POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Meta Learning the Step Size in Policy Gradient Methods

Supervisor: Prof. Marcello Restelli

Co-supervisor: Eng. Luca Sabbioni

Candidate:
Francesco Corda, ID 920212

Academic Year 2020-2021

Ai miei genitori.

Abstract in English

Over the last years, Reinforcement Learning (RL) research has achieved astonishing results in many areas, ranging from robotics and autonomous driving to complex games such as Go and Poker. In an RL task, an agent interacts with an environment by observing a representation of its state, performing an action, and receiving a numeric reward in return. The goal is to identify a strategy, also known as policy, that maximizes the cumulative reward obtained over a certain time horizon.

Among the various RL algorithms, this thesis focuses on Policy Gradient methods, which optimize the policy by means of iterative Gradient Ascent (GA) updates. These approaches, while praised for their convergence properties and strong theoretical groundings, require precise and problem-specific hyperparameter tuning to achieve good performance. As a consequence, they tend to struggle when asked to accomplish a series of heterogeneous tasks.

To solve these issues, this thesis adopts a Meta Reinforcement Learning (Meta-RL) approach. Meta-RL aims to create models that can learn quickly and adapt to unseen RL settings. In our work, we introduce a formulation to solve Meta-RL tasks, known as meta-MDP, and we propose an algorithm to solve meta-MDPs with PG learners. In these cases, the "meta" action reduces to the choice of the step size of each GA iteration. The idea of the approach is to apply a batch mode, value-based algorithm, known as Fitted Q Iteration (FQI), to derive an estimate of the expected model improvements and to dynamically recommend the most adequate step size in the current scenario. We conclude our work by evaluating the approach in different settings and reflecting on open questions and future improvements.

Keywords: Reinforcement Learning, Meta Learning, Value Based, Policy Gradient, Adaptive Step Size, Fitted Q Iteration.

Estratto in Italiano

Nel corso degli ultimi anni, la ricerca nel campo dell'Apprendimento per Rinforzo, Reinforcement Learning (RL) in inglese, ha raggiunto risultati straordinari in diversi campi, spaziando dai progressi nella Robotica e nella Guida Autonoma fino alla risoluzione di giochi complessi come il Go e il Poker.

In uno scenario RL, un agente interagisce con un ambiente esterno osservandone una rappresentazione dello stato, compiendo un'azione e ricevendo in cambio un premio numerico. Il ripetersi di tale interazione dà origine ad un Problema Sequenziale di Decisione, il cui obiettivo è l'identificazione di una strategia, policy in inglese, che massimizzi il valore cumulato dei premi ottenuti lungo un certo orizzonte temporale. Nella teoria RL, il problema appena descritto è formalizzato in un modello denominato Processo Decisionale di Markov, Markov Decision Process (MDP) in inglese, all'interno del quale operano gli algoritmi di apprendimento.

Tra le varie tipologie di algoritmi RL, questa tesi è dedicata ai metodi Policy Gradient (PG), che ottimizzano la policy attraverso una serie di step di Ascesa del Gradiente, Gradient Ascent (GA) in inglese. Questi approcci, pur essendo elogiati per le loro proprietà di convergenza e solide basi teoriche, raggiungono delle performance adeguate solo attraverso una precisa configurazione dei loro iperparametri che varia in base al problema. Come conseguenza, i risultati ottenuti da questi metodi tendono a deteriorare se gli stessi sono utilizzati per compiere una sequenza eterogenea di attività.

Per risolvere i limiti appena menzionati, il nostro lavoro adotta un approccio di Meta Apprendimento per Rinforzo, Meta Reinforcement Learning (meta-RL) in inglese. Un algoritmo meta-RL ha come obiettivo la creazione di modelli che possano imparare nuove abilità velocemente e adattarsi a scenari RL mai visti, utilizzando un numero ridotto di esempi di apprendimento. Come si può dedurre dal nome, questi approcci agiscono ad un livello di astrazione superiore rispetto ai classici algoritmi di apprendimento, mirando a ottimizzare non il ritorno cumulato su una singola attività, ma il processo stesso di apprendimento, ovvero il problema di "imparare ad imparare" una nuova abilità.

La tesi si sviluppa nel modo seguente. Il Capitolo 2 è dedicato all'introduzione dei prerequisiti necessari alla comprensione dei contributi presentati. Inizialmente, si definiscono i Problemi di Decisione Sequenziali e la loro formalizzazione nel modello MDP.

In seguito, si introducono gli algoritmi RL, presentando una loro classificazione generale e spiegando i punti di forza e le debolezze dei vari approcci. Durante la trattazione, un particolare riguardo è rivolto agli algoritmi di tipo Value Based e Policy Gradient, poiché la nostra soluzione meta-RL utilizza un algoritmo della prima classe per migliorare le performance degli algoritmi della seconda.

Nel Capitolo 3, si descrive il problema del Meta Apprendimento, spiegandone i concetti generali, le motivazioni e gli obiettivi nel contesto dell'Apprendimento Supervisionato, di più facile comprensione per un primo approccio all'argomento. Vengono illustrati i principali metodi di risoluzione del problema, tra cui gli algoritmi Meta Learning basati sull'ottimizzazione, a cui appartiene anche il nostro approccio. Questi algoritmi mirano ad identificare dinamicamente il set di parametri che meglio si adatti a scenari inediti, in seguito ad un addestramento compiuto su attività paragonabili. Il capitolo prosegue con una descrizione dettagliata del problema meta-RL, che consiste nell'applicazione di questi approcci al dominio RL. In conclusione, si presentano alcuni esempi di algoritmi meta-RL "stato dell'arte" che sono stati di ispirazione per il nostro lavoro.

Nel Capitolo 4 si presentano i contributi teorici e metodologici della tesi. Si inizia descrivendo il concetto di Processo Decisionale di Markov Contestuale, Contextual Markov Decision Process (CMDP) in inglese, un modello che consente di racchiudere un insieme di MDP determinato da un insieme di parametri comuni, detto contesto, in un'unica entità. Questo modello, per quanto adatto a rappresentare un gruppo eterogeneo di MDP, non è abbastanza espressivo per risolvere il problema meta-RL. Per superare questo limite, si introduce un nuovo modello, chiamato meta-MDP, che contiene un CMDP oltre a degli elementi utili all'ottimizzazione meta-RL. In particolare, un meta-MDP è caratterizzato da una meta funzione di costo, che valuta le variazioni di apprendimento di un algoritmo in un insieme di attività, da un meta stato, che può essere visto come una generalizzazione dello spazio di osservazione di un normale MDP, da un algoritmo di ottimizzazione e, per finire, dalle meta azioni, ovvero l'insieme di iperparametri che regolano l'algoritmo di ottimizzazione. In seguito ad una spiegazione dettagliata del modello, analizziamo il caso dei Lipschitz meta-MDP, in cui l'insieme di MDP rispetta la proprietà di continuità di Lipschitz. Sulla base di questa assunzione, siamo in grado di ricavare delle garanzie sul valore atteso del modello e sul suo gradiente.

Concludiamo il capitolo con l'introduzione di un algoritmo per la risoluzione dei meta-MDP in cui il metodo di ottimizzazione è di tipo PG. In questi casi, la "meta" azione si riduce al solo passo di apprendimento, step size in inglese, di un'iterazione di GA. L'idea dell'approccio è quella di applicare un algoritmo di tipo batch e value-based, chiamato FQI, che sfrutta la capacità di generalizzazione della regressione per ricavare una approssimazione dei miglioramenti attesi del modello. La stima, dipendente dalle attuali meta caratteristiche e dalla meta azione, è in seguito utilizzata per raccomandare dinamicamente la step size che più si adatta allo scenario corrente.

In seguito alla descrizione delle fasi dell'algoritmo, il Capitolo 5 è dedicato ad una valutazione empirica delle sue performance. Si eseguono una serie di esperimenti in diversi ambienti RL simulati, sviluppati in compatibilità con la libreria OpenAI Gym (Brockman et al., 2016). Si valutano la velocità di convergenza e la stabilità dei risultati ottenuti, e si confrontano le performance con diverse configurazioni di un algoritmo PG, mostrando che l'approccio può rappresentare un vantaggio rispetto ai metodi classici. Infine, si valutano i fattori di maggior impatto sui risultati, tra cui le diverse implementazioni del meta stato proposte nel capitolo precedente.

Il Capitolo 6 conclude la tesi, riflettendo sugli obiettivi raggiunti e sui quesiti incontrati nel corso delle ricerche. In particolare, si discutono possibili miglioramenti all'algoritmo FQI e alcune variazioni da apportare al modello meta-MDP per renderlo maggiormente scalabile a problemi complessi.

Parole Chiave: Apprendimento per Rinforzo, Meta Apprendimento, Value-Based, Policy Gradient, Passo di Apprendimento Adattivo, Fitted Q Iteration.

Ringraziamenti

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivations and Goals

Artificial Intelligence is, without any doubt, one of the most vibrant research areas of our time, with astonishing progress being made over the last few decades. We saw AI algorithms solve problems once thought intractable. Among others, DeepMind's AlphaZero (Silver et al., 2017) reached super-human capabilities in complex deterministic games such as Chess and Go. Similar results were obtained in non-deterministic games, with Facebook's Pluribus (Brown and Sandholm, 2019) beating 15 professional players in a match of Texas Hold'em Poker. Other excellent examples are the advances in the fields of computer vision and autonomous driving, or the solution of the protein folding problem by DeepMind's AlphaFold (Senior et al., 2020).

Many other examples can testify the achievements obtained and it is reasonable to think that the trend of improvement will continue in the future. AI will take on increasingly challenging tasks and master them to perfection. A more difficult aspect to assess is whether these algorithms can be considered steps towards the creation of "general" intelligent agents, in addition to excellent executors in a single domain.

If we look at the qualities that make humans intelligent, we could cite, among others: the versatility in different scenarios, the ability to quickly learn new tasks from very few examples and the capability of re-applying lessons learned from past experiences to solve new problems.

Unfortunately, most AI algorithms still struggle in these aspects and, when asked to accomplish a sequence of (even reasonably simple) heterogeneous tasks, they tend to fail. In addition, machine learning generally requires a large amount of time and training data to reach good performances, coupled with a modelling tailored for the specific case.

This "brute-force" approach to AI is not affordable if we want our agent to learn different skills and adapt to various settings in a reasonable amount of time. What

we need for our agents is to *learn how to learn* new tasks faster by reusing previous experience, instead of keeping each new task in isolation. Meta-learning, also known as learning to learn, is a set of techniques to produce models that are fast and versatile learners. It can be adopted in many areas of machine learning, including Reinforcement Learning (RL), the subject of analysis of this thesis.

After an introduction to the background knowledge necessary to understand the topic, we present the problem of Meta Reinforcement Learning (Meta-RL) and propose a group of techniques to solve it. In our analysis, we consider various heterogeneous tasks and a RL learner, configured by a set of hyperparameters, that optimizes the performances over their distribution.

The main question of our work is whether is possible to dynamically select an optimal set of hyperparameters such that the learner can rapidly adapt to unseen scenarios and reach its goal in fewer steps compared to pre-configured approaches, that tend to struggle in these settings. To answer this, we propose a solution that exploits the generalization capabilities of regression algorithms to predict these hyperparameters from a dataset of meta-training samples.

In particular, our efforts are focused on optimizing Policy Gradient (PG) learners, that base their update on the Gradient Ascent (GA) step. Under this constraint, the set of hyperparameters reduces to one element, the step size α.

1.2 Contributions

The contributions of this work are theoretical, algorithmic and experimental. To start, we propose a formalization of the Meta-RL problem, known as meta-MDP. This general framework allows to solve a set of RL tasks, grouped as a contextual Markov Decision Process. We discuss the main elements of the model, i.e. the meta objective function $\mathcal{L}$, the meta action h and the meta state x, of which we present various implementations, providing motivations for all our choices. Then, we consider Lipschitz meta-MDPs, the instances of these models in which the tasks are Lipschitz continuous, and we derive some gurantees on the expected return and on its gradient under this condition.

Subsequently, we propose an algorithm to learn the step size α of PG methods in a meta-MDP. The idea of the approach is to apply a batch mode, value based algorithm, known as FQI, to derive an estimate of the action-value function Q, on the basis of the meta features x and of the meta action h. This approximation is used to dynamically recommend the most adequate step size in the current scenario.

In conclusion, we evaluate our approach in various simulated environments, highlighting its strengths and current limitations. We compare our results against different configurations of the classic PG algorithm. We discuss the features that contribute to the outcome and we analyze the impact of different implementations of the meta state.

1.3 Thesis Structure

This thesis is structured as follows.

In Chapter 2, we provide a general description of the preliminary knowledge necessary to understand this work. The first sections are dedicated to the introduction of Sequential Decision Making Problems and Dynamic Programming approaches. Subsequently, we provide a classification of various Reinforcement Learning methods, with a particular focus on Value Based and Policy Gradient algorithms, that are adopted in our work.

In Chapter 3, we present the field of Meta Learning in the context of Supervised Learning, explaining the motivations, the goals and various solution techniques. Then, we focus on the problem of Meta Reinforcement Learning, describing some state of the art meta-RL approaches (e.g. RL^2 and MAML). We conclude the chapter by introducing the concept of Domain Randomization to generate a distribution of RL tasks.

In Chapter 4, we start by giving the definition of Meta Markov Decision Process (meta-MDP), the framework upon which our solution is based. Then, we analyze the case of Lipschitz meta-MDPs, deriving some general boundaries on the performance under the Lipschitz Continuity property. We conclude the chapter by presenting our solution, describing its properties and detailed functioning.

In Chapter 5, we show the experimental results obtained by the algorithm in various simulated meta tasks, developed in compliance with the OpenAI Gym library. For each experiment, we provide results of our approach evaluated in different settings and we compare our performance with baselines of pre-configured Policy Gradient algorithms.

In Chapter 6, we draw the conclusions about the achievements of this thesis. We analyze the results obtained and describe the limitations of our method. We conclude our work with a reflection about further improvements and open questions.

Chapter 2

Background

This chapter is dedicated to the introduction of the background knowledge needed to understand the contributions of this thesis. We begin our description by presenting the concept of Sequential Decision Making Problem, formalized under the scheme of a Markov Decision Process (MDP), and its solution with Dynamic Programming methods. Subsequently, we present the set of techniques known as Reinforcement Learning (RL), detailing its motivations and main properties. The chapter concludes with a classification of various RL algorithms, with special attention to the ones adopted during our thesis work. For a further explanation of these contents, we suggest the excellent introductions to the topic by (Sutton and Barto, 2018) and (Szepesvari, 2010), from which this chapter is highly inspired.

2.1 Sequential Decision Making Problems

One of the great AI challenges is to replicate the human mind capability to think strategically and learn from experiences gathered from the external world. Many real problem applications require a sequence of actions in order to obtain a result. We can imagine all these scenarios as *Sequential Decision Making Problems*, in which an agent repeatedly interacts with an environment in order to achieve a certain goal.

The agent has the ability to act on an environment, which changes according to a transition probability and produces rewards in response to actions. The RL paradigm aims to build an intelligent agent that can learn from its past actions and results, hence improving the performance. This very general setting is called the *Agent-Environment Interface*, and can be used to describe many real world problems. It also poses various fascinating challenges, because it requires the agent to see long-term consequences of its action and to renounce immediate rewards in favour of greater gains in the long run. This thesis is aimed at describing and trying to answer some of these challenges.

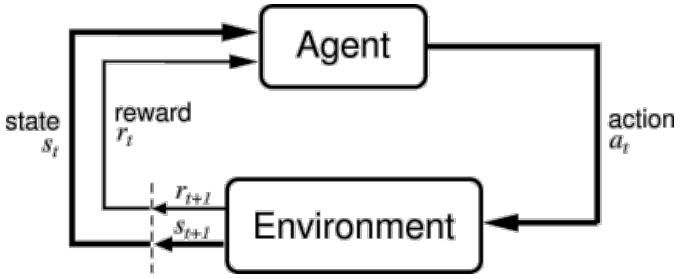


Figure 2.1: Agent-Environment Interface.
Source: Richard S. Sutton.

2.1.1 Markov Decision Processes

Definition 2.1.1 (Markov Decision Process). *Markov Decision Processes (MDPs) are a formalization of the problem of learning solely through interactions, without any external supervision. They are usually formalized as a tuple* $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$*, where:*

- $\mathcal{S}$ *is a measurable state space that can be assumed by the environment;*
- $\mathcal{A}$ *is a measurable action space available to the agent;*
- P *is the state transition probability kernel, it's a function* $P : \mathcal{S} \times \mathcal{A} \times \mathcal{A} \to [0, 1]$ *that assigns a probability* $P(s' \mid s, a) = P(s_t = s' \mid s_{t-1} = s, a_{t-1} = a)$*;*
- R *is a reward function, it computes the expected a reward based on the current state and the action chosen by the agent:* $R(s, a) = \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a]$*;*
- γ *is a discount factor,* $\gamma \in [0, 1]$*;*
- μ *is the initial state distribution.*

The model described above is the one of a finite, discrete-time MDP, but it can be adapted to study more complex cases such as continuous state spaces and action spaces or continuous time.

In this model, the agent and the environment interact at each discrete time instant. At each step, the agent captures a representation of the environment state and decides on an action. In response, the environment produces a reward and modifies its state according to the transition P. This interaction goes on for a number of time steps. The sequence of states, actions and rewards produced during the interactions when grouped together forms a trajectory $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$ that constitutes the basis used by the agent to learn.

At last, we want to highlight the property that gives this model its name: the Markov Property (or assumption). This property states the following:

$$P(x_{t+1} = j \mid x_t = i, x_{t-1} = k_{t_1}, \ldots, x_1 = k_1, x_0 = k_0) = P(x_{t+1} = j \mid x_t = i). \quad (2.1)$$

The property enters in play in the transition kernel P and affirms that the state captures all the information about the history of the environment until the current moment.

The goal of the agent is defined in terms of a reward, which is a real number received at each time step. The agent aims to maximize the total reward received during the interaction with the environment. To express this concept in a more precise way, we define the return G_t as the discounted sum of the rewards collected along the trajectory:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.2}$$

The discount factor serves two purposes: it can be interpreted as the probability that the process will go on after n steps, or it can be used to favor immediate rewards with respect to delayed rewards. Values of γ close to 0 will lead to myopic evaluations, while values close to 1 will lead to far-sighted evaluation.

2.1.2 Policy

The policy defines the behavior of the agent. It decides, at any given point in time, which action the agent selects.

Definition 2.1.2 (Policy). *A policy $\pi = (\pi_t)_{t \in \mathbb{N}}$ is a mapping from the state space $\mathcal{S}$ to a probability distribution over the action space $\mathcal{A}$:*

$$\pi_t(a \mid s) = P\left[a_t = a \mid s_t = s\right]. \tag{2.3}$$

If for each $t \in \mathbb{N}$ and for each $s \in \mathcal{S}$, the policy π_t assigns a value of one to a single action in $\mathcal{A}$, π is called deterministic. If π_t does not depend on t, π is called stationary.

Definition 2.1.3 (State Occupancy). *The policy π can be used to define the (discounted) state occupancy measure $\delta_{\mu,\pi}$, as follows:*

$$\delta_{\mu,\pi} = (1-\gamma) \sum_{s \in \mathcal{S}} \mu(s_0) \sum_{t=0}^{\infty} \gamma^t P\left[s_t = s \mid s_0\right]. \tag{2.4}$$

2.1.3 Value Functions

Given a certain policy π, it's possible to define some functions, called value functions, that express the expected utility of applying π over the state space.

Definition 2.1.4 (Action-Value Function). *The action-value function, denoted as q_π, is the expected return of selecting action a in state s and then following policy π:*

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim P(\cdot \mid s_t, a_t) \\ a_{t+1} \sim \pi(\cdot \mid s_{t+1})}} \Big[G_t \mid s_t = s, a_t = a \Big]; \tag{2.5}$$

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

In a similar manner, we can define the state-value function as follows:

Definition 2.1.5 (State-Value Function). *The state-value function of a state s under a policy π, denoted as $V_\pi(s)$, is the expected return obtained starting from s and following π afterwards. Formally:*

$$V_\pi(s) = \mathop{\mathbb{E}}_{\substack{s_{t+1}\sim P(\cdot|s_t,a_t)\\ a_t\sim\pi(\cdot|s_t)}}\Big[G_t \mid s_t = s\Big]; \tag{2.6}$$

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

At last, the expected return of a policy π in a MDP is defined below.

