μ_2, we have:

$$\begin{aligned} D_{KL}(\mathcal{N}_1, \mathcal{N}_2) &= \frac{\sigma^2 + (\mu_1 - \mu_2)^2}{2\sigma^2} - \frac{1}{2} \\ &= \frac{(\mu_1 - \mu_2)^2}{2\sigma^2}. \end{aligned} \tag{A.4}$$

In the Double Integrator task, a Gaussian noise $\mathcal{N}(0, \sigma_i)$ is added to each dimension i of the arriving state. If we consider each dimension separately, as we did in the maximum likelihood problem, $P(\cdot|s_i, a)$ is a Gaussian distribution $\mathcal{N}(s'_i, \sigma_i)$, where s'_i is the i-th dimension of the arriving state in absence of noise and σ_i is the standard deviation of the Gaussian zero-mean noise added to s'_i. In the Double Integrator task, $s' = As + Ba$. Hence, going back to (A.3) we write:

$$TV(P(\cdot|s_i, a), P(\cdot|s_i, \widetilde{a})) \leqslant \left| \frac{B_i(a - \widetilde{a})}{2\sigma_i} \right|, \tag{A.5}$$

where $\left|\frac{B_i}{2\sigma_i}\right| = L_{TV}$, with B_i the i-th row of matrix B (B_i is a scalar in this task). Since $L_{\widetilde{P}_i} = 2L_{TV}$, the Lipschitz constant of the problem, for the i-th dimension, is $L_{\widetilde{P}_i} = \left|\frac{B_i}{\sigma_i}\right|$.