Definition 2.1.6 (Expected Return). *The expected return J_π can be defined using two distinct formulations, one based on the transition probabilities and the other based on the state occupancy $\delta_{\mu,\pi}$:*

$$J_\pi = \mathop{\mathbb{E}}_{\substack{s_0\sim\mu\\ s_{t+1}\sim P(\cdot|s_t,a_t)\\ a_t\sim\pi(\cdot|s_t)}}\Big[G_t\Big] = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim\delta_{\mu,\pi}\\ a\sim\pi(\cdot|s)}}\Big[R(s,a)\Big]. \tag{2.7}$$

2.1.4 Bellman Operators

The state-value function can be decomposed into an immediate reward and a discounted value of the next state:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s] \\ &= \sum_{a\in\mathcal{A}} \pi(a \mid s)\left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a)V_\pi(s')\right).^{1} \end{aligned} \tag{2.8}$$

Also the action-value function can be decomposed:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a] \\ &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a)V_\pi(s') \\ &= R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) \sum_{a'\in\mathcal{A}} \pi(a' \mid s')Q_\pi(s',a'). \end{aligned} \tag{2.9}$$

These are also called *Bellman Equations*, and they define the value function in a recursive way. They can be expressed in a matrix form:

$$V_\pi = R_\pi + \gamma P_\pi V_\pi. \tag{2.10}$$

We now introduce the Bellman Operators, as stated below.

[1] In case of continuous state-action spaces, the summations become integrals, i.e. $\int_{\mathcal{A}}$ and $\int_{\mathcal{S}}$.

Definition 2.1.7 (Bellman Operators). *The Bellman operator for V_π is a function $T_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:*

$$(T_\pi V_\pi)(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_\pi(s') \right). \tag{2.11}$$

The Bellman operator for Q_π is a function $T_\pi : \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$:

$$(T_\pi Q_\pi)(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q_\pi(s',a'). \tag{2.12}$$

In compact forms:

$$\begin{aligned} T_\pi V_\pi &= V_\pi \\ T_\pi Q_\pi &= Q_\pi. \end{aligned} \tag{2.13}$$

The value functions are fixed points of the Bellman operators T_π. If $\gamma \in (0,1)$, then T_π is a contraction with respect to the maximum norm.

The combination of these two properties provides a method to evaluate the policy π. The fact that T_π is a contraction guarantees the convergence of the Bellman Equation to its fixed point, i.e. the value function. Solving an MDP consist in finding a policy π that produces the optimal value function V^* for each state. To accomplish this, we note that V_π induces a partial ordering over policies, in the following way:

$$\pi \geq \pi' \text{ if and only if } V_\pi(s) \geq V_{\pi'}(s) \text{ for all } s \in \mathcal{S}. \tag{2.14}$$

Meaning that it exists at least one optimal policy with performance better than or equal to all others. As a consequence, the optimal state-value function is $V^*(s) = \max_\pi V_\pi(s)$ and the optimal action-value function is $Q^*(s,a) = \max_\pi Q_\pi(s,a)$.

Theorem 2.1.1. *For any Markov Decision Process:*

- *there exists an optimal policy π^* that is better than or equal to all other policies $\pi^* \geq \pi$, $\forall \pi$;*
- *all optimal policies achieve the optimal state-value function, $V_{\pi^*}(s) = V^*(s)$;*
- *all optimal policies achieve the optimal action-value function, $Q_{\pi^*}(s,a) = Q^*(s,a)$;*
- *there is always a deterministic optimal policy for any MPD:*

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} Q^*(s) \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

Definition 2.1.8 (Bellman Optimality Operators). *The Bellman optimality operator for V^* is a function $T^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:*

$$(T^*V^*)(s) = \max_{a \in \mathcal{A}} \pi(a \mid s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s') \right). \tag{2.16}$$

The Bellman optimality operator for Q^* is a function $T^* : \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$:

$$(T^*Q^*)(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \max_{a' \in \mathcal{A}} \pi(a' \mid s') Q^*(s',a'). \tag{2.17}$$

These are the properties of the Bellman optimality operators:

- *monotonocity*, if $V_1 \leq V_2$ component-wise:

$$T^*V_1 \leq T^*V_2; \tag{2.18}$$

- *max-norm contraction*, for the two vectors V_1 and V_2:

$$\|T^*V_1 - T^*V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty ; \tag{2.19}$$

- V^* is the *unique fixed-point of* T^*;
- for any vector $f \in \mathbb{R}^{|S|}$ and any policy π, we have:

$$\lim_{k \to \infty} (T^*)^k f = V^*. \tag{2.20}$$

The Bellman optimality equations are non-linear and there isn't a closed form solution for the general case. In the next sections, we describe some iterative methods to obtain approximate solutions.

2.2 Dynamic Programming

A naive approach to find an optimal policy that solves an MDP is to enumerate all the deterministic Markov policies, evaluate them, and return the best one. The problem with this solution is that the number of policies is exponential, $|A|^S$, making it intractable even for normal size MDPs.

Dynamic Programming (DP) is a general set of techniques to solve complex sequential problems that can be split into subproblems. The ideal case for a DP algorithm is a problem in which the search of an optimal solution can be separated into the search of optimal solutions for the subproblems, and in which the same subproblems recur many times during the search. This way, the algorithm can cache and reuse results

previously encountered. MDPs satisfy all these properties, having Bellman Equations that naturally give a recursive decomposition and Value functions that store and reuse solutions.

The main assumption of Dynamic Programming is a complete knowledge of the structure of the MDP, an assumption that is rarely satisfied in real world problems, being completely unobtainable in some cases.

We can use DP techniques to perform planning inside an MDP, a problem that can be divided in two parts:

- *Prediction*, the task of returning a value function V_π, given an MDP $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$ and a policy π;
- *Control*, the task of returning the optimal value function V^* and an optimal policy π^*, given an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$.

2.2.1 Policy Evaluation

A Policy Evaluation is an iterative application of the Bellman Expectation backup:

$$V_0 \to V_1 \to \cdots \to V_k \to V_{k+1} \to \cdots \to V_\pi.$$

A full backup is expressed as:

$$V_{k+1}(a) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V_k(s') \right]. \tag{2.21}$$

A sweep is the act of of applying a backup operation to each state. Using synchronous backups, at each iteration $k+1$ we update $V_{k+1}(s)$ from $V_k(s')$ for all states $s \in \mathcal{S}$. This operation is called a full backup. The complete algorithm is stated below:

Algorithm 1 Policy Evaluation

Input: policy π to evaluate, accuracy threshold $\theta > 0$
Output: value function $V \approx V_\pi$

procedure PolicyEvaluation(π, θ)
    Initialize $V_0(s)$ arbitrarily for all $s \in \mathcal{S}$, except for $V_0(terminal) = 0$
    repeat:
        $\Delta \leftarrow 0$
        for each $s \in \mathcal{S}$ **do**
            $V_{k+1}(a) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V_k(s') \right]$
            $\Delta \leftarrow max(\Delta, |V_k(s) - V_{k+1}(s)|)$
    until $\Delta < \theta$

2.2.2 Policy Improvement

Policy Improvement is the task of updating the current policy to obtain a new policy with better performance over the states of the MDP. To start, let's consider a given state s and a deterministic policy π. We can improve the value function $V_\pi(s)$ induced by the policy π with a greedy selection of the action a:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s, a). \tag{2.22}$$

By repeating the same greedy approach for all states $s \in \mathcal{S}$, we obtain a policy π' strictly superior than the previous one. This approach is based on a general result, known as the *Policy Improvement Theorem*, that states the following:

Theorem 2.2.1 (Policy Improvement Theorem). *Let π and π' be a pair of deterministic policies such that $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$, $\forall s \in \mathcal{S}$. Then the policy π' must be as good as, or better than π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$: $V_{\pi'}(s) \geq V_\pi(s)$, $s \in \mathcal{S}$.*

2.2.3 Policy Iteration

Let's consider a policy π that has been improved using V_π to produce a second policy π'. We can take this new policy π' and perform the same step to obtain a third policy π'', that is again guaranteed to be a strict improvement w.r.t. the previous one. By repeating this procedure iteratively, we get a sequence of monotonically improving policies and value functions, of the following form:

$$\pi_0 \xrightarrow{\text{E}} V_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} V^*.$$

This sequence of evaluations $\left(\xrightarrow{\text{E}}\right)$ and improvements $\left(\xrightarrow{\text{I}}\right)$ is known as the *Policy Iteration* algorithm. The convergence to an optimal policy is guaranteed by the fact that a finite MDP can only produce a finite number of policies. In practice, the usual implementation does not wait until the exact convergence, but adopts a stopping condition or runs a priori for k iterations. Below is the complete description of the algorithm:

Algorithm 2 Policy Iteration

Input: policy π to optimize, accuracy threshold $\theta > 0$
Output: value function $V \approx V^*$ and policy $\pi \approx \pi^*$

procedure PolicyIteration(π, θ)
  Initialization:
    $V_0(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
  PolicyEvaluation:
    repeat
      $\Delta \leftarrow 0$
      for each $s \in \mathcal{S}$ **do**
        $V_{k+1}(s) \leftarrow \sum_{s',r} P(s', r \mid s, \pi(s)) [r + \gamma V_k(s')]$
        $\Delta \leftarrow \max(\Delta, |V_k(s) - V_{k+1}(s)|)$
    until $\Delta < \theta$
  PolicyImprovement:
    policy-stable $\leftarrow$ true
    for each $s \in \mathcal{S}$ **do**
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} P(s', r \mid s, a) [r + \gamma V(s')]$
      if old-action $\neq \pi(s)$ **then** policy-stable $\leftarrow$ false
    if policy-stable **then return** $V \approx V^*$ and $\pi \approx \pi^*$
    else go to PolicyEvaluation

2.2.4 Value Iteration

The main limit of Policy Iteration is represented by the fact that each step involves a complete Policy Evaluation, that could significantly affect the time complexity of the process. The *Value Iteration* algorithm modifies Policy Iteration without losing its convergence guarantees. In particular, it consist in stopping the Policy Evaluation step after one sweep (one update of each state). The complete algorithm is described below:

Algorithm 3 Value Iteration

Input: accuracy threshold $\theta > 0$

Output: deterministic policy $\pi \approx \pi^*$

procedure ValueIteration(θ)
    Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$, except for $V(terminal) = 0$
    repeat:
        $\Delta \leftarrow 0$
        for each $s \in \mathcal{S}$ **do**
            $V_{k+1}(s) \leftarrow \max_a \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma V_k(s')]$
            $\Delta \leftarrow \max(\Delta, |V_k(s) - V_{k+1}(s)|)$
    until $\Delta < \theta$
    return $\pi(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma V(s')]$

As we can see, Value Iteration combines in a step, one sweep of Policy Evaluation and one sweep of Policy Improvement. Its update rule is identical to the one of Policy Evaluation if we substitute the sum with a maximum taken over all actions. A second point of view to describe Value Iteration is that its step can be derived by the Bellman Optimality equation by turning it into an update rule.

2.3 Reinforcement Learning

Even if DP approaches can provide a solution to the Prediction and Control problems in MDPs, their usage is limited to tasks in which we have complete knowledge of the transition kernel P. This requirement is rarely satisfied in real world scenarios, where the model is often too big to be represented or simply unknown.

Reinforcement Learning (RL) algorithms, on the other hand, can learn in a MDP just by interacting it, without any knowledge of its internal functioning. In the following section, we present a series of RL approaches, explaining their functioning and properties. Among the methods described, we dedicate particular attention to Value Based and Policy Gradient algorithms, that we adopt in our solution for Meta Reinforcement Learning. In particular, our approach will concern the use of a Value Based methods to improve the convergence properties of Policy Gradient algorithms.

2.3.1 Model-Free vs Model-Based

As a first category of classification for Reinforcement Learning approaches, we describe the fundamental distinction between Model-Free and Model-Based methods:

- *Model-Free* RL algorithms don't have any knowledge about the structure of the environment and of the laws that regulate its transitions and reward functions. These methods rely on *learning* from samples of experience generated by the environment and don't use any internal prediction of the next state reward to condition their behaviour. Classic examples of Model-Free approaches are: Monte Carlo Control, SARSA, Q-learning and Actor-Critic;

- *Model-Based* RL algorithms use a model to simulate the behavior of the environment, or more generally, to infer the future behavior of the environment. For example, given the current state and action, the model can be used to predict the next state and reward. Model-based approaches rely on *planning*, i.e. the construction of a strategy of actions that considers possible future situations before they are experienced. Model-Based approaches include Dynamic Programming, in which the model accesses the transition probabilities and the expected rewards from each state-action pair, and heuristic search.

These methods have also many aspects in common, being both based on looking ahead into future events and on computing approximate value functions.

2.3.2 On-Policy Learning vs Off-Policy Learning

A Reinforcement Learning control problem must balance between its objective to learn the action-value function (and the consequent policy) and the necessity to behave non-optimally to explore enough actions to discover the optimal ones. Naturally, these two requirements generate a trade-off, known as the *Exploitation-Exploration Trade-Off*, present in all Reinforcement Learning problems. We now present the two main approaches to build models that can learn an optimal policy while maintaining an exploratory behavior.

The first class of algorithms, formed by *On-Policy* learners, is in general the simplest approach to the problem. These methods attempt to directly improve the policy that is used to make the decisions. Since they work with only one policy, they tend to compromise, learning actions not towards the optimal policy, but towards an almost-optimal one, while still ensuring a certain level of exploration. Classic examples of such methods are Monte Carlo control and SARSA algorithm, both of which use an ϵ-Greedy Policy Improvement, that, in the case of m discrete actions, works as follows:

$$\pi(s,a) = \begin{cases} 1-\epsilon & \text{if } a^* = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \frac{\epsilon}{m-1} & \text{otherwise.} \end{cases} \tag{2.23}$$

As shown by the formula above, the greedy action is chosen with a probability of $1-\epsilon$, while leaving space for exploration of the rest of choices.

The second class of algorithms consists of *Off-Policy* approaches, a more refined solution that directly translates the conflicting objectives of the trade-off into the use of two policies:

- the first policy π, called *target policy*, is the one that improves to become the optimal policy;
- the second policy $\overline{\pi}$, called *behavioral policy*, it's used to generate the actions and provides the needed exploration.

These methods usually have higher variances compared to their on-policy equivalents, since they don't use data coming from the same policy that undergoes the learning process. They also tend to have slower convergences.

However, they have a series of advantages. They can re-use experience generated from old policies $\pi_1, \pi_1, \ldots, \pi_{t-1}$ and can learn from data generated by various kinds of agents, including human experts. It's also possible to see on-policy methods as a special case of off-policy methods, in which the target and behavioral policies coincide.

To understand the off-policy approach, we need to introduce a technique, called *importance sampling*, that allows to estimate the expected values under one distribution given samples coming from another distribution:

$$\mathbb{E}_{x\sim P}[f(x)] = \sum P(x)f(x) = \sum Q(x)\frac{P(x)}{Q(x)}f(x) = \mathbb{E}_{x\sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right]. \tag{2.24}$$

In order for the sampling to work, we must assume that $P(x) = 0$ whenever $Q(x) = 0$, a property denoted as *coverage*.

Importance sampling is expressed by the *importance-sampling ratio*, which weights returns by measuring the relative probability of the trajectories occurring under the target and behavioral policies. Starting from a state s_t, the probability of the following state-action trajectory $a_t, s_{t+1}, a_{t+1}, \ldots, s_T$, occurring under any policy π, is:

$$\begin{aligned} &P\left(a_t, s_{t+1}, a_{t+1}, \ldots, s_T \mid s_t, s_{t:T-1} \sim \pi\right) \\ &= \pi\left(a_t \mid s_t\right) P\left(s_{t+1} \mid s_t, a_t\right) \pi\left(a_{t+1} \mid s_{t+1}\right) \cdots P\left(s_T \mid s_{T-1}, a_{T-1}\right) \\ &= \prod_{k=t}^{T-1} \pi\left(a_k \mid s_k\right) P\left(s_{k+1} \mid s_k, a_k\right); \end{aligned} \tag{2.25}$$

where P is the state-transition probability function. The resulting importance sampling ratio is:

$$\rho_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi\left(a_k \mid s_k\right) P\left(s_{k+1} \mid s_k, a_k\right)}{\prod_{k=t}^{T-1} \overline{\pi}\left(a_k \mid s_k\right) P\left(s_{k+1} \mid s_k, a_k\right)} = \prod_{k=t}^{T-1} \frac{\pi\left(a_k \mid s_k\right)}{\overline{\pi}\left(a_k \mid s_k\right)}. \tag{2.26}$$

As shown in the formula above, the probabilities of the transitions cancel each other out and the ratio depends only on the sequence of the two policies. The importance

sampling ratio is used to correct the return values collected along the trajectory under the behavioral policy $\overline{\pi}$, in the following way:

$$V_\pi(s) = \mathbb{E}[\rho_{t:T-1} r_t \mid s_t = s]. \tag{2.27}$$

The most straightforward way to implement the expected value is called *ordinary importance sampling*, and it's used in Off-Policy Monte Carlo control. It consists in averaging the returns obtained in state s over the episode. Another techniques is the *weighted importance sampling*, in which the returns are weighted by the ratios.

Especially in its ordinary version, importance sampling may lead to dramatic increases in prediction uncertainty, since the variance of the ratios can be unbounded. This aspect is in part mitigated by the weighted version, at the cost of introducing some bias in the evaluation.

Most algorithms can be adapted to an off-policy version. Common examples are Off-Policy Monte Carlo, Off-Policy SARSA and Q-learning.

2.3.3 Online Learning vs Offline Learning

Another important distinction in the characterization of RL algorithms separates Online and Offline approaches. These are general concepts, that can be applied to all fields of machine learning.

Online learning algorithms work with data incoming as a stream of inputs and use them as soon as they are available. A strictly online algorithm takes each new sample as it arrives and uses it to improve incrementally. Online learning approaches are data efficient: the data is discarded once it's being consumed, since the knowledge that could be derived from it is already embedded in the learner. Even if it's not a requirement, it's usually desirable for an online algorithm to forget older examples over time, so that it can adapt to non-stationary populations. This produces a learner able to adapt on the fly to changing trends in the data. A classic example of Online Learning is Stochastic gradient descent, that updates the gradient at each data sample.

On the other hand, *Offline learning* algorithms acquire all the data in bulk, from a dataset. If there are changes in the data, it's necessary to re-run them from the start in order to learn. These approaches are also called batch learning. Examples of offline learning algorithms are Batch gradient descent, SVMs and Random Forests.

In a sense, online algorithms can be seen as a generalization of their offline equivalent. Indeed, it's possible to simulate offline learning by feeding a stored dataset to an online learner, while it's impossible to accomplish learning on the moment with an offline learner. However, this doesn't mean that they are generally superior: as state by the "No Free Lunch Theorem" (Wolpert and Macready, 1997), there isn't an approach guaranteed to beat another in every possible case. Online learning often has its downsides in terms of sample efficiency and computational costs.

For these reasons, many attempts have been made to find a compromise between the two approaches, such as mini-batches in neural network training. In the field of reinforcement learning, Deep Q Networks (Mnih et al., 2013) use a mixed approach called experience replay. It consists of storing a rolling history and sample from it, instead of having all the experience theoretically needed to fully train an agent.

2.3.4 Value Based Reinforcement Learning

Value-Based methods aim to learn the state-value function $V(s)$ or action-value function $Q(s,a)$ of an MDP. They are usually based on Temporal Difference (TD) learning, that in its simplest form updates the value $V(s_t)$ towards the estimated return $r_{t+1} + \gamma V(s_{t+1})$:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right).^{2} \tag{2.28}$$

A classic example of Value-Based RL is Q-learning (Watkins and Dayan, 1992), an on-policy Temporal Difference control algorithm that was one of the first breakthroughs in the field. Its update rule is defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]. \tag{2.29}$$

In Q-Learning, the learned action-value function Q directly approximates the optimal action-value function Q^*, independently of the behavioral policy used in the episode. Of course, the policy still determines which state-action pairs are visited and updated. Nonetheless, the only requirement to reach convergence is that all pairs continue to be updated. Under this assumption, it can be demonstrated that Q converges to Q^* with probability 1. The Q-learning algorithm is shown below in its procedural form.

Algorithm 4 Q-Learning

Input: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

procedure Q-Learning(α, ε)
  Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, except for $Q(terminal, \cdot) = 0$
  for each episode **do**
    Initialize s
    repeat for each step of the episode
      Choose a from s using policy derived from Q (e.g., ε-greedy)
      Take action a, observe r, s'
      $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_a Q(s', a) - Q(s,a)\right]$
      $s \leftarrow s'$
    until s is terminal

[2] In Equations 2.28,2.29 the term $\alpha \in (0, 1]$ is a learning rate.

Fitted Q Iteration

As we saw in the introductory sections, the Q-function can be represented in tabular form, given that the state and action spaces are finite and of reasonable dimensions. If instead we have to deal with continuous or very large spaces, it's not possible to contain the whole function inside a table. For this reason, we need to find an approximation of the Q-function over all the state-action space. To tackle this problem, we describe *Fitted Q Iteration* (Ernst et al., 2005), a batch mode learning algorithm that takes inspiration from the Q-learning paradigm.

This approach takes as input a set of four-tuples $\mathcal{F}$, such that:

$$\mathcal{F} = \{(s_i, a_i, s'_i, r_i)\}, \quad i = 1, \ldots, |\mathcal{F}|; \tag{2.30}$$

where each sample includes the action a_i taken in state s_i and the resulting reward r_i and next state s'_i. Then, applying the idea of Fitted Value Iteration, it computes an approximation of the Q-function by means of a series of kernel-based regressions. This solution allows to transfer the generalization capabilities of regression algorithms to the problem of reinforcement learning. The approximation is made iteratively, extending at each step the optimization horizon:

- the first iteration approximates the function $Q_1(s, a) = \mathbb{E}[r_t \mid s_t = s, a_t = a]$, corresponding to a 1-step optimization. This step is achieved through a batch mode regression algorithm that takes as inputs the state-action pairs (s_t, a_t) and as target outputs the instantaneous rewards r_t, such that:

$$Q_{1,t} = r_t; \tag{2.31}$$

- the n-th iteration generalizes the first one, approximating a Q_n-function corresponding to an n-step optimization horizon. To maintain the framework of a batch mode regression, the training set uses the same state-action pairs as input and derives the target output through the approximation function $\hat{Q}_{n-1}$ of the previous step, as following:

$$Q_{n,t} = r_t + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{n-1}(s_{t+1}, a); \tag{2.32}$$

  where $\gamma \in [0, 1)$ is the discount factor. This procedure is equivalent to performing a regression on the outcome of the application of Bellman optimality operator.

Other than interrupting the algorithm after a predefined number of iterations, it's possible to define a stopping condition. Given a sequence of optimal policies π^*_N, an error bound on the sub-optimality in terms of number of iterations is given by the following equation:

$$\left| J^{\pi^*_N}_\infty - J^{\pi^*}_\infty \right|_\infty \leq 2 \frac{\gamma^N B_r}{(1-\gamma)^2}. \tag{2.33}$$

Given a bound on the reward B_r and a desired level of accuracy, it's possible to compute the minimum value of N that satisfies the fixed tolerance.

The final policy obtained when the stopping condition is reached is the following:

$$\hat{\pi}_N^*(x) = \arg\max_{a \in \mathcal{A}} \hat{Q}_N(s, a). \tag{2.34}$$

The complete FQI algorithm is listed below:

Algorithm 5 Fitted Q-Iteration

Input: a set of four-tuples $\mathcal{F} = \{(s_i, a_i, s_i', r_i)\}$ and a regression algorithm f

1: **procedure** FQI($\mathcal{F}, f$)
2: Initialize $\hat{Q}_n = 0$ everywhere on $\mathcal{S} \times \mathcal{A}$
3: $n \leftarrow 0$
4: **repeat**
5: $n \leftarrow n + 1$
6: Build the training set $\mathcal{TS} = \left\{ \left(i^l, o^l\right), l = 1, \cdots, |\mathcal{F}| \right\}$ based on $\hat{Q}_{N-1}$ and $\mathcal{F}$:

$$i^l = \left(s_t^l, a_t^l\right)$$
$$o^l = r_t^l + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{n-1}\left(s_{t+1}^l, a\right)$$

7: Use f to induce from $\mathcal{T}S$ the function $\hat{Q}_n(s, a)$
8: **until** the stopping conditions are reached

An interesting aspect of the algorithm is that, at each iteration, the call to the supervised learning algorithm is independent from the others. This freedom allows us to tailor the regression to the characteristics of each step, reaching the optimal bias/variance trade-off.

The Q function can assume a completely unpredictable shape, so it's necessary that the algorithm adopted for the regression offers great generalization qualities. Usually, Fitted Q Iteration performs this task through tree-based methods. These approaches are non-parametric and flexible, in addition to having high computational efficiency and maintaining robustness to noise.

This family of regression algorithms is top-down: they create a partition starting with a single subset, which is then refined by splitting its subsets into pieces. Various approaches may differ by the number of regression trees, the growth method of the tree and, when an ensemble of trees is used, the method to build the training set of a particular tree from the original training set.

To conclude, we describe some tree regression approaches that can be used with FQI:

- **Tree Bagging**: bagging, also known as Bootstrap Aggregation, is a meta algorithm that averages the outputs of a series of noisy but approximately unbiased models to reduce the uncertainty of the prediction. This technique is particularly useful in high-variance and low-bias contexts, of which trees are a classic example. In our context, Tree Bagging builds a group of "bootstrap replicas": a sequence of sets built by *sampling with replacement* from the same distribution. This replicas are then used to form an ensemble of unpruned "Classification and Regression Trees" (CART). The simplest CART tree is a binary decision tree in which at each the input space is partitioned into two parts along a dimension. The leaf nodes contain the expected value of the output and are used to perform the predictions. The expansion of the nodes is interrupted if the number of samples in the node is less than a threshold. Tree Bagging significantly improves the accuracy of the model, at the cost of an increase in computational complexity.

- **ExtraTrees**: another technique that can be adopted is known as Extremely Randomized Trees (Geurts et al., 2006), or "ExtraTrees". The algorithm is similar to the previous approaches for its use of an ensemble of trees, but adds some alterations. The trees are derived from the complete training set, instead of the bootstrap replicas seen before. In addition, a split in a node is performed by randomly selecting k cut directions and cut points. A score is computed for each of them and the selected direction is the one that maximizes the score.

2.3.5 Policy Gradient Methods

The methods described so far focus on learning the state or action value function and then choosing greedy[3] policies based on their estimates. Policy Gradient (PG) methods aim to skip this step by directly learning a parameterized policy that can select actions without considering a value function. These methods may still use a value function to learn the policy parameters, but they don't depend on it to select the action.

The policy is parameterized through a set of inputs, denoted as $\boldsymbol{\theta} \in \mathbb{R}^d$, and is expressed as below:

$$\pi(a \mid s, \boldsymbol{\theta}) = P(a_t = a \mid s_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}). \tag{2.35}$$

There are many possible choices for parameterization, with the only constraint that the policy $\pi(a \mid s, \boldsymbol{\theta})$ is differentiable with respect to its parameters $\boldsymbol{\theta}$. A gradient is a column vector of partial derivatives of $\pi(a \mid s, \boldsymbol{\theta})$ with respect to the components of $\boldsymbol{\theta}$.

Our objective is to measure the performance of our learner, dependent on the value of the parameterization. The performance can be expressed as $J(\boldsymbol{\theta}) = V_{\pi_{\boldsymbol{\theta}}}(s_0)$, where s_0

[3]In many cases, in order to grant enough exploration of the state-action space, the chosen policy is the so-called ϵ-greedy: with a probability ϵ, the action is chosen randomly on the action space.

is the initial state of the environment, and the goal is to find a maximum of this function through Gradient Ascent (GA): $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta})$.

The parameter $\boldsymbol{\theta}$ influences both the action selections and the distribution of the visited states. Computing the impact of the parameters on the choice of the actions is usually simple if we know the current state of the environment. On the other hand, computing the impact that the parameter has on the state distribution can be really difficult, because it's usually dependent of an unknown function of the environment.

This problem finds an answer in the Policy Gradient Theorem, an essential result upon which all PG methods are based.

Theorem 2.3.1 (Policy Gradient Theorem). *The Policy Gradient Theorem states the following:*

$$\nabla J(\boldsymbol{\theta}) \propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} Q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta}). \tag{2.36}$$

In other words, it's possible to estimate the gradient of the performance with respect to the policy parameter $\nabla J(\boldsymbol{\theta})$ as an explicit function without deriving the state distribution.

REINFORCE Algorithm

The first PG method we analyze is known as the REINFORCE algorithm. Its update can be derived directly from the Policy Gradient Theorem, in the following way:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a Q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta}) \\
&= \mathbb{E}_\pi \left[\sum_a Q_\pi\left(s_t, a\right) \nabla \pi\left(a \mid s_t, \boldsymbol{\theta}\right) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi\left(a \mid s_t, \boldsymbol{\theta}\right) Q_\pi\left(s_t, a\right) \frac{\nabla \pi\left(a \mid s_t, \boldsymbol{\theta}\right)}{\pi\left(a \mid s_t, \boldsymbol{\theta}\right)} \right] \\
&= \mathbb{E}_\pi \left[Q_\pi\left(s_t, a_t\right) \frac{\nabla \pi\left(a_t \mid s_t, \boldsymbol{\theta}\right)}{\pi\left(a_t \mid s_t, \boldsymbol{\theta}\right)} \right] \\
&= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi\left(a_t \mid s_t, \boldsymbol{\theta}\right)}{\pi\left(a_t \mid s_t, \boldsymbol{\theta}\right)} \right].
\end{aligned} \tag{2.37}$$

The symbols s_t and a_t represent the state and the action at time t. They are introduced by replacing a sum over all possible values of a random variable with its expectation under π, and then sampling that expectation. The last step derives from the fact that $\mathbb{E}_\pi[G_t \mid s_t, a_t] = Q_\pi(s_t, a_t)$, where $G_t = \sum_k \gamma^k r_{t+k+1}$ is the usual return.

In the end, we obtain a quantity proportional to $\nabla J(\boldsymbol{\theta})$ that, at each time step, does not depend on the state distribution and therefore can be sampled. With this result, we

obtain a definition for a stochastic gradient ascent algorithm, called the REINFORCE update:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}{\pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}. \tag{2.38}$$

In this update rule, the direction of increment is given by the gradient of the probability of taking the action a_t, the direction that most increases the chance of selecting action a_t next time we are in state s_t. The value of the increment is proportional to the product of the gradient and the return G_t, to push the parameter in the directions of the actions with higher returns. It's also inversely proportional to the action probability, to avoid giving an advantage to frequently selected actions.

The REINFORCE algorithm can be considered as a Monte Carlo approach, since it requires a complete knowledge of the episode in order to work. This is visible by the fact that it uses the complete return G_t. The full algorithm proceeds as below:

Algorithm 6 REINFORCE

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$, a step size $\alpha > 0$

procedure REINFORCE(π, α)
    Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^d$
    loop for each episode
        Generate an episode $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$, following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$
        loop for each step of the episode $t = 0, 1, \ldots, T-1$
            $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G_t \nabla \ln \pi \left(a_t \mid s_t, \boldsymbol{\theta}\right)$

It's possible to generalize the Policy Gradient Theorem to include a comparison of the action-value to an arbitrary baseline $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(Q_\pi(s,a) - b(s)\right) \nabla \pi(a \mid s, \boldsymbol{\theta}). \tag{2.39}$$

The baseline $b(s)$ can assume any form, with the only requirement of not varying with a. Under this constraint, the gradient $\nabla J(\boldsymbol{\theta})$ remains unchanged because the subtracted quantity is equal to zero:

$$\sum_a b(s) \nabla \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla \sum_a \pi(a \mid s, \boldsymbol{\theta}) = b(s) \nabla 1 = 0. \tag{2.40}$$

This property can can be used to derive an update rule for new version of the algorithm, known as REINFORCE with baseline:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t - b\left(s_t\right)\right) \frac{\nabla \pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}{\pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}. \tag{2.41}$$

A common choice for a baseline is the state-value function, resulting in the use of the Advantage Function $A(s,a) = Q(s,a) - V(s)$ in the gradient ascent update. Another option is to choose the baseline that minimizes the variance of the gradient estimate, as done in (Deisenroth et al., 2013).

Actor-Critic Methods

A Policy Gradient gradient method is said to be an Actor-Critic if it aims to learn the value function, in addition to the optimal policy model. This function is used to assist the agent in the policy update, by evaluating the chosen actions, and to reduce variance during learning.

An Actor-Critic approach is formed by two models, that work alternatively:

- the *critic* is responsible of the estimation of the value function. It optimizes the function parameters $\mathbf{w}$, referred to the action-value $Q(a \mid s, \mathbf{w})$ or the state-value $V(s, \mathbf{w})$ depending on the case;
- the *actor* updates the policy in the direction suggested by the critic, by optimizing the policy parameters $\boldsymbol{\theta}$ for $\pi(a \mid s, \boldsymbol{\theta})$.

An online, one-step version of an Actor-Critic algorithm replaces the return of REINFORCE with the one-step return $G_{t:t+1}$ and use a state-value function $\hat{V}$ estimated by the critic as a baseline:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{V}\left(s_t, \mathbf{w}\right) \right) \frac{\nabla \pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}{\pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)} \\
&= \boldsymbol{\theta}_t + \alpha \left(r_{t+1} + \gamma \hat{V}\left(s_{t+1}, \mathbf{w}\right) - \hat{V}\left(s_t, \mathbf{w}\right) \right) \frac{\nabla \pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)}{\pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right)} \\
&= \boldsymbol{\theta}_t + \alpha \delta_t \nabla \ln \pi \left(a_t \mid s_t, \boldsymbol{\theta}_t\right).
\end{aligned} \tag{2.42}$$

The complete algorithm is outlined below:

Algorithm 7 Actor-Critic

Input: a policy $\pi(a \mid s, \boldsymbol{\theta})$, a state-value function $\hat{V}(s, \mathbf{w})$

procedure ActorCritic($\pi, \hat{V}$)
  Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
  loop for each episode
    Initialize s, $I \leftarrow 1$
    while s is not terminal (for each time step) **do**
      Sample $a \sim \pi(\cdot \mid s, \boldsymbol{\theta})$, take action a and observe s', r
      $\delta \leftarrow r + \gamma \hat{V}(s', \mathbf{w}) - \hat{V}(s, \mathbf{w})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{V}(s, \mathbf{w})$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} I \delta \nabla \ln \pi(a \mid s, \boldsymbol{\theta})$
      $I \leftarrow \gamma I$, $s \leftarrow s'$

2.3.6 Trust Region Policy Optimization

We now describe some limitations that affect PG methods and one approach that aims to solve them, known as Trust Region Policy Optimization (Schulman et al., 2017).

The first limit comes directly from the use of GA as the update rule. Using the gradient $\nabla J(\boldsymbol{\theta})$ as the increment means updating the policy towards the direction with the steepest ascent of the reward. The use of the first derivative produces the great disadvantage of approximating any kind of surface to a flat one. If the surface isn't particularly regular, the risk is to make overconfident moves that lead to local mimima from which is difficult to recover.

The second limit of these methods is the difficulty of setting generally well-performing hyperparameters. For example, choosing a good step size for a non-trivial learning task can be a challenging exercise.

To conclude, these methods require large amounts of data to train succesfully. In particular, an update in episode-based methods require sampling a complete trajectory, which can be arbitrarily long. Especially in real world control applications, this is extremely sample inefficient.

Minorization-Maximization algorithm

The Minorization-Maximization algorithm (MM) is an iterative method that, at each step, maximizes a lower bound function that locally approximates the expected reward. Doing so, it guarantees that every policy update improves the expected rewards. The algorithm starts by guessing an initial policy and finding a lower bound M that locally approximate the expected reward J of the current guess. Then, it computes the optimal value of M and uses it as the policy for the next step. The iteration is repeated by

finding the new lower bound and optimizing its value. In the end, the estimate will converge to the optimal policy.

Trust Regions

One of the limitations of Gradient Ascent methods generates from the fact that they operate in a *line search*, that consists in deciding the update direction and taking a step towards it. An alternative approach, known as Trust Region, constraints the movement by defining a maximum step size, instead of looking for a direction. The next value is selected as the optimal point within this "trusted" region. We define δ as the radius of the region, m an approximation of the objective function f. The problem has the following formulation:

$$\begin{aligned} &\max_{s \in \mathbb{R}^n} m_k(s) \\ &\text{s.t. } \|s\| \leq \delta. \end{aligned} \tag{2.43}$$

The process is iterated to reach an optimum. The learning speed can be controlled by expanding or shrinking the value of δ, according to the curvature of the surface. A possible decision metric is to shrink the trust region when m is a poor approximation of f and expanding otherwise. An alternative is to shrink the region if the divergence of the new and current policy increases (and vice versa).

To tackle the problem of sample inefficiency, we adopt the importance sampling technique, that let us use samples from old policies to compute the policy gradient. Let's revise the equation involved in the Policy Gradient update:

$$\nabla_{\boldsymbol{\theta}} J\left(\pi_{\boldsymbol{\theta}}\right) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right) A_t\left(s_t, a_t\right) \right]. \tag{2.44}$$

Inverting the derivative we obtain the following objective function:

$$L^{PG}(\boldsymbol{\theta}) = \mathbb{E}_t \left[\log \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right) A_t \right]. \tag{2.45}$$

Which is equivalent to the following objective function, that uses importance sampling:

$$L^{IS}_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{\theta}) = \mathbb{E}_t \left[\frac{\pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right)}{\pi_{\boldsymbol{\theta}_{\text{old}}}\left(a_t \mid s_t\right)} A_t \right]. \tag{2.46}$$

In fact, the two objective functions share the same derivative, therefore the same optimal value of $\boldsymbol{\theta}$:

$$\underbrace{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right)\big|_{\boldsymbol{\theta}_{\text{old}}}}_{\nabla_{\boldsymbol{\theta}} L^{PG}(\boldsymbol{\theta})} = \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right)\big|_{\boldsymbol{\theta}_{\text{old}}}}{\pi_{\boldsymbol{\theta}_{\text{old}}}\left(a_t \mid s_t\right)} = \underbrace{\nabla_{\boldsymbol{\theta}} \left(\frac{\pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right)}{\pi_{\boldsymbol{\theta}_{\text{old}}}\left(a_t \mid s_t\right)} \right)\Bigg|_{\boldsymbol{\theta}_{\text{old}}}}_{\nabla_{\boldsymbol{\theta}} L^{IS}_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{\theta})}. \tag{2.47}$$

Having two policies in the optimization objective is the key to constrain the policy change.

Optimization Problem

As usual, the objective is to find the optimal policy $\pi_{\boldsymbol{\theta}}$ such that $J(\boldsymbol{\theta})$ is maximum. Since the variable to optimize is $\pi_{\boldsymbol{\theta}}$, the problem can also be formalized as:

$$\max_{\pi_{\boldsymbol{\theta}}} J\left(\pi_{\boldsymbol{\theta}}\right) = \max_{\pi_{\boldsymbol{\theta}}} J\left(\pi_{\boldsymbol{\theta}}\right) - J(\pi_{\boldsymbol{\theta}_{old}}). \tag{2.48}$$

We define the lower bound function $\mathcal{L}$ to use in the MM iterations as:

$$\mathcal{L}_{\pi_{\boldsymbol{\theta}_{old}}}\left(\pi_{\boldsymbol{\theta}}\right) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\boldsymbol{\theta}_{old}}} \\ a\sim\pi_{\boldsymbol{\theta}_{old}}}} \left[\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}_{old}}(a \mid s)} A(s,a)\right] = \mathop{\mathbb{E}}_{\tau\sim\pi_{\boldsymbol{\theta}_{old}}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right)}{\pi_{\boldsymbol{\theta}_{old}}\left(a_t \mid s_t\right)} A\left(s_t, a_t\right)\right] \tag{2.49}$$

where $d^{\pi} = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P\left(s_t = s \mid \pi_{\boldsymbol{\theta}_{old}}\right)$.
It can be demonstrated that the following inequality holds:

$$\left|J\left(\pi_{\boldsymbol{\theta}}\right) - \left(J(\pi_{\boldsymbol{\theta}_{old}}) + \mathcal{L}_{\pi_{\boldsymbol{\theta}_{old}}}\left(\pi_{\boldsymbol{\theta}}\right)\right)\right| \leq C\sqrt{\mathrm{E}_{s\sim d^{\pi_{\boldsymbol{\theta}_{old}}}} D_{KL}\left(\pi_{\boldsymbol{\theta}} || \pi_{\boldsymbol{\theta}_{old}}\right)}; \tag{2.50}$$

where $C \propto \frac{\epsilon\gamma}{(1-\gamma)^2}$ and D_{KL} is the Kullback-Leibler divergence, a measures of the difference between two probability distributions P and Q:

$$D_{KL}(P \mid\mid Q) = \sum_{x=1}^{N} P(x) \log \frac{P(x)}{Q(x)}. \tag{2.51}$$

Moving around the inequality, we obtain the lower bound M to be maximized by the MM algorithm:

$$J\left(\pi_{\boldsymbol{\theta}}\right) - J(\pi_{\boldsymbol{\theta}_{old}}) \geq \underbrace{\mathcal{L}_{\pi_{\boldsymbol{\theta}_{old}}}\left(\pi_{\boldsymbol{\theta}}\right) - C\sqrt{\sum_{s\sim d^{\pi_{\boldsymbol{\theta}_{old}}}} D_{KL}\left(\pi_{\boldsymbol{\theta}} \mid \pi_{\boldsymbol{\theta}_{old}}\right)}}_{M}. \tag{2.52}$$

As a result of the the Lagrangian Duality, the objective can be expressed as follows:

$$\begin{aligned} &\max_{\pi_{\boldsymbol{\theta}}} \mathcal{L}_{\pi_{\boldsymbol{\theta}_{old}}}\left(\pi_{\boldsymbol{\theta}}\right) \\ &\text{s.t.} \mathop{\mathbb{E}}_{s\sim d^{\pi_{\boldsymbol{\theta}_{old}}}} \left[D_{KL}\left(\pi_{\boldsymbol{\theta}} \mid \pi_{\boldsymbol{\theta}_{old}}\right)\right] \leq \delta. \end{aligned} \tag{2.53}$$

While the two formulations are mathematically equivalent, the hyperparameter C is usually harder to tune than δ.

Natural Policy Gradient

The Natural Policy Gradient (Kakade, 2001) is a method to analytically solve the optimization objective. To start, we expand the Taylor's series of both terms to the second

order. Discarding all terms close to 0, we obtain the following relations:

$$\begin{aligned}
&\mathcal{L}_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) \approx g^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_k), \text{ where } g = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}_k}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k} \\
&D_{KL}(\boldsymbol{\theta} \mid \boldsymbol{\theta}_k) \approx \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T H (\boldsymbol{\theta} - \boldsymbol{\theta}_k), \text{ where } H = \nabla^2_{\boldsymbol{\theta}} D_{KL}(\boldsymbol{\theta} \mid \boldsymbol{\theta}_k)\big|_{\boldsymbol{\theta}_k}.
\end{aligned} \tag{2.54}$$

H measure the sensitivity (curvature) of the policy relative to the parameter $\boldsymbol{\theta}$, g is the policy gradient. The objective becomes:

$$\begin{aligned}
\boldsymbol{\theta}_{k+1} = &\arg\max_{\boldsymbol{\theta}} g^T (\boldsymbol{\theta} - \boldsymbol{\theta}_k) \\
&\text{s.t. } \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T H (\boldsymbol{\theta} - \boldsymbol{\theta}_k) \leq \delta.
\end{aligned} \tag{2.55}$$

From which we obtain the Natural Policy Gradient analytical solution:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2\delta}{g^\top H^{-1} g} H^{-1} g}. \tag{2.56}$$

The term $H^{-1}g$ assumes different values in different parameterizations, but it generates parameter updates $\Delta\boldsymbol{\theta}$ that map to the same policy change, independently from the parameterization choice. The Natural Policy Gradient is a second order optimization method: H is a Hessian matrix in the generic form of $\nabla^2 f$. In particular, H measures the second-order derivative (curvature) of the log probability of the policy, and it's known as Fisher Information Matrix (FIM):

$$F = \mathop{\mathrm{E}}_{s,a \sim \boldsymbol{\theta}^k} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)|_{\boldsymbol{\theta}_k} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)\Big|_{\boldsymbol{\theta}_k} \right]. \tag{2.57}$$

If the policy has a complex parameterization, finding the inverse of H can be expensive. The result can also be numerically unstable. For these reasons, we settle for an approximate solution of the product: $x_k \approx \hat{H}_k^{-1} \hat{g}_k$. We convert this equation into an optimization problem for a quadratic equation: Solving an equation of the form $Ax = b$ can be converted into the optimization problem of minimizing $f(x) = \frac{1}{2} x^T A x - b^T x$, since $f'(x) = Ax - b = 0$. Our problem becomes:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T H x - g^T x. \tag{2.58}$$

Since the objective function is quadratic, we can optimize it using the Conjugate Gradient method. Given a second-degree function with n parameters, this approach guarantees to find the optimal point in at most n steps. The algorithms makes its first update in the direction of the gradient, stopping in the optimal point with respect to that direction. In the following steps, each direction is orthogonal (conjugate) to all previous directions.

Proceeding this way, the algorithm "consumes" all dimensions one at a time, never undoing part of the progress made previously. As a consequence, given an n-dimensional space, the solution can be found in at most n steps.

The Trust Region Policy Optimization (TRPO) algorithm combines these elements together, applying the Conjugate Gradient method to the Natural Policy gradient algorithm. In practical implementations, the trust region for the natural policy gradient is small, so it's relaxed to a bigger parameter. Additionally, the quadratic approximation decreases the accuracy, causing problems to the policy updates. To contrast these tendencies, at each step we verify that the KL-divergence for the policy $\boldsymbol{\theta}$ is lower than δ and that $\mathcal{L}(\boldsymbol{\theta}) \geq 0$. If the verification fails, it's necessary to reduce the natural policy gradient by a factor of $\alpha \in (0, 1)$ until the conditions are met.

Unfortunately, even if TRPO reduces the problem of inverting the FIM with the conjugate gradient method, it still requires the knowledge of F at each time step. An accurate estimate of this value requires a large batch of rollouts, making TRPO less sample efficient than first-order methods.

2.4 Lipschitz MDP

We conclude the chapter by introducing the notions of Lipschitz continuity and Lipschitz MDP as described by (Pirotta et al., 2015). These concepts will be used in Chapter 4 as assumptions to derive some boundaries in the performance of our meta-RL approach.

Definition 2.4.1 (Lipschitz Continuity). *Given two metric sets (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 (L_f-LC), with $L_f \geq 0$ if:*

$$\forall x, x' \in X, d_Y\left(f(x), f\left(x'\right)\right)\right) \leq L_f d_X\left(x, x'\right). \tag{2.59}$$

A function $f : X \to Y$ is called Pointwise Lipschitz continuous (PLC) in state x if there exists a constant $L_f(x)$ such that:

$$\forall x' \in X, d_Y\left(f(x), f\left(x'\right)\right) \leq L_f(x) d_X\left(x, x'\right) \text{ where } \forall x \in X, L_f(x) \leq L_f. \tag{2.60}$$

The Lipschitz semi-norm over a function space $\mathcal{F}(X, Y)$ is defined as:

$$\|f\|_L := \sup_{x_1 \neq x_2} \left\{ \frac{d_Y\left(f\left(x_1\right), f\left(x_2\right)\right)}{d_X\left(x_1, x_2\right)} \right\}. \tag{2.61}$$

For real functions, the usual metric is the Euclidean distance: $d_Y\left(y, y'\right) = \|y - y'\|^2$. For distributions, a common metric is the Kantorovich, also known as L^1-Wasserstein distance:

$$\mathcal{K}(p, q) := \sup_{f : \|f\|_L \leq 1} \left\{ \left\| \int_X f d(p - q) \right\| \right\}. \tag{2.62}$$

Assumption 2.4.1 (Lipschitz MDP). *Let $\mathcal{M}$ be an MDP. $\mathcal{M}$ is called (L_P, L_R)-LC if for all $(s,a),(\bar{s},\bar{a}) \in \mathcal{S}\times\mathcal{A}$:*

$$\begin{aligned} \mathcal{K}(P(\cdot \mid s,a), P(\cdot \mid \bar{s},\bar{a})) &\leq L_P d_{\mathcal{S}\times\mathcal{A}}((s,a),(\bar{s},\bar{a})) \\ \left|R(s,a) - R(\bar{s},\bar{a})\right| &\leq L_R d_{\mathcal{S}\times\mathcal{A}}((s,a),(\bar{s},\bar{a})). \end{aligned} \tag{2.63}$$

Assumption 2.4.2 (Lipschitz Policy). *Let $\pi \in \Pi$ be a Markovian stationary policy. π is called L_π-LC if for all $s,\bar{s} \in \mathcal{S}$:*

$$\mathcal{K}(\pi(\cdot \mid s), \pi(\cdot \mid \bar{s})) \leq L_\pi d_{\mathcal{S}}(s,\bar{s}). \tag{2.64}$$

Since we usually deal with parametric policies, other useful assumptions rely on the Lipschitz continuity w.r.t. the policy parameters $\boldsymbol{\theta}$ and their gradient:

Assumption 2.4.3 (Lipschitz Parametric Policy). *Let $\pi_{\boldsymbol{\theta}} \in \Pi$ be a policy parameterized in the parameters space $\boldsymbol{\theta}$. A LC-policy π satisfies the following conditions:*

$$\begin{aligned} &\forall \boldsymbol{\theta} \in \Theta, \forall s,s' \in \mathcal{S} \quad \mathcal{K}\left(\pi_{\boldsymbol{\theta}}(\cdot \mid s), \pi_{\boldsymbol{\theta}}\left(\cdot \mid s'\right)\right) \leq L_{\pi_{\boldsymbol{\theta}}} d_{\mathcal{S}}\left(s,s'\right) \\ &\forall s \in \mathcal{S}, \forall \boldsymbol{\theta},\boldsymbol{\theta}' \in \Theta \quad \mathcal{K}\left(\pi_{\boldsymbol{\theta}}(\cdot \mid s), \pi_{\boldsymbol{\theta}'}(\cdot \mid s)\right) \leq L_\pi(\boldsymbol{\theta}) d_\Theta\left(\boldsymbol{\theta},\boldsymbol{\theta}'\right). \end{aligned} \tag{2.65}$$

Assumption 2.4.4 (Lipschitz Gradient of Policy Logarithm). *The gradient of the policy logarithm must satisfy the conditions of:*

1. *Uniformly bounded gradient: $\forall (s,a) \in \mathcal{S}\times\mathcal{A}, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1,\dots,d$*

$$\left|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s)\right| \leq M^i_{\boldsymbol{\theta}}; \tag{2.66}$$

2. *State-action LC: $\forall\,(s,s',a,s') \in \mathcal{S}^2\times\mathcal{A}^2, \forall \boldsymbol{\theta} \in \Theta, \forall i = 1\dots,d$*

$$\left|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s) - \nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}\left(a' \mid s'\right)\right| \leq L^i_\nabla \log \pi d_{\mathcal{S}\times\mathcal{A}}\left((s,a),\left(s',a'\right)\right); \tag{2.67}$$

3. *Parametric PLC: $\forall\,(\boldsymbol{\theta},\boldsymbol{\theta}') \in \Theta, \forall (s,a) \in \mathcal{S}\times\mathcal{A}, \forall i = 1,\dots,d$*

$$\left|\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a \mid s) - \nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}'}(a \mid s)\right| \leq L^i_{\nabla \log \pi}(\boldsymbol{\theta}) d_\Theta\left(\boldsymbol{\theta},\boldsymbol{\theta}'\right). \tag{2.68}$$

Under assumptions 2.4.2 and 2.4.4, it's possible to prove the LC of the corresponding value functions.

Lemma 2.4.1 (Lipschitz value functions). *Given an (L_P, L_R)-LC MDP and a L_π-LC stationary policy π, if $\gamma L_P\left(1 + L_\pi\right) < 1$, then the Q-function Q_π is L_{Q_π}-LC and the V function is L_{V_π}-LC w.r.t. the joint state-action space:*

$$L_{Q_\pi} = \frac{L_R}{1 - \gamma L_P\left(1 + L_\pi\right)}; \quad L_{V_\pi} = L_{Q_\pi}\left(1 + L_\pi\right). \tag{2.69}$$

Proposition 2.4.2 (Lipschitz bound on the return). *Given an (L_P, L_R)-LC MDP, for any pair of stationary policies corresponding to parameters $\boldsymbol{\theta}$ and $\widehat{\boldsymbol{\theta}}$, the absolute difference between the relative performances $\pi_{\boldsymbol{\theta}}$ and $\pi_{\widehat{\boldsymbol{\theta}}}$ can be bounded as follows:*

$$\left|J_\mu^{\boldsymbol{\theta}} - J_\mu^{\widehat{\boldsymbol{\theta}}}\right| \leq \frac{L_R}{1-\gamma}\mathcal{K}\left(\zeta_\mu^{\boldsymbol{\theta}}, \zeta_\mu^{\widehat{\boldsymbol{\theta}}}\right). \tag{2.70}$$

Moreover, if Assumption 2.4.3 holds, and $\gamma L_P\left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) < 1$, then the performance measure $J_\mu^{\boldsymbol{\theta}}$ if $L_J(\boldsymbol{\theta})$-PLC w.r.t. the policy parameters:

$$\left|J_\mu^{\boldsymbol{\theta}} - J_\mu^{\widehat{\boldsymbol{\theta}}}\right| \leq L_J(\boldsymbol{\theta}) d_\Theta(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}); \tag{2.71}$$

where

$$L_J(\boldsymbol{\theta}) := \frac{L_R L_\pi(\boldsymbol{\theta})}{(1-\gamma)\left(1 - \gamma L_P\left(1 + L_\pi \boldsymbol{\theta}\right)\right)}. \tag{2.72}$$

2.4.1 Lipschitz Continuity of the Policy Gradient

In order to consider the gradient of the return w.r.t. the policy parameterization, we consider the function $\eta^{\boldsymbol{\theta}}(s,a) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a) Q^{\boldsymbol{\theta}}(s,a)$. In particular, we consider the component-wise absolute difference between gradients corresponding to different parameterizations:

$$\left|\nabla_{\boldsymbol{\theta}_i} J_\mu^{\boldsymbol{\theta}} - \nabla_{\boldsymbol{\theta}_i} J_\mu^{\widehat{\boldsymbol{\theta}}}\right| = \frac{1}{1-\gamma}\left|\mathop{\mathbb{E}}_{(s,a)\sim\zeta_\mu^{\boldsymbol{\theta}}}\left[\boldsymbol{\eta}_i^{\boldsymbol{\theta}}(s,a)\right] - \mathop{\mathbb{E}}_{(s,a)\sim\zeta_\mu \boldsymbol{\theta}}\left[\boldsymbol{\eta}_i^{\widehat{\boldsymbol{\theta}}}(s,a)\right]\right|. \tag{2.73}$$

Proposition 2.4.3 (Lipschitz Bound on the policy gradient). *Under assumptions 2.4.1, 2.4.3 and 2.4.4, the i-th component of the gradient $\nabla_{\boldsymbol{\theta}} J_\mu^{\boldsymbol{\theta}}$ is $L_{\nabla J}^i(\boldsymbol{\theta})$-PLC, that is, $\forall(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) \in \Theta^2$:*

$$\left|\nabla_{\boldsymbol{\theta}_i} J_\mu^{\boldsymbol{\theta}} - \nabla_{\boldsymbol{\theta}_i} J_\mu^{\widehat{\boldsymbol{\theta}}}\right| \leq L_{\nabla J}^i(\boldsymbol{\theta}) d_\Theta(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}); \tag{2.74}$$

for the definition of $L_{\nabla J}$ formulated in (Pirotta et al., 2015).

Now, we consider a result showed by (X. Li and Orabona, 2019), in the framework of general optimization of a loss function f.

Proposition 2.4.4 (Progress bound in T steps). *Consider the optimization problem $\min_{x\in\mathbb{R}^d} f(x)$ with $f : \mathbb{R}^d \to \mathbb{R}$. Let's assume that:*

1. *f is M-smooth, meaning that f is differentiable and ∇f is M-LC*

$$\left\|\nabla f(x) - \nabla f(y)\right\| \leq M\left\|x - y\right\|; \tag{2.75}$$

2. *There exists an unbiased stochastic approximation of the gradient, i.e., there exists a function* $g(x, \xi)$ *such that* $\mathbb{E}_{\xi}[g(x, \xi)] = \nabla f(x)$, $\forall x \in \mathbb{R}^d$. *Then, the iterations of a SGD problem, where the update rule is* $x_{t+1} = x_t - \eta_t g(x_t, \xi_t)$ *satisfy the following inequality, where the step size* η_t *can be used in the form of a matrix in* $\mathbb{R}^{d \times d}$

$$\mathbb{E}\left[\sum_{t=1}^{T} \langle \nabla f(x_t), \eta_t \nabla f(x_t) \rangle\right] \leq f(x_1) - f^* + \frac{M}{2} \mathbb{E}\left[\sum_{t=1}^{T} \|\eta_t g(x_t, \xi_t)\|_2^2\right]. \quad (2.76)$$

Chapter 3

Meta Learning

3.1 Meta Learning

As suggested by the name, Meta Learning implies a raise in the level of abstraction with respect to regular machine learning. If the objective of machine learning is to automatically extract relevant information from data for a specific task, meta learning exploits the outputs of other learning algorithms as training samples, learning how to quickly learn new tasks and how to adapt to changes. Working a level above the usual model, Meta Learning can be applied to all fields of machine learning, with various approaches and different goals.

We now provide a general introduction to the argument in the context of Supervised Learning, to then focus on Meta Reinforcement Learning, the subject of this thesis. The following sections are inspired by Lilian Weng's great introduction to the topic (Weng, 2018; Weng, 2019b), as well as Jonathan Hui's blog (Hui, 2020), Sergey Levine's Berkeley Deep RL course (Levine, 2020) and Chelsea Finn's Stanford Deep Multi-Task and Meta Learning course (Finn, 2020).

3.1.1 Meta-Training and Meta-Testing

For our introduction, let's consider the problem of meta classification. In this context, a meta-learning process consists in a sequence of trainings over various classification tasks, with the objective of finding the model that optimizes the performance over their distribution. Each task τ_i is associated with a dataset of feature vectors and labels $\mathcal{D}_i = \{(\mathbf{x}_i, y_i)\}$. As usual, this dataset is split in two parts, used to train and test within the task.

The terms training and testing can create some confusion in the context of meta-learning, since these operations are performed at two levels of abstraction. For this

reason, we denote the portion of $\mathcal{D}_i$ dedicated to training as the support set S and the portion used for testing as the prediction set B. Each label in the dataset belongs to the set of labels $\mathcal{L}$. These datasets are combined to form the complete dataset $\mathcal{D}$.

The optimization process is similar to a normal classification problem: the objective is to find the model that minimizes a loss function $\mathcal{L}_{\boldsymbol{\theta}}(\mathcal{D})$ over the training dataset. The difference is that, in this case, one dataset $\mathcal{D}_i$ represents one data point at the meta level:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{D}_i \sim p(\mathcal{D})} \left[\mathcal{L}_{\boldsymbol{\theta}}(\mathcal{D}_i) \right]. \tag{3.1}$$

After the training phase, the performance of the model is assessed on unseen samples. In this context, this stage is denoted as meta-testing and consists in the evaluation of the results on a training task different from the ones seen before.

An example of meta classification is a K-shot N-class classification task, in which the meta-training is exposed to N classes with K samples each:

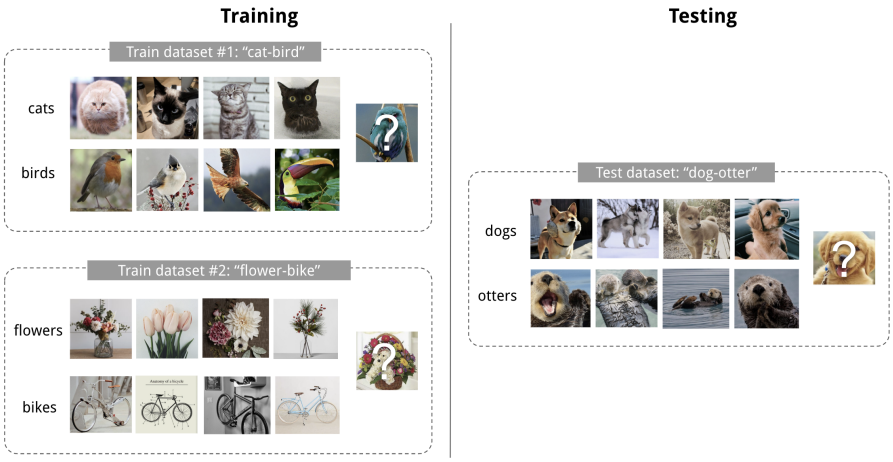


Figure 3.1: K-shot N-class Classification.
Source: Lilian Weng.

The objective defined above can also be expressed as a maximization problem. We denote the output of the classifier as $f_{\boldsymbol{\theta}} = P_{\boldsymbol{\theta}}(y|\mathbf{x})$, which is the a posteriori probability assigned to the label y, given the input vector $\mathbf{x}$. The optimal classifier is the one that returns the maximum a posteriori probability of correct labels over the dataset:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}} \left[P_{\boldsymbol{\theta}}(y|\mathbf{x}) \right]. \tag{3.2}$$

We can decompose a meta-learning optimization into two processes, working at different levels:

- at the lower level, a classifier $f_{\boldsymbol{\theta}}$, also known as the *learner* model, trains to optimize a specific task;
- at the upper level, an optimizer $g_{\boldsymbol{\phi}}$, also known as the *meta-learner* model, learns how to update the learner's parameters according to the support set S: $\boldsymbol{\theta}' = g_{\boldsymbol{\phi}}(\boldsymbol{\theta}, S)$. This level is responsible of discovering the general principles that characterize all tasks and of transferring them to unseen settings.

In the end, the objective involves optimizing both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, to obtain the following:

$$\mathbb{E}_{L \subset \mathcal{L}} \left[\mathbb{E}_{S^L \subset \mathcal{D}, B^L \subset \mathcal{D}} \left[\sum_{(\mathbf{x}, y) \in B^L} P_{g_{\boldsymbol{\phi}}(\boldsymbol{\theta}, S^L)}(y \mid \mathbf{x}) \right] \right]. \tag{3.3}$$

In this formalization with two learners, meta-learning is reduced to the construction of a meta-learner that produces learners adapted to specific tasks.

An example of a few-shot learning classifier is shown in the picture below. In this case, a Long Short Term Memory (LSTM) uses the support set to predict the vector $\boldsymbol{\theta}^*$, that is used as the parameterization of an Multi Layer Perceptron (MLP) network. This inner model is responsible for the predictions in the specific tasks. The LSTM meta-model, parameterized by $\boldsymbol{\phi}$, extracts the task context of $\mathcal{D}$ from the support and improves the learning capabilities of the subordinate model.

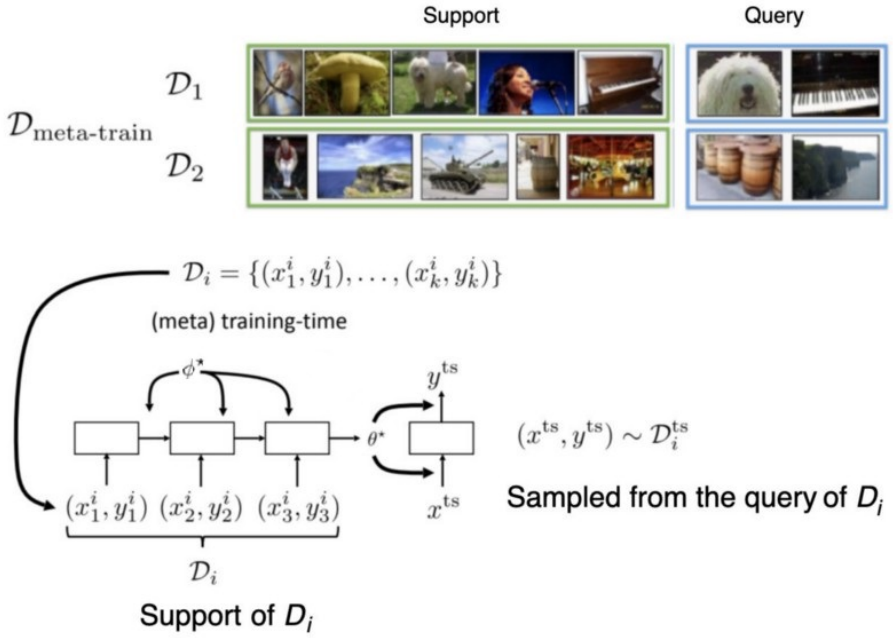


Figure 3.2: LSTM Meta Model.
Source: Jonathan Hui.

Most meta-learning approaches can be classified in one of the following categories:

- *metric-based*, they operate following the framework of metric learning;
- *model-based*, they build a memory of previous tasks to adapt the model parameters in various settings;
- *optimization-based*, they operate some form of Gradient Ascent optimization to the model parameters.

In general, all methods share the presence of an inner loop and an outer loop. The inner loop contains an *adaptation objective* that adapts the learner $\boldsymbol{\theta}_i$ to the given task τ_i. The outer loop aims to learn the meta-model $\boldsymbol{\phi}^*$ to produce the optimal learner $\boldsymbol{\theta}_i$.

3.1.2 Metric-Based Meta Learning

To understand the concept of metric-based learning, we need to define the distinction between parametric and non-parametric learning approaches. A parametric method aims to find the optimal model parameters which capture the information contained in a training set. Once the model is built, the data is no longer needed to make predictions, since all the knowledge extracted is contained in the model itself. On the other hand, non-parametric methods don't rely on training a model, but use directly the data to make inference, usually by exploring similarities and patterns in them.

Non-parametric algorithms completely avoid the temporal and computational costs of building a model, at the expense of higher complexities at inference time. Since the training data can't be discarded, these approaches can have some limitations when dealing with large datasets. Luckily, this is rarely the case in the context of meta-learning, where the support set is usually made of few samples.

Metric-based meta-learning is built upon similar concepts of classic non-parametric methods such as the k-nearest neighbors and kernel density estimation algorithms. The predicted probability over a set of labels y is given by a weighted sum of labels of support set samples:

$$P_{\boldsymbol{\theta}}(y \mid \mathbf{x}, S) = \sum_{(\mathbf{x}_i, y_i) \in S} k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}_i)\, y_i; \tag{3.4}$$

where the kernel function $k_{\boldsymbol{\theta}}$ measures the similarity between two data samples. Of course, the definition of a kernel that accurately represent the relations between data points is a problem-dependent task.

Metric-based meta learning can still use a parametric model to capture the knowledge of the meta-training dataset, usually through a feature extraction algorithm. An example of such approach is known as Convolutional Siamese Neural Networks (Koch et al., 2015), a technique based on the idea that in order to create an optimal classifier, we need to

learn how to discriminate between two classes. This is accomplished by using two twin networks, with exactly the same weights and model parameters.

During training, two samples are given as inputs to the twin networks, that output two vectors of extracted features. These features are then fed to a discriminator, that outputs the probability of the two samples belonging to the same class. During testing, the Siamese architecture compares all the test samples with the support set. The predicted class is the one with the highest probability.

3.1.3 Model-Based Meta Learning

Model-based meta-learning approaches aim to build models that can quickly incorporate new information, thus adapting to new tasks in the minimal number of steps. This goal can be reached directly by the model's architecture or by means of a meta-learner, that uses samples from the support set to adapt the parameters.

Memory Augmented Neural Networks

A common approach to enhance the generalization capabilities of a learner is to expand it with an external memory. Having the ability to load information into a data storage and to retrieve it at a later stage allows a model to quickly incorporate complex information and to use it as a base to build new knowledge.

In the context of neural networks, this kind of models are known as Memory-Augmented Neural Networks (MANN) and include Neural Turing Machines (Graves et al., 2014), among others. In this implementation of the framework, the external memory added to the network imitates the functioning of a Turing Machine tape. A NTM is formed by two elements: a memory that act as a knowledge repository and a neural network controller that learns to read and write the memory. A generic iteration works as follows: the controller receives and processes the input, interacts with the memory and returns an output. During the interaction the reading and writing operations happen in parallel. They are also performed in a "soft" manner, through a mechanism called Attention, that enhances the important parts of the data.

These architectures have been adapted to meta-learning (Santoro et al., 2016), with some changes to the training algorithm and to the memory addressing mechanism. In order to obtain fast learning of new tasks, it's required a quick encoding of new information and an easy and reliable access to the saved knowledge.

To ensure a long lasting storage of information in the memory, during training the true label y_t is always presented one episode after the corresponding input. In other words, at each training step t, the controller receives an input with the form $(\mathbf{x}_t, y_{t-1})$. This delay forces the network to hold information in memory at each step, thus improving the generalization capabilities of the model.

3.1.4 Optimization-Based Meta Learning

Among the many optimization techniques adopted in machine learning, Gradient Descent algorithms are certainly the most popular. In particular, they are widely adopted to train Deep Neural Networks of various forms. Unfortunately, they can suffer of some limitations:

- they usually require a large number of samples to reach good performances;
- they tend to converge after a large number of iterations.

The objective of optimization-based meta learning is the creation of an optimizer that can train a model using very few data samples and the minimum amount of steps. We now present an algorithm belonging to this class, known as Reptile. The description is carried with a higher level of details compared to previous methods, since the approach proposed in this thesis can be classified as optimization-based.

Reptile

Reptile is an optimization-based algorithm that attempts to solve meta-learning by finding an optimal initialization for the parameters of a model. Starting the optimization process from a favourable position can considerably reduce the training time, allowing the model to generalize from a small number of samples. This approach is very similar to another popular meta-learning algorithm called MAML, that will be described in section 3.2.1 in the context of RL.

Reptile is a model-agnostic algorithm, meaning that it can be applied to any learning problem and any model, as long as the training is done through a gradient descent procedure. For this reason Reptile can be used to solve classification, regression and Reinforcement Learning tasks.

The algorithm applies iteratively the following steps:

1. sample a task;
2. execute a series of gradient descent steps on the task;
3. update the model weights towards the new parameters.

Minimal distance from the manifolds of all tasks

It's reasonable to assume that each task τ in a meta-learning problem has a manifold of optimal parameter values $\mathcal{W}_\tau^*$, in simpler terms a region of the parameter space that yields the optimal performance. The closest the parameters $\boldsymbol{\theta}^*$ are to this region, the better the performance will be on that task. In order to find an initialization that maximizes the efficiency over the tasks distribution, Reptile optimizes $\boldsymbol{\theta}$ with the objective

of finding the value that minimizes the squared distance from the optimal manifolds of all tasks:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\tau \sim p(\tau)} \left[\frac{\operatorname{dist}\left(\boldsymbol{\theta}, \mathcal{W}_\tau^*\right)^2}{2} \right]. \tag{3.5}$$

Figure 3.3 shows an example of an iterative optimization that converges towards the point that minimizes the distance from two optimal manifolds $\mathcal{W}_1$ and $\mathcal{W}_2$.

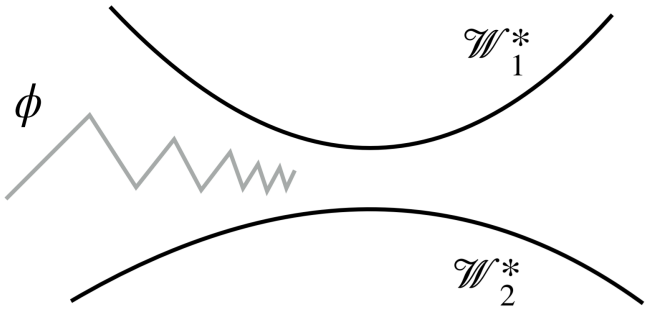


Figure 3.3: Minimization of the distance from the optimal manifolds.
Source: Nichol et al., 2018.

The distance between $\boldsymbol{\theta}$ and the set $\mathcal{W}_\tau^*$ is defined as the distance between $\boldsymbol{\theta}$ and its closest point on $W_\tau^*(\boldsymbol{\theta})$:

$$\operatorname{dist}\left(\boldsymbol{\theta}, \mathcal{W}_\tau^*\right) = \operatorname{dist}\left(\boldsymbol{\theta}, W_\tau^*(\boldsymbol{\theta})\right), \text{ where } W_\tau^*(\boldsymbol{\theta}) = \arg\min_{W \in \mathcal{W}_\tau^*} \operatorname{dist}(\boldsymbol{\theta}, W). \tag{3.6}$$

Finally, if we assume to use an Euclidean Distance, we can compute the gradient as:

$$\nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} \operatorname{dist}\left(\boldsymbol{\theta}, W_{\tau_i}^*(\boldsymbol{\theta})\right)^2 \right] = \nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} \left(\boldsymbol{\theta} - W_{\tau_i}^*(\boldsymbol{\theta})\right)^2 \right] = \boldsymbol{\theta} - W_{\tau_i}^*(\boldsymbol{\theta}); \tag{3.7}$$

yielding the following update rule:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} \operatorname{dist}\left(\boldsymbol{\theta}, \mathcal{W}_{\tau_i}^*\right)^2 \right] = \boldsymbol{\theta} - \alpha \left(\boldsymbol{\theta} - W_{\tau_i}^*(\boldsymbol{\theta})\right) = (1-\alpha)\boldsymbol{\theta} + \alpha W_{\tau_i}^*(\boldsymbol{\theta}). \tag{3.8}$$

Unfortunately, it's not possible to exactly compute the closest point to $\boldsymbol{\theta}$ on the optimal manifold $W_{\tau_i}^*(\boldsymbol{\theta})$. For this reason Reptile uses the $\operatorname{SGD}\left(\mathcal{L}_\tau, \boldsymbol{\theta}, k\right)$ as an approximate solution.

We now illustrate the complete Reptile algorithm in batch version, i.e. when multiple tasks are sampled at each interaction:

Algorithm 8 Reptile

procedure REPTILE
  Initialize $\boldsymbol{\theta}$
  for iteration $= 1, 2, \ldots$ **do**
    Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
    for $i = 1, 2, \ldots, n$ **do**
      Compute $W_i = \text{SGD}\left(L_{\tau_i}, \boldsymbol{\theta}, k\right)$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \frac{1}{n} \sum_{i=1}^{n} \left(W_i - \boldsymbol{\theta}\right)$

It's also possible to consider $(\boldsymbol{\theta} - W)/\alpha$ as a gradient, where α is the step size used in the SGD update, and plug it into a more sophisticated optimizer like Adam.

3.2 Meta Reinforcement Learning

Meta Reinforcement Learning (meta-RL) consists in applying meta learning techniques to RL problems, with the goal of creating a RL algorithm that can perform with speed and efficiency in different settings. As usual, the meta-RL process begins with an initial meta training phase in which the system is subject to a set of training tasks, followed by a meta-testing phase in which the model is evaluated against an unseen task. It's common practice to sample the training and testing tasks from the same distribution of problems $\mathcal{M}$. An RL task is of course represented by an MDP $M_i \in \mathcal{M}$. In Figure 3.4, we show a few examples of meta-RL scenarios. The first instance is a robot trained to accomplish various household chores and tested on an new one. The second is a "HalfCheetah" simulated environment trained to run in different fashions and tested on an unseen speed and direction.

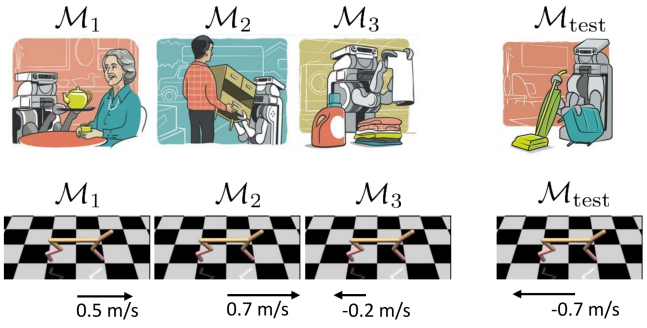


Figure 3.4: Examples of applications of meta-RL.
Source: Sergey Levine.

The meta-RL field is not new in general, but it gained most of its popularity in the recent years. In particular, the algorithms RL^2 (Duan et al., 2016) and Meta-RL (Wang et al., 2017) are considered the origin of the recent wave of interest for the topic.

The general idea behind these algorithms is to construct a memory of experiences from previous tasks in order to capture patterns that describe the correlations between state-action pairs and rewards. This general knowledge is then used to optimize the model's policy in an unseen MDP.

To provide the model with this memorization ability, the policy $\pi_{\boldsymbol{\theta}}(s_t)$ needs to be dependent also on the reward r_{t-1} and action a_{t-1} of the previous step, resulting in $\pi_{\boldsymbol{\theta}}(a_{t-1}, r_{t-1}, s_t)$. In practice, the algorithms RL^2 and Meta-RL adopt a slightly different approach, implementing the policies with a LSTM network. The LSTM hidden states are inherently suited to memorize information from the current task, avoiding the need to explicitly add inputs from previous steps.

A general meta-RL training process follows these steps:

1. sample a random MDP $M_i \sim \mathcal{M}$;
2. reset the hidden state of the model;
3. sample a set of trajectories and update the model weights according to the current MDP.

In the cases of Meta-RL and RL^2, the update is performed through an ordinary LSTM gradient descent step.

3.2.1 Meta-RL State of the Art

In this section, we describe a few state of the art meta-RL algorithms that inspired our work. Due to the generality of the problem, different approaches focus on various aspects of the optimization, such as the model weights, the hyperparameters, the loss function or the exploration strategies.

Apart from their differences, these approaches generally share the two levels architecture seen on most meta-learning algorithms. The outer level consists in an optimization loop that samples a new task at every step and adapts the model to the current setting. The inner level contains the actual RL algorithm that optimizes the interaction between the agent and the environment to obtain the maximal cumulative reward.

Another aspect in common between these methods is that they all introduce an inductive bias, i.e. the set of assumptions used by the learner to predict outputs of unseen inputs, an ability that generally lacks in RL algorithms (Botvinick et al., 2019). As a general ML rule, a learning algorithm with weak inductive bias is able to master a wider range of variance, while being usually less sample-efficient. Depending on the

meta-RL approach, certain inductive biases from the task distribution are included and stored in memory, improving the learning in a particular dimension.

MAML

Model-Agnostic Meta-Learning (Finn et al., 2017), also known as MAML, is one of the most popular meta-learning algorithms of the recent years. As the name suggests, it's a model agnostic optimization algorithm, meaning that it's applicable to any learning model trainable with a gradient descent approach.

MAML is an optimization-based method that aims to find the optimal model initialization for a specific task. It's very similar to Reptile, another algorithm introduced in section 3.1.4. Due to its generality, MAML can be used in a variety of machine learning problems, including classification, regression, and reinforcement learning. In this section, we focus on the application of MAML to RL tasks.

As usual, we consider a model $f_{\boldsymbol{\theta}}$ parameterized by the set of parameters$\boldsymbol{\theta}$. In the RL context, $f_{\boldsymbol{\theta}}$ is a policy that associates a state $\mathbf{s}_t$ to a distribution over actions a_t at each timestep $t \in \{1, \dots, H\}$. The model acts in a set of tasks $p(\mathcal{T})$ represented by Markov decision processes. Each task τ_i is defined by an initial state distribution $\mathbf{s}_0 \sim q_i$ and a transition distribution $s_{t+1} \sim q_i(\cdot \mid s_t, a_t)$.

Given a specific task τ_i, a gradient descent update of the model's parameters has the following form:

$$\boldsymbol{\theta}_i' = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\tau_i} \left(f_{\boldsymbol{\theta}} \right); \tag{3.9}$$

where $\mathcal{L}_{\mathcal{T}_i}$ is the negative reward function R:

$$\mathcal{L}_{\tau_i} \left(f_{\boldsymbol{\theta}} \right) = -\mathbb{E}_{s_t, a_t \sim f_\phi, q_{\tau_i}} \left[\sum_{t=1}^{H} R_i \left(s_t, a_t \right) \right]. \tag{3.10}$$

The meta-learning objective is to maximize the performance over a distribution of tasks, or equivalently to minimize the expected loss over a training data batch:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)} \left(f_{\boldsymbol{\theta}_i'} \right) = \arg\min_{\boldsymbol{\theta}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)} \left(f_{\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\tau_i}^{(0)} (f_{\boldsymbol{\theta}})} \right). \tag{3.11}$$

The superscripts (0) and (1) indicate that the same loss $\mathcal{L}_{\tau_i}$ is measured in different data batches at the two levels of the optimization. This objective can be formulated as a gradient descent update of the following form:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)} \left(f_{\boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\tau_i}^{(0)} (f_{\boldsymbol{\theta}})} \right). \tag{3.12}$$

The following is a visualization of a MAML gradient step:

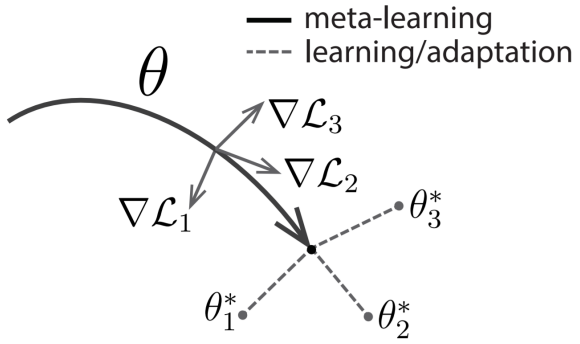


Figure 3.5: MAML Gradient Update.
Source: Finn et al., 2017.

The complete MAML algorithm for RL is outlined in Algorithm 9:

Algorithm 9 MAML for RL

Input: distribution over tasks $p(\mathcal{T})$, step size hyperparameters α, β

procedure MAML($p(\mathcal{T}), \alpha, \beta$)
  randomly initialize $\boldsymbol{\theta}$
  while not done **do**
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ **do**
      Sample K trajectories $\mathcal{D} = \{(s_1, a_1, \dots s_H)\}$ using $f_{\boldsymbol{\theta}}$ in $\mathcal{T}_i$
      Evaluate $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$
      Compute adapted parameters with gradient descent:
      $\boldsymbol{\theta}_i' = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})$
      Sample trajectories $\mathcal{D}_i' = \{(s_1, a_1, \dots s_H)\}$ using $f_{\boldsymbol{\theta}_i'}$ in $\mathcal{T}_i$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\boldsymbol{\theta}_i'}\right)$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$

Let's now examine the MAML optimization step. Starting from an initial model parameter $\boldsymbol{\theta}_{\text{meta}}$, we consider an episode of k gradient descent updates on a given task:

$$\begin{aligned} \boldsymbol{\theta}_0 &= \boldsymbol{\theta}_{\text{meta}} \\ &\dots \\ \boldsymbol{\theta}_k &= \boldsymbol{\theta}_{k-1} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}^{(0)}(\boldsymbol{\theta}_{k-1}). \end{aligned} \tag{3.13}$$

At the end of the episode, a new batch is sampled and the meta objective is updated as follows:

$$\boldsymbol{\theta}'_{\text{meta}} \leftarrow \boldsymbol{\theta}_{\text{meta}} - \beta g_{\text{MAML}}; \tag{3.14}$$

where

$$
\begin{aligned}
g_{\text{MAML}} &= \nabla_{\boldsymbol{\theta}_{\text{meta}}} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right) \\
&= \nabla_{\boldsymbol{\theta}_k} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right) \cdot \left(\nabla_{\boldsymbol{\theta}_{k-1}} \boldsymbol{\theta}_k\right) \ldots \left(\nabla_{\boldsymbol{\theta}_0} \boldsymbol{\theta}_1\right) \quad \text{(chain rule)} \\
&= \nabla_{\boldsymbol{\theta}_k} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right) \cdot \left(\prod_{i=1}^{k} \nabla_{\boldsymbol{\theta}_{i-1}} \boldsymbol{\theta}_i\right) \\
&= \nabla_{\boldsymbol{\theta}_k} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right) \cdot \prod_{i=1}^{k} \nabla_{\boldsymbol{\theta}_{i-1}} \left(\boldsymbol{\theta}_{i-1} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}^{(0)}\left(\boldsymbol{\theta}_{i-1}\right)\right) \\
&= \nabla_{\boldsymbol{\theta}_k} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right) \cdot \prod_{i=1}^{k} \left(I - \alpha \nabla_{\boldsymbol{\theta}_{i-1}} \left(\nabla_{\boldsymbol{\theta}} \mathcal{L}^{(0)}\left(\boldsymbol{\theta}_{i-1}\right)\right)\right).
\end{aligned}
\tag{3.15}
$$

As we can see, the gradient descent step in the meta level is a second-order optimization. The additional information about the curvature allows MAML to better estimate the update direction, but also increases significantly the complexity of the computations.

To reduce these drawbacks, a simplified approach is to ignore the second-order derivative in the gradient. This method, known as First-Order MAML (FOMAML), has the following update rule:

$$
g_{\text{FOMAML}} = \nabla_{\boldsymbol{\theta}_k} \mathcal{L}^{(1)}\left(\boldsymbol{\theta}_k\right). \tag{3.16}
$$

One may notice the similarity between the FOMAML and Reptile updates. It can be demonstrated that the two algorithms optimize two very comparable objective functions.

Meta-Gradient RL

In general, a reinforcement learning algorithm maximizes a given value function to find an optimal policy. Differently from supervised learning problems, that provide the true value function as an input, most reinforcement learning algorithms have to optimize an estimate of the function.

Meta-Gradient RL (Xu et al., 2018) is a technique that learns a value function G_t defined by a set of tunable hyperparameters η, such as the discount factor γ and the bootstrapping parameter λ:

- n-step return function, $G_\eta^{(n)}\left(\tau_t\right) = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_{\boldsymbol{\theta}}\left(s_{t+n}\right)$;
- mixture of n-step return functions, $G_\eta^{(\lambda)}\left(\tau_t\right) = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_\eta^{(n)}$.

The meta parameter η is adjusted online by interacting with the environment, to create a value function that can quickly adapt to different settings and be optimized for each specific task. Given a model parameterized by $\boldsymbol{\theta}$, the optimization process is based on the following update rule:

$$
\boldsymbol{\theta}' = \boldsymbol{\theta} + f(\tau, \boldsymbol{\theta}, \eta); \tag{3.17}
$$

where the gradient $f(\tau, \boldsymbol{\theta}, \eta)$ used to adjust the policy is a function of a trajectory of experiences $\tau_t = \{s_t, a_t, r_{t+1}, \ldots\}$, the model weights $\boldsymbol{\theta}$ and the meta-parameters η. Meta-Gradient RL is based on the principle of *online cross-validation*, using a sequence of sample experiences: the RL algorithm is applied to one sample and evaluated in the consecutive sample. In particular, given a meta-objective function $J(\tau, \boldsymbol{\theta}, \eta)$, the algorithm works as follows:

1. the policy $\pi_{\boldsymbol{\theta}}$ is updated on the first batch of samples τ, producing the new parameter $\boldsymbol{\theta}'$;

2. the new policy $\pi_{\boldsymbol{\theta}'}$ is then applied to a new set of experiences τ', consecutive in time to τ. Its performance is evaluated as $J(\tau', \boldsymbol{\theta}', \bar{\eta})$ using a fixed meta-parameter $\bar{\eta}$;

3. the value of η is updated using the gradient of the meta-objective $J(\tau', \boldsymbol{\theta}', \bar{\eta})$ with respect to η:
$$\Delta\eta = -\beta \frac{\partial J(\tau', \boldsymbol{\theta}', \bar{\eta})}{\partial \eta} = -\beta \frac{\partial J(\tau', \boldsymbol{\theta}', \bar{\eta})}{\partial \boldsymbol{\theta}'} \frac{d\boldsymbol{\theta}'}{d\eta}; \tag{3.18}$$
where β is the learning rate.

The gradient of the update $\frac{\mathrm{d}\boldsymbol{\theta}'}{\mathrm{d}\eta}$ can be further manipulated using the multivariate chain rule:
$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{\theta}'}{\mathrm{d}\eta} &= \frac{\partial(\boldsymbol{\theta} + f(\tau, \boldsymbol{\theta}, \eta))}{\partial \eta} \\ &= \left(\frac{d\boldsymbol{\theta}}{d\eta} + \frac{\partial f(\tau, \boldsymbol{\theta}, \eta)}{\partial \boldsymbol{\theta}} \frac{d\boldsymbol{\theta}}{d\eta} + \frac{\partial f(\tau, \boldsymbol{\theta}, \eta)}{\partial \eta} \frac{d\eta}{d\eta} \right) \\ &= \left(\left(\mathbf{I} + \frac{\partial f(\tau, \boldsymbol{\theta}, \eta)}{\partial \boldsymbol{\theta}} \right) \frac{d\boldsymbol{\theta}}{d\eta} + \frac{\partial f(\tau, \boldsymbol{\theta}, \eta)}{\partial \eta} \right). \end{aligned} \tag{3.19}$$
As we can see, this gradient contains a second-order derivative, making it a challenging expression to compute. In practice, Meta-Gradient RL simplifies the update by discarding the secondary gradient, resulting in the final expression:
$$\Delta\eta = -\beta \frac{\partial J(\tau', \boldsymbol{\theta}', \bar{\eta})}{\partial \boldsymbol{\theta}'} \frac{\partial f(\tau, \boldsymbol{\theta}, \eta)}{\partial \eta}. \tag{3.20}$$

3.2.2 Distribution of Training Tasks

We conclude the chapter by introducing a simple technique to build a distribution of tasks. As usual for machine learning problems, an effective construction of the training data is essential for the success of a meta-RL process, sometimes even more than the actual learning algorithm.

Given a task formulated as a MDP $M_i = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle \in \mathcal{M}$, it's possible to generate a distribution by altering two features:

- the reward function R, to yield different returns in response to similar actions in different tasks;
- the transition function P, to produce different environment dynamics for different tasks.

The most straightforward technique to transform these parameters is known as Uniform Domain Randomization (Tobin et al., 2017; Sadeghi and Levine, 2017). As the name suggests, it consists in performing a uniform sampling for each environment parameter ξ_i to be randomized. The sampled values are bounded by an interval $\xi_i \in \left[\xi_i^{\text{low}}, \xi_i^{\text{high}}\right]$ for each $i = 1, \ldots, n$.

Domain Randomization allows to manipulate many aspects of a task. For example, if we consider a generic control problem in a physical setting, we could modify:

- properties of objects, such as mass, position, dimension and shape;
- physical dynamic features, such as friction and gravity constants;
- observation noises and action delays.

Of course, modifying the environment parameters to generate a distribution is only feasible in simulations, where we control the complete dynamic of the system. Obtaining the same task variety in real world scenarios is much more challenging, because the collection of sample trajectories is an expensive and time consuming procedure.

For this reason, Domain Randomization is also used to solve one of the greatest issues of RL applied to Robotics, known as the *sim2real gap*. This is the struggle to transfer results obtained in simulated environment to the physical world, due to inconsistencies between parameters or overly simplified physical modelings.

Domain Randomization reduces the gap by simulating a set of environments with randomized properties. If the number of simulations is large enough, the model will experience enough variations during training to be able to maintain its performance also in the real-world, that is one sample of the task distribution.

If interested in more advanced approaches to Domain Randomization, Lilian Weng's blog provides an excellent survey on the topic (Weng, 2019a).

Chapter 4

Proposed Solution

In this chapter, we present the theoretical and algorithmic contributes of this thesis. We begin by providing the definition of Contextual Markov Decision Process (CMDP), an extension of the classic MDP that groups a set of heterogeneous tasks in a single entity. Then, we introduce a framework to solve meta-RL problems, called meta-MDP. This model includes a CMDP, coupled with additional elements used for the optimization, i.e. the meta space, the meta action, the meta reward function and the learning function.

After deriving some guarantees about the loss function of meta-MDPs under the Lipschitz continuity property, we propose an algorithm to optimize the model when the learner is a PG method. Our approach is focused on learning an adaptive the step size for Gradient Ascent updates, to select the most adequate value of this parameter for each scenario. We describe the details of our algorithm, highlighting its motivations, goals and major strengths.

4.1 Contextual MDP

A Contextual Markov Decision Process (CMDP), as defined by (Hallak et al., 2015), is a framework to describe an environment where the state transitions and rewards depend on a hidden parameterization, known as the *context*. The objective of a CMDP is to learn a policy that maximizes the cumulated reward over all contexts.

Given its structure, a CMDP is a natural candidate to represent the set of tasks $\mathscr{M} = \{\mathcal{M}_i\}_{i \in \mathcal{I}}$ of a meta-RL problems. The formal definition is expressed as follows:

Definition 4.1.1 (Contextual Markov Decision Process). *A Contextual Markov Decision Process is a tuple* $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}(c))$ *where* $\mathcal{C}$ *is called the context space,* $\mathcal{S}$ *and* $\mathcal{A}$ *are the state and action space correspondingly, and* $\mathcal{M}$ *is function mapping any context* $c \in \mathcal{C}$ *to an MDP, such that* $\mathcal{M}(c) = \langle \mathcal{S}, \mathcal{A}, P^c, R^c, \gamma^c, \mu^c \rangle$.

In other words, a CMDP enclodes in a single entity a group of tasks that share the same state and action space. These tasks are defined by a set of variables, called the *context*. In the simplest case, the context of a CMDP is directly observable and the problem reduces to adapting the model to the current setting. Furthermore, if the context is finite, $|\mathcal{C}| = K$, to solve a CMDP it's sufficient to learn K different models.

Let's assume a finite CMDP, with a context space such that $|\mathcal{C}| = K$ and a sequence of H episodes of lenght T. Each episode starts with different contexts and initial states. The goal is to maximize the cumulative rewards collected over the episodes, while maintaining a sufficient exploration of the policy parameters and hidden context. In this context, the performance of the model can expressed as a regret, framing the problem as a Regret Minimization.

Definition 4.1.2. *The regret of H episodes for a finite CMDP is*

$$Regret = \sum_{h=1}^{H} J_h^* - \sum_{h=1}^{H} \sum_{t=1}^{T_h} r_{h,t}; \tag{4.1}$$

where, at each episode h, J_h^* is the optimal value function and $r_{h,t}$ is the reward obtained by the agent at step t.

To solve the problem, (Hallak et al., 2015) proposed a framework denoted Cluster-Explore-Classify-Exploit (CECE), that divides the episodes in minibatches and applies the following steps:

1. *cluster*, all previously observed trajectories are used to create K different models (one for each possible context);
2. *explore*, for each trajectory in the current mini-batch the agent generates a partial trajectory;
3. *classify*, the partial trajectory is classified to a context;
4. *exploit*, the agent adapts the policy to the identified context for the remainder of the episode.

As can be noted, the resolution of the objective above assumes the knowledge of the optimal value function for each context. This requirement is satisfiable for finite CMDPs, where states and contexts are limited, but isn't for infinite or continuous cases. In addition, the performance of the CECE approach is based on the accuracy of the clustering phase, that becomes hard to evaluate for large contexts.

Due to these limits, it is clear that CMDPs are not expressive enough to solve real world meta-RL problems. In the next section, we propose a framework that is more general and does not require this level of knowledge of the environment, called meta-MDP.

4.2 Meta-MDP

We now present the concept of meta-MDP, a framework to solve meta-RL tasks that extends the CMDP model and attempts to overcome its limitations. Similar approaches to the one proposed in this section can be found in (Garcia and Thomas, 2019) and in (K. Li and Malik, 2016).

To start, let's consider the various tasks used in a meta-training procedure as a set of MDPs $\mathscr{M} = \{\mathcal{M}_i\}_{i \in \mathcal{I}}$, such that each task $\mathcal{M}_i$ can be sampled from the distribution p inducted by the set with probability $p_i = P(\mathcal{M}_i)$. Similarly, the parameters vector $\boldsymbol{\theta}$ induces a distribution q over the policy space $\boldsymbol{\theta}$. At each iteration in an MPD $\mathcal{M}_i$, the policy parameters are initialized to a value drawn from the distribution ($\boldsymbol{\theta}_0 \sim q(\boldsymbol{\theta})$). As pointed out in the previous section, the set of tasks can be formulated as a CMDP, where each $\mathcal{M}_i \in \mathscr{M}$ is represented by a function $\phi_{\mathcal{M}}$ determined by the context. In our case, we consider it to be the parameterized context itself, i.e. $\phi_{\mathcal{M}}(\mathcal{M}_i) = c$.

Definition 4.2.1 (Meta-MDP). *A meta-MDP is a tuple $\langle \mathcal{X}, \mathcal{H}, \mathcal{L}, \widetilde{\gamma}, (\mathscr{M}, p), (\boldsymbol{\theta}, q), f \rangle$, where:*

- *$\mathcal{X}$ and $\mathcal{H}$ are respectively the meta state space and the learning action;*
- *$\mathcal{L}$ is the meta reward function;*
- *$\widetilde{\gamma}$ is the meta-discount factor;*
- *$(\mathscr{M}, p)$ and $(\boldsymbol{\theta}, q)$ contain respectively the set of tasks (MDPs) and the policy space, with their related distributions;*
- *$f : \boldsymbol{\theta} \times \mathcal{H} \to \boldsymbol{\theta}$ is the update rule of the learning model chosen.*

In particular, a meta-MDP attempts to enclose the general elements necessary to learn a RL task into a model with similar properties to a classic MDP. Of course, some aspects differ from the usual framework, since a meta-MDP is a strict extension of it. Once the definition is given, we can try to describe the properties of the model.

The meta state $\mathcal{X}$ of a meta-MDP can be considered as the generalization of the observation space $\mathcal{S}$ in classic MDPs. Its initial distribution depends on the joint distribution over the policy space and the set $\mathscr{M}$. This means that each meta-episode is based on a different initial policy and a different MDP. Of course, one could decide to perform a batch of trajectories with the same initialization for every task.

Each action $h \in \mathcal{H}$ performed on the meta-MDP determines a specific hyperparameter that regulates the update rule f, such that, in each episode of the current task we have $\boldsymbol{\theta}_{k+1} = f(\boldsymbol{\theta}_k, h_k)$. In particular, this thesis focuses on Stochastic Gradient Ascent (SGA) learning approaches, in which the action h determines the step size α, and the

update rule takes the form $f(\boldsymbol{\theta}, h) = \boldsymbol{\theta} + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$, where J_i is the expected return observed in a specific task $\mathcal{M}_i$.

Similarly to a standard RL problem, the training of a meta-MDP is accomplished by optimizing a reward function. In order to accelerate the learning over the current MDP $\mathcal{M}_i$, this function should reflect variations between the returns obtained in different learning steps. To accomplish this, we define $\mathcal{L}(x, \boldsymbol{\theta}, h)$ as:

$$\mathcal{L}(x, \boldsymbol{\theta}, h) := J_i(f(\boldsymbol{\theta}, h)) - J_i(\boldsymbol{\theta}); \tag{4.2}$$

where $J_i(\boldsymbol{\theta})$ and $J_i(f(\boldsymbol{\theta}, h))$ are respectively the expected returns in the sub-task $\mathcal{M}_i$ before and after one update step according to the function f.

Hence, the meta reward function is equivalent to the gain in terms of return obtained after a meta-learning update step. In the particular case of SGA, the function assumes the following form:

$$\mathcal{L}(x, \boldsymbol{\theta}, h) = J_i\left(\boldsymbol{\theta} + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})\right) - J_i(\boldsymbol{\theta}). \tag{4.3}$$

Another aspect observable from Definition 4.2.1 is that, differently from a standard MDP, a meta-MDP does not include a Markovian transition model that regulates its dynamics. The state space $\mathcal{X}$ of a meta-MDP can contain information about the state of the inner task $\mathcal{M}_i \in \mathscr{M}$, and the state of the learning model, as for example the policy parameters $\boldsymbol{\theta}$, the current performance $J_i(\boldsymbol{\theta})$ or its gradient $\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$. Consequently, given the current state x_k and the parameters $\boldsymbol{\theta}_k$ in a specific MDP $\mathcal{M}_i$, the transition to the next state x_{k+1} and to $\boldsymbol{\theta}_{k+1}$ is, of course, stochastic, but depends only on $\mathcal{M}_i\left(\mu, \mathcal{P}, \pi_{\boldsymbol{\theta}}\right)$ and on the update rule f. For this reason, we do not consider a meta-transition Kernel, but we define the internal MDP dynamics as $\delta(\cdot \mid \boldsymbol{\theta}, \mathcal{M}_i)$, such that $x \sim \delta(\cdot \mid \boldsymbol{\theta}, \mathcal{M}_i)$. The initial distribution and transition probability can be expressed in the following manner:

$$\begin{aligned} x_0 \sim d_i^0\left(\boldsymbol{\theta}_0\right) &:= d\left(\cdot \mid \boldsymbol{\theta}_0, \mathcal{M}_i\right) \\ \left(x_{k+1}, \boldsymbol{\theta}_{k+1}\right) \sim \delta_i\left(\cdot \mid \left(x_k, \boldsymbol{\theta}_k\right), h_k\right) &:= \delta\left(\cdot \mid f\left(\boldsymbol{\theta}_k, h_k\right), \mathcal{M}_i\right). \end{aligned} \tag{4.4}$$

The meta state space $\mathcal{X}$ can be implemented in various forms, according to the information we want to include in it. There are, however, some properties that are generally desirable in a meta state space:

- Markov property: the meta state captures the entire information about the environment history until the current moment;
- Policy-specific information: some form of knowledge about the current policy is necessary to adapt the meta actions to the current setting of the model;
- Task-specific information: a crucial aspect to obtain good results over a distribution of MDPs. The information about the task is used to achieve an *implicit task-identification*, a necessary step to optimize the learning process with respect to a new task, based on similarities with older tasks.

Some examples of meta states $x \in \mathcal{X}$ that can be defined are:

- $\langle \boldsymbol{\theta}_t, \phi_{\mathcal{M}}(\mathcal{M}_t) \rangle$, it's the most straightforward implementation of the definition of meta state. It consists of the current policy parameters and the current representation of the task $\mathcal{M}_t$;
- $\langle \boldsymbol{\theta}_t, \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_t) \rangle$, this version attempts to embed task information in the meta state using the gradient of the current policy parameters $\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_t)$, under the assumption that its value changes depending on the specific task $\mathcal{M}_t$. The implicit information about the task $\mathcal{M}_t$ contained in the task is stochastic and dependent on the trajectories used to generate it. This stochasticity could be reduced by averaging over multiple trajectories, producing a more reliable estimate of the true gradient. If instead we choose to not average it, we will renounce to some accuracy in favor of additional (implicit) information about the particular trajectory used for the estimate (different trajectories lead to different gradients);
- $\langle \boldsymbol{\theta}_t, \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_t), \phi_{\mathcal{M}}(\mathcal{M}_t) \rangle$, a combination of the previous two examples. It's the solution adopted in most experimental results illustrated in the following chapter of this thesis.

The dynamics of $\mathcal{X}$, defined above as $x \sim d(\cdot \mid \boldsymbol{\theta}, \mathcal{M})$, assumes different forms based on the the meta state space selected. For example, the case $\langle \boldsymbol{\theta}_t, \phi_{\mathcal{M}}(\mathcal{M}_t) \rangle$ produces the following meta state transition:

$$x_{t+1} = \langle \boldsymbol{\theta}_t + h_t \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t, \tau_t), \phi_{\mathcal{M}}(\mathcal{M}_t) \rangle. \tag{4.5}$$

To conclude this section, we point out some adjustments to make in the construction of the meta state, according to the adopted training process:

- On-Policy learning: in the case the parameters $\boldsymbol{\theta}_t$ are the ones used by the policy to generate the trajectories, batch information about the trajectory (such as averaged gradient or entropy) should be directly related to the task;
- Off-Policy learning: in the case the parameters $\boldsymbol{\theta}_t$ are not used to generate the trajectories, the gradient does not provide information about the task if coupled only with the policy parameters. An option could be to add a function $\phi_{\pi\pi'}$ of the importance sampling weight $\frac{\pi_{\boldsymbol{\theta}}}{\pi'}$ to the meta-state.

4.3 Lipschitz Meta-MDP

In this section, we consider a meta-MDP in which all inner tasks satisfy the Lipschitz continuity assumption. Under this condition, we are able to derive a set of guarantees on the performance of the model. Among others, we obtain boundaries for two crucial meta-MDP elements, i.e. the meta return function $J(\boldsymbol{\theta})$ and its gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, providing theoretical foundations for their use in our approach.

To start, let's suppose that each task $\mathcal{M} \in \mathscr{M}$ can be parameterized through parameters $\omega \in \Omega$. This task $\mathcal{M}_\omega$ can be expressed as a tuple $\langle \mathcal{S}, \mathcal{A}, P_\omega, R_\omega, \gamma, \mu \rangle$, in such a way that Assumption 2.4.1 is verified:

$$\begin{aligned} \mathcal{K}\left(P_\omega(\cdot \mid s,a), P_\omega(\cdot \mid \bar{s},\bar{a})\right) &\leq L_P(\omega) d_{\mathcal{S}\times\mathcal{A}}((s,a),(\bar{s},\bar{a})) \\ \left|R_\omega(s,a) - R_\omega(\bar{s},\bar{a})\right| &\leq L_r(\omega) d_{\mathcal{S}\times\mathcal{A}}((s,a),(\bar{s},\bar{a})). \end{aligned} \tag{4.6}$$

Let's also suppose that Assumption 2.4.2 is verified and that the set of MDPs is Lipschitz continuous w.r.t the parameterization of the task:

Assumption 4.3.1 (Lipschitz Task Conditions).

$$\begin{aligned} \mathcal{K}\left(P_\omega(\cdot \mid s,a), P_{\widehat{\omega}}(\cdot \mid s,a))\right) &\leq L_{\omega_P} d_\Omega(\omega,\widehat{\omega}) \\ \left|R_\omega(s,a) - R_{\widehat{\omega}}(s,a)\right| &\leq L_{\omega_r} d_\Omega(\omega,\widehat{\omega}); \end{aligned} \tag{4.7}$$

This means that we have some notion of continuity w.r.t. the task: when two MDPs with similar contexts are considered, then their transition and reward processes cannot be completely different.

These assumptions allow us to make some considerations regarding the inference on the Q-value function:

Proposition 4.3.1 (Lipschitz Q_ω). *Given a set of parameterized tasks $\mathscr{M} = \{\mathcal{M}_\omega\}_{\omega\in\Omega}$ for which Assumption 4.3.1 holds, and such that, $\mathcal{M}_\omega$ is $(L_P(\omega), L_r(\omega))$-LC $\forall \omega \in \Omega$, given a L_π-LC policy π, the action value function $Q^\pi_\omega(s,a)$ is L_{ω_Q}-LC w.r.t. the task ω, i.e.:*

$$\left|Q^\pi_\omega(s,a) - Q^\pi_{\widehat{\omega}}(s,a)\right| \leq L_{\omega_Q}(\pi) d_\Omega(\omega,\widehat{\omega}); \tag{4.8}$$

where

$$L_{\omega_Q}(\pi) = \frac{L_{\omega_r} + \gamma L_{\omega_p} L_{V^\pi}(\omega)}{1-\gamma}. \tag{4.9}$$

and $L_{V^\pi}(\omega)$ is the Lipschitz constant of the value function V^π_ω related to equation 2.70.

Given this result, the consequence on the return function is immediate:

$$\begin{aligned}\left|J_{\omega}(\pi)-J_{\widehat{\omega}}(\pi)\right| &= \left|\int_{\mathcal{S}}\mu\left(s_0\right)\left[V_{\omega}^{\pi}\left(s_0\right)-V_{\widehat{\omega}}^{\pi}\left(s_0\right)\right]ds_0\right| \\ &\leq \int_{\mathcal{S}\times\mathcal{A}}\mu\left(s_0\right)\pi\left(a\mid s_0\right)\left|Q_{\omega}^{\pi}\left(s_0,a\right)-Q_{\widehat{\omega}}^{\pi}\left(s_0,a\right)\right|dads_0 \\ &\leq L_{\omega_Q}(\pi)d_{\Omega}(\omega,\widehat{\omega}).\end{aligned} \tag{4.10}$$

In simpler terms, Proposition 4.3.1 exploits the LC property to derive an upper bound on the distance between the Q_{ω}^{π} functions of two tasks $\omega_1, \omega_2 \in \Omega$ belonging to the same set $\mathcal{M}$. This result represents an important guarantee on the generalization capabilities of the approach. Indeed, if we think of ω_1 as a task seen during the training phase of a generic meta-RL algorithm, and of ω_2 as another task encountered during testing, this property limits the distance between the expected returns $J_{\omega_1}(\pi)$ and $J_{\omega_2}(\pi)$ achieved by the same policy π in the two tasks. As a consequence, it provides a boundary on the error obtained in testing by making inference on a Q function based on the training tasks. The following is a proof of the proposition:

Proof. We follow the same ideas as in (Rachelson and Lagoudakis, 2010): first of all, given an L_{ω_Q}-LC continuous Q function Q^{π} w.r.t. the task space Ω, the related value function V_{ω}^{π} is L_{ω_Q}-LC. Indeed,

$$\begin{aligned}\left|V_{\omega}^{\pi}(s)-V_{\widehat{\omega}}^{\pi}(s)\right| &= \left|\int_{\mathcal{A}}\pi(a\mid s)\left(Q_{\omega}^{\pi}(s,a)-Q_{\widehat{\omega}}^{\pi}(s,a)\right)da\right| \\ &\leq \int_{\mathcal{A}}\pi(a\mid s)\left|Q_{\omega}^{\pi}(s,a)-Q_{\widehat{\omega}}^{\pi}(s,a)\right|da \\ &\leq \max_{a}\left|Q_{\omega}^{\pi}(s,a)-Q_{\widehat{\omega}}^{\pi}(s,a)\right| \leq L_{\omega_Q}d_{\Omega}(\omega,\widehat{\omega}).\end{aligned} \tag{4.11}$$

Now, we prove that Q_n^{π} is $L_{\omega_Q}^n$-LC continuous, and that satisfies the recurrence relation:

$$L_{\omega_Q}^{n+1} = L_{\omega_r} + \gamma L_{\pi} L_V(\omega) + \gamma L_{\omega_Q}^n. \tag{4.12}$$

Indeed, for $n = 1$ the property holds immediately, since:

$$\left|Q_{\omega}^{\pi,1}(s,a)-Q_{\widehat{\omega}}^{\pi,1}(s,a)\right| = \left|R_{\omega}(s,a)-R_{\widehat{\omega}}(s,a)\right| \leq L_{\omega_r}d_{\Omega}(\omega,\widehat{\omega}). \tag{4.13}$$

Now, let us suppose the property holds for n. Then:

$$\begin{aligned}\left|Q_{\omega}^{\pi,n+1}(s,a)-Q_{\widehat{\omega}}^{\pi,n+1}(s,a)\right| = \Big|&R_{\omega}(s,a)-R_{\widehat{\omega}}(s,a) \\ &+\gamma\int_{\mathcal{S}}P_{\omega}\left(s'\mid s,a\right)V_{\omega}^{\pi,n}\left(s'\right)ds' \\ &-\gamma\int_{\mathcal{S}}P_{\widehat{\omega}}\left(s'\mid s,a\right)V_{\widehat{\omega}}^{\pi}\left(s'\right)ds'\Big|\end{aligned}$$

$$
\begin{aligned}
&\leq L_{\omega_r} d_\Omega(\omega, \widehat{\omega}) \\
&\quad + \gamma \left| \int_{\mathcal{S}} \left(P_\omega \left(s' \mid s, a \right) - P_{\widehat{\omega}} \left(s' \mid s, a \right) \right) V_\omega^{\pi,n} \left(s' \right) ds' \right| \\
&\quad + \gamma \left| \int_{\mathcal{S}} P_{\widehat{\omega}} \left(s' \mid s, a \right) \left(V_\omega^{\pi,n} \left(s' \right) - V_{\widehat{\omega}}^{\pi,n} \left(s' \right) \right) ds' \right| \\
&\leq L_{\omega_r} d_\Omega(\omega, \widehat{\omega}) + \gamma L_V(\omega) \sup_{\|f\|_L \leq 1} \left\{ \left| \int_{\mathcal{S}} \left(P_\omega \left(s' \mid s, a \right) \right. \right. \right. \\
&\quad \left. \left. \left. - P_{\widehat{\omega}} \left(s' \mid s, a \right) \right) f \left(s' \right) ds' \right| \right\} \\
&\quad + \gamma \max_{s'} \left| V_\omega^{\pi,n} \left(s' \right) - V_{\widehat{\omega}}^{\pi,n} \left(s' \right) \right| \\
&\leq \left(L_{\omega_r} + \gamma L_{\omega_P} L_V(\omega) + \gamma L_{\omega_Q}^n \right) d_\Omega(\omega, \widehat{\omega}). \qquad (4.14)
\end{aligned}
$$

Consequently, the inequality 4.8 holds. Now, if the sequence $L_{\omega_Q}^n$ is convergent, it converges to the fixed point of the recurrence equation:

$$
L_{\omega_Q} = L_{\omega_r} + \gamma L_{\omega_P} L_V(\omega) + \gamma L_{\omega_Q}. \qquad (4.15)
$$

Hence the limit point is the one expressed in 4.9, and the sequence can be proven to be convergent since $\gamma < 1$. □

Lemma 4.3.2. *Given Assumptions 2.4.1, 2.4.3 and 4.3.1, if $\gamma L_P(1+ L_{\pi_{\boldsymbol{\theta}}}) < 1$, the Kantorovich distance between a pair of γ-discounted feature state distributions is PLC w.r.t. parameters $\boldsymbol{\theta} : \forall(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}) \in \Theta^2$:*

$$
\mathcal{K} \left(\delta_\mu^{\boldsymbol{\theta}}, \delta_\mu^{\widehat{\boldsymbol{\theta}}} \right) \leq L_\delta(\boldsymbol{\theta}) d_\Theta(\boldsymbol{\theta}, \widehat{\boldsymbol{\theta}}); \qquad (4.16)
$$

where $L_\delta(\boldsymbol{\theta}) = \frac{\gamma L_P L_\pi(\boldsymbol{\theta})}{1 - \gamma L_P \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right)}$.

In the same fashion, we can prove the following lemma:

Lemma 4.3.3 (L-continuity of meta state occupancy measures). *Given Assumptions 1.1.1.3 and 3.1, if $\gamma L_P(\omega) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) < 1$, then the Kantorovich distance between a pair of γ-discounted feature-state distributions is PLC w.r.t. task ω:*

$$
\mathcal{K} \left(\delta_{\mu,\omega}^{\boldsymbol{\theta}}, \delta_{\mu,\widehat{\omega}}^{\boldsymbol{\theta}} \right) \leq L_\delta(\omega) d_\Omega(\omega, \widehat{\omega}), \quad \forall(\omega, \widehat{\omega}) \in \Omega^2; \qquad (4.17)
$$

where $L_\delta(\omega) = \frac{\gamma L_{\omega_P}}{1 - \gamma L_P(\omega) \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right)}$.

Proof.

$$
\begin{aligned}
&\mathcal{K}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega},\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)=\sup_{f}\left\{\left|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}(s)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}(s)\right)f(s)ds\right|:\|f\|_{L}\leq 1\right\}\\
&=\sup_{f}\left\{\left|_{\mathcal{S}}\left(\mu(s)+\gamma\int_{\mathcal{S}}\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)P_{\omega}\left(s\mid s',a\right)\delta_{\mu,\omega}\left(s'\right)dads'\right)f(s)-\right.\right.\\
&\quad\left.\left.-\left(\mu(s)+\gamma\int_{\mathcal{A}}\int_{\mathcal{S}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)P_{\widehat{\omega}}\left(s\mid s',a\right)\delta_{\mu,\widehat{\omega}}\left(s'\right)dads'\right)f(s)ds\right|:\|f\|_{L}\leq 1\right\}\\
&=\gamma\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}f(s)\int_{\mathcal{A}}\int_{\mathcal{S}}\left(P_{\omega}\left(s\mid s',a\right)\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)\right.\right.\right.\\
&\quad\left.\left.\left.\cdot-P_{\widehat{\omega}}\left(s\mid s',a\right)\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)ds'dads\right|\right\}\\
&=\gamma\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}f(s)\int_{\mathcal{A}}\int_{\mathcal{S}}P_{\omega}\left(s\mid s',a\right)\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)ds'dads\right.\right.\\
&\quad\left.\left.+\int_{\mathcal{S}}f(s)\int_{\mathcal{A}}\int_{\mathcal{S}}\left(P_{\omega}\left(s\mid s',a\right)-P_{\widehat{\omega}}\left(s\mid s',a\right)\right)\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)ds'dads\right|\right\}\\
&\leq\underbrace{\gamma\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\int_{\mathcal{S}}P_{\omega}\left(s\mid s',a\right)f(s)dsdads'\right|\right\}}_{(1)}\\
&\quad+\underbrace{\gamma\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\int_{\pi}^{\boldsymbol{\theta}}\left(a\mid s'\right)\int_{\mathcal{S}}\left(P_{\omega}\left(s\mid s',a\right)-P_{\widehat{\omega}}\left(s\mid s',a\right)\right)f(s)dsdads'\right|\right\}}_{(2)}.
\end{aligned}
\tag{4.18}
$$

Now, we focus on term (1):

$$
\begin{aligned}
&\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)\underbrace{\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\int_{\mathcal{S}}P_{\omega}\left(s\mid s',a\right)f(s)dsda}_{h^{\boldsymbol{\theta}}_{f,\omega}(s')}ds'\right|\right\}\\
&=L_{P}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\sup_{f:\|f\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)\frac{h^{\boldsymbol{\theta}}_{f,\omega}\left(s'\right)}{L_{P}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)}ds'\right|\right\}\\
&\leq L_{P}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\sup_{\tilde{f}:\|\tilde{f}\|_{L}\leq 1}\left\{\left|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}\left(s'\right)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\right)\tilde{f}\left(s'\right)ds'\right|\right\} \qquad (4.19)\\
&\leq L_{P}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\mathcal{K}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega},\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right). \qquad (4.20)
\end{aligned}
$$

where 4.19 comes from the fact that $h^{\boldsymbol{\theta}}_{f,\omega}\left(s'\right):=\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\int_{\mathcal{S}}P_{\omega}\left(s\mid s',a\right)f(s)dsda$ is $L_{P}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)$-PLC w.r.t. the state space $\mathcal{S}$. From the other side, the term (2) can be

bounded as follows:

$$\begin{aligned}
&\sup_{f:\|f\|_L\leq 1}\left\{\left|\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\int_{\mathcal{S}}\left(P_{\omega}\left(s\mid s',a\right)-P_{\widehat{\omega}}\left(s\mid s',a\right)\right)f(s)dsdads'\right|\right\}\\
&\leq\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\sup_{f:\|f\|_L\leq 1}\left\{\left|\int_{\mathcal{S}}\left(P_{\omega}\left(s\mid s',a\right)-P_{\widehat{\omega}}\left(s\mid s',a\right)\right)f(s)ds\right|\right\}dads'\\
&\leq\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\left(s'\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}\left(a\mid s'\right)\mathcal{K}\left(P_{\omega}\left(\cdot\mid s',a\right),P_{\widehat{\omega}}\left(\cdot\mid s',a\right)\right)dads'\\
&\leq L_{\omega_P}d_{\Omega}(\omega,\widehat{\omega}).
\end{aligned}\tag{4.21}$$

Finally, merging everything together:

$$\begin{aligned}
\mathcal{K}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega},\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)&\leq\gamma L_P(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\mathcal{K}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega},\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)+\gamma L_{\omega_P}d_{\Omega}(\omega,\widehat{\omega})\\
&\leq\frac{\gamma L_{\omega_P}}{1-\gamma L_P(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)}d_{\Omega}(\omega,\widehat{\omega}).
\end{aligned}\tag{4.22}$$

□

As a direct consequence, it is easy to prove that:

$$\mathcal{K}\left(\zeta^{\boldsymbol{\theta}}_{\mu,\omega},\zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)\leq L_{\delta}(\omega)\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)d_{\Omega}(\omega,\widehat{\omega}).\tag{4.23}$$

Proof.

$$\begin{aligned}
\mathcal{K}\left(\zeta^{\boldsymbol{\theta}}_{\mu,\omega},\zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)&=\sup_{f:\|f\|_L\leq 1}\left\{\left\|\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\omega}(s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a\mid s)f(s,a)dads\right.\right.\\
&\qquad\left.\left.-\int_{\mathcal{S}}\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}(s)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a\mid s)f(s,a)dads\right\|\right\}\\
&=\sup_{f:\|f\|_L\leq 1}\left\{\left\|\int_{\mathcal{S}}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega}(s)-\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}(s)\right)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a\mid s)f(s,a)dads\right\|\right\}\\
&\leq\left(1+L_{\pi_{\boldsymbol{\theta}}}\right)\mathcal{K}\left(\delta^{\boldsymbol{\theta}}_{\mu,\omega},\delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right).
\end{aligned}\tag{4.24}$$

where in 4.24 we used the fact that, for a function f defined on $\mathcal{S}\times\mathcal{A}$ such that $\|f\|F_L\leq 1$, then $\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a\mid s)f(s,a)da$ is $(1+L_{\pi_{\boldsymbol{\theta}}})$-LC. □

Lemma 4.3.4 (L-continuity of η). *Given Assumptions 2.4.1, 2.4.3, 2.4.4 and 4.3.1, $\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a)=\nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}(s,a)Q^{\boldsymbol{\theta}}_{\omega}(s,a)$ is L-LC w.r.t. the task space Ω:*

$$\begin{aligned}
\left|\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a)-\eta^{\boldsymbol{\theta}}_{i,\widehat{\omega}}(s,a)\right|&=\left|\nabla_{\boldsymbol{\theta}_i}\log\pi_{\boldsymbol{\theta}}(a\mid s)\left(Q^{\boldsymbol{\theta}}_{\omega}(s,a)-Q^{\boldsymbol{\theta}}_{\widehat{\omega}}(s,a)\right)\right|\\
&\leq\mathcal{M}^i_{\boldsymbol{\theta}}\left|Q^{\boldsymbol{\theta}}_{\omega}(s,a)-Q^{\boldsymbol{\theta}}_{\widehat{\omega}}(s,a)\right|\\
&\leq\mathcal{M}^i_{\boldsymbol{\theta}}L_{\omega_Q}d_{\Omega}(\omega,\widehat{\omega}).
\end{aligned}\tag{4.25}$$

Moreover, η is also L-LC w.r.t. the joint state-action space $\mathcal{S} \times \mathcal{A}$:

$$\left|\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a) - \eta^{\boldsymbol{\theta}}_{i,\omega}(\widehat{s},\widehat{a})\right| \leq L^{i}_{\eta^{\boldsymbol{\theta}}(\omega)} d_{\mathcal{S}\times\mathcal{A}}((s,a),(\widehat{s},\widehat{a})); \tag{4.26}$$

where $L^{i}_{\eta^{\boldsymbol{\theta}}(\omega)} = \frac{R_{\max}}{1-\gamma} L^{i}_{\nabla \log \pi_{\boldsymbol{\theta}}} + \mathcal{M}^{i}_{\boldsymbol{\theta}} L_{Q^{\boldsymbol{\theta}}(\omega)}$.

Theorem 4.3.5 (L-continuity of ∇J). *Given Assumptions 2.4.1, 2.4.3, 2.4.4 and 4.3.1, the return gradient is LC-continuous w.r.t. the task space Ω:*

$$\left|\nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\omega} - \nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right| \leq L_{\nabla J}(\omega) d_{\Omega}(\omega,\widehat{\omega}); \tag{4.27}$$

where $L_{\nabla J}(\omega) = L^{i}_{\eta^{\boldsymbol{\theta}}_{\omega}} \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) L_{\delta}(\omega) + \mathcal{M}^{i}_{\boldsymbol{\theta}} L_{\omega_Q}$.

Proof.

$$\begin{aligned}
&\left|\nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\omega} - \nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right| = \left|\mathop{\mathbb{E}}_{(s,a)\sim\zeta^{\boldsymbol{\theta}}_{\mu,\omega}}\left[\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a)\right] - \mathop{\mathbb{E}}_{(s,a)\sim\zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}}\left[\eta^{\boldsymbol{\theta}}_{i,\widehat{\omega}}(s,a)\right]\right| \\
&\leq \left|\int_{\mathcal{S}}\int_{\mathcal{A}} \left(\zeta^{\boldsymbol{\theta}}_{\mu,\omega} - \zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right)(s,a)\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a) da ds\right| + \left|\mathop{\mathbb{E}}_{(s,a)\sim\zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}}\left[\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a) - \eta^{\boldsymbol{\theta}}_{i,\widehat{\omega}}(s,a)\right]\right| \\
&\leq L^{i}_{\eta^{\boldsymbol{\theta}}(\omega)} \mathcal{K}\left(\zeta^{\boldsymbol{\theta}}_{\mu,\omega}, \zeta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}\right) + \left|\int_{\mathcal{S}} \delta^{\boldsymbol{\theta}}_{\mu,\widehat{\omega}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s)\left(\eta^{\boldsymbol{\theta}}_{i,\omega}(s,a) - \eta^{\boldsymbol{\theta}}_{i,\widehat{\omega}}(s,a)\right) da ds\right| \\
&\leq \left[L^{i}_{\eta^{\boldsymbol{\theta}}_{\omega}} \left(1 + L_{\pi_{\boldsymbol{\theta}}}\right) L_{\delta}(\omega) + \mathcal{M}^{i}_{\boldsymbol{\theta}} L_{\omega_Q}\right] d_{\Omega}(\omega,\widehat{\omega}).
\end{aligned} \tag{4.28}$$

□

The Theorem 4.3.5 derives an upper bound on the distance between the gradients $\nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\omega_1}$ and $\nabla_{\boldsymbol{\theta}_i} J^{\boldsymbol{\theta}}_{\mu,\omega_2}$ of two tasks $\omega_1, \omega_2 \in \Omega$ belonging to the same set $\mathcal{M}$. Similarly to Proposition 4.3.1, this result represents a guarantee on the generalization capabilities of the approach under the LC assumption. In particular, a boundary on the distance between gradients of different tasks ensures a certain regularity in the surface of the return function, which is important since the gradient is included in the meta state to capture implicit information about the context space.

4.4 The Algorithm

We conclude the chapter by proposing an algorithm for the solution of meta-MPDs. As highlighted in the previous section, these models are intended to be general and they can be applied to learning agents of any form. These properties, although necessary to represent meta-RL tasks, make the optimization problem difficult to solve in the general case. For this reason, this thesis restrict the analysis to meta-MDPs in which the learning model is a Policy Gradient method, thus having an update rule of the form:

$$f(\boldsymbol{\theta}, h) = \boldsymbol{\theta} + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}). \tag{4.29}$$

Under this assumption, the learning action $h \in \mathcal{H}$ reduces to the choice of the step size of a Gradient Ascent update. The algorithm described in this section, together with the experimental results presented in the next chapter, refer to the optimization of this value. Despite this, most aspects discussed in this context can be generalized to different models.

As a meta-learning approach, the objectives of our algorithm are to improve the generalization capabilities of PG methods and to remove the need of manually tuning the learning rate for each task. The identification of an optimal dynamic step size serves two purposes: it maximizes the convergence speed by performing large updates when allowed and it improves the overall training stability, selecting low values when the return is near to the optimum or the current region is uncertain.

To accomplish these goals, we propose the adoption of the FQI algorithm to the problem of hyperparameter learning. This choice is motivated by the convergences guarantees of value based approaches such as FQI. The idea is to use the generalization capabilities of regression algorithms to dynamically recommend the most adequate step size for each setting. The algorithm can be outlined in three phases: data generation, learning and evaluation.

4.4.1 Phase 1 - Data Generation

The initial phase is responsible for the construction of a four-tuples dataset $\mathcal{F} = \{(x_i, h_i, x'_i, \mathcal{L}_i)\}$ to be used in the estimation of a meta action-value function $\hat{Q}(x, h)$. The dataset is created by generating a sequence of n meta episodes, associated to random tasks $\mathcal{M}_i$ sampled from $\mathscr{M}$. At the start of each episode, the policy $\pi(\boldsymbol{\theta}_i)$ is initialized to a fixed value or drawn form a probability distribution. Then, each meta step consists in generating a trajectory in the task $\mathcal{M}_i$ according to the policy $\pi(\boldsymbol{\theta}_i)$ and computing the gradient $\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i)$. The meta reward $\mathcal{L}(x, \boldsymbol{\theta}_i, h)$ is produced and $\pi(\boldsymbol{\theta}_i)$ is updated using a random meta action h, as follows:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i). \tag{4.30}$$

The tuple $\{(x, h, x', \mathcal{L})\}$ is added to F, to be used in the next phase of the algorithm. A compact version of this phase is described below:

Algorithm 10 Generate Dataset

Input: set of tasks $\mathscr{M}$, parameterized policy $\pi(\boldsymbol{\theta})$, number of meta episodes n, time horizon of meta episode T

procedure GENERATEDATASET($\mathscr{M}, \pi(\boldsymbol{\theta})$, n, T)
  $\mathcal{F} = \{\}$
  for $i = 1, \ldots, n$ meta episodes **do**
    Sample task $\mathcal{M}_i \sim p(\mathscr{M})$
    Sample initial policy $\pi(\boldsymbol{\theta}_i)$
    repeat
      Sample trajectory $\{(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)\}$ in task $\mathcal{M}_i$ according to $\pi(\boldsymbol{\theta}_i)$
      Compute gradient $\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i)$
      Sample meta action $h \sim U(h_{min}, h_{max})$
      Update policy $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i)$
      Compute meta reward $\mathcal{L}(x, \boldsymbol{\theta}_i, h)$
      Append $\{(x, h, x', \mathcal{L})\}$ to $\mathcal{F}$
    until the time horizon T is reached
  return $\mathcal{F}$

To be successful, this phase must create a set of experiences that accurately represent the returns across the context and policy spaces. In practice, this consists in collecting as many meta state-action pairs (x, h) as possible.

Each observed meta state x includes a representation of the current task and policy. For example, a possible formulation contains the policy and the context parameters, i.e. $x = \langle \boldsymbol{\theta}_i, \phi_{\mathcal{M}}(\mathcal{M}_i) \rangle$. Given a desired dimension $|\mathcal{F}| = n \times T$ for the dataset, we can prioritize the exploration of one element with respect to the other, according to the problem's needs. To maximize the number of tasks observed, we need to set a high number of meta episodes n, at the expense of less policies collected for each context. Setting a high value of T produces the opposite result. The degenerate case is a purely generative approach, caused by a time horizon $T = 1$, in which a single step is taken in each task.

4.4.2 Phase 2 - Learning the Step Size

In its second phase, our approach applies the FQI algorithm to the set previously created. In this case, FQI iteratively generates an approximation of the action-value function $\hat{Q}$,

dependent on x and h, able to identify the context of unseen tasks and induce an optimal policy.

Each iteration n builds a training set $\mathcal{TS} = \left\{\left(i^l, o^l\right), l = 1, \cdots, |\mathcal{F}|\right\}$ to feed a regression algorithm. In this set, the inputs i consist of meta state-action couples, while the outputs o are created using the meta reward $\mathcal{L}$ and the previous approximation $\hat{Q}_{n-1}$:

$$\begin{aligned} i &= (x, h) \\ o &= \mathcal{L} + \tilde{\gamma} \max_{h \in \mathcal{H}} \hat{Q}_{n-1}\left(x', h\right). \end{aligned} \tag{4.31}$$

Once the training set is built, a regression algorithm f is used to learn a new approximation $\hat{Q}_n$. The complete process is described below:

Algorithm 11 Fitted Q-Iteration

Input: a set of four-tuples $\mathcal{F} = \{(x_i, h_i, x'_i, \mathcal{L}_i)\}$ and a regression algorithm f

1: **procedure** FQI($\mathcal{F}, f$)
2: Initialize $\hat{Q}_n = 0$ everywhere on $\mathcal{X} \times \mathcal{H}$
3: $n \leftarrow 0$
4: **repeat**
5: $n \leftarrow n + 1$
6: Build the training set $\mathcal{TS} = \left\{\left(i^l, o^l\right), l = 1, \cdots, |\mathcal{F}|\right\}$ based on $\hat{Q}_{n-1}$ and $\mathcal{F}$:

$$\begin{aligned} i^l &= \left(x_t^l, h_t^l\right) \\ o^l &= r_t^l + \gamma \max_{h \in \mathcal{H}} \hat{Q}_{n-1}\left(x_{t+1}^l, h\right) \end{aligned}$$

7: Use f to induce from $\mathcal{T}S$ the function $\hat{Q}_n(x, h)$
8: **until** the stopping conditions are reached

Regression Algorithm

In our solution, the regression task is carried by an ExtraTrees algorithm, that bases its logic on an ensemble of decision trees. This approach, introduced in Section 2.3.4, selects a random subset of candidate features and random thresholds at each decision node. Then, it compute a score for each candidate and selects the best to split the node. This process usually reduces the variance at the expense of a slight increase in bias. An ExtraTrees method is mainly regulated by two parameters:

- the number of estimators (trees), that regulates the prediction accuracy. Generally, larger ensembles improve the regression quality until a certain number is reached,

after which the accuracy reaches a plateau. In addition, a high number of trees can significantly impact the time complexity;

- the minimum number of samples to split a node, that influences the generalization capabilities of each element in the ensemble. A decision tree in which the nodes contains few samples is more likely to overfit the training set and fail on new data.

Double Clipped Q Function

As described before, each FQI iteration approximates the Q function using the estimates made in the previous step. As the process goes on, the sequence of these compounding approximations can seriously degrade the overall performance of the algorithm. In particular, FQI tends to suffer of overestimation bias, similarly to other value based approaches that rely on taking the maximum of a noisy Q function.

To countermeasure this tendency, we adopt a modified version of Clipped Double Q-learning, introduced by (Fujimoto et al., 2019), to penalize uncertainties over future states. This approach consists in maintaining two parallel Q functions and choosing the action h that maximizes a convex combination of the minimum and the maximum between them:

$$\mathcal{L} + \tilde{\gamma} \max_{h} \left[\lambda \min_{j=1,2} Q_{\boldsymbol{\theta}_j} \left(x', h \right) + (1 - \lambda) \max_{j=1,2} Q_{\boldsymbol{\theta}_j} \left(x', h \right) \right],$$

with $\lambda > 0.5$. If we set $\lambda = 1$, the update corresponds to Clipped Double Q-learning. The minimum operator penalizes high variance estimates in regions of uncertainty and pushes the policy towards actions leading to states already seen in the dataset.

4.4.3 Phase 3 - Performance Evaluation

In the last phase, the performance of our approach is measured on unseen samples. It's common practice for meta learning to evaluate the algorithm on new tasks coming from the same distribution used for training.

Following this method, we sample a random context and generate a meta episode. At each step, we measure the return obtained after selecting the best action h in a greedy manner:

$$h^* = \arg\max_{h} \hat{Q}(x, h). \tag{4.32}$$

The complete evaluation procedure is listed below:

Algorithm 12 Evaluate meta-FQI

Input: set of tasks $\mathscr{M}$, parameterized policy $\pi(\boldsymbol{\theta})$, time horizon of meta episode T, meta-action value function estimator $\hat{Q}$.

procedure EVALUATE($\mathscr{M}, \pi(\boldsymbol{\theta}), T$)
    Meta rewards $\mathcal{R} = \{\}$
    Sample task $\mathcal{M}_i \sim p(\mathscr{M})$
    Sample initial policy $\pi(\boldsymbol{\theta}_i)$
    repeat
        Sample trajectory $\{(s_1, a_1, r_1, \ldots, s_k, a_k, r_k)\}$ in task $\mathcal{M}_i$ according to $\pi(\boldsymbol{\theta}_i)$
        Compute gradient $\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i)$
        Select action $h^* = \arg\max_h \hat{Q}(x, h)$
        Update policy $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + h\nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}_i)$
        Compute meta reward $\mathcal{L}(x, \boldsymbol{\theta}_i, h)$
        Append $\mathcal{L}$ to $\mathcal{R}$
    until time horizon T is reached
    return $\mathcal{R}$

Chapter 5

Experimental Results

In this chapter, we analyze in detail the performance of our approach in various settings. We present two experiments that highlight the strengths of the algorithm, followed by an example in which some limitations arise. For each empirical evaluation, we provide an interpretation of the results and reflect on possible improvements.

During each experiment, we measure different configurations of the same algorithm, addressing the factors that impact the results. Then, we compare the performance against classic PG methods trained with different update techniques.

5.1 Implementation Details

To guarantee a reliable implementation and the reproducibility of the results, our solution is built on top of OpenAI Gym (Brockman et al., 2016), a Tensorflow-compatible (Abadi et al., 2016) library designed to create simulated RL environments. This toolkit is widely adopted for its stability and minimal interface. It also offers a broad range of popular benchmark tasks, including many classic control problems and Atari Games.

For similar reasons, the main algorithm follows the design choices proposed by another OpenAI library, known as Baselines (Dhariwal et al., 2017). As the name suggests, Baselines offers standardized implementations of various RL algorithms, to serve as baselines for comparisons with new approaches.

Given these premises, we now present the implementation choices adopted to configure the FQI algorithm and to build the components of the meta-MDP.

FQI Configuration

The main aspect to configure in the FQI optimization is the regressor, that in our approach is implemented as an ExtraTrees algorithm. As described in Section 4.4.2, this method can be tuned by varying mainly two parameters: the number of trees in the

ensemble and the minimum number of samples required to split a node. After a series of trials, we completed the experiments in this chapter with the following configurations:

- number of trees = 50 to 150, that represent a good compromise between accuracy of estimation and computational costs;
- minimum samples for the split = 1% to 2.5% of the total samples, to ensure a good level of generalization on unseen tasks.

Set of Tasks

In all experiments, the set of tasks $\mathscr{M}$ is realized as a CMDP $\mathcal{M}(c) = \langle \mathcal{S}, \mathcal{A}, P^c, R^c, \gamma^c, \mu^c \rangle$, where the context c is a set of n parameters that alter the dynamics of the system, inducing a probability distribution. During training, each meta episode takes place in a different task, obtained by sampling the context from a Multivariate Uniform distribution:

$$c \sim U(c_{min}, c_{max}); \tag{5.1}$$

where c_{min} and c_{max} are n-dimensional vectors representing the boundaries of the context space examined.

Meta Space

The meta space $\mathcal{X}$ determines the level of information that the algorithm observes about the context and policy spaces. Among the various choices proposed in the previous Chapter, we found the most promising results with two implementations of $x \in \mathcal{X}$, respectively $x = \langle \boldsymbol{\theta}_t, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t), \phi_{\mathcal{M}}(\mathcal{M}_t) \rangle$ and $x = \langle \boldsymbol{\theta}_t, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) \rangle$. All the experiments in this chapter employ one of these meta state instances.

Policy

The policy $\pi(\boldsymbol{\theta})$ is responsible for selecting the actions to take in the specific tasks. It's also the entity we aim to optimize during training. In our implementation, it's realized as a multi layers perceptron (MLP) or, in most cases, as a linear policy:

$$\pi(\boldsymbol{\theta}, s) = \boldsymbol{\theta}^\top s. \tag{5.2}$$

At the start of each episode, the policy parameter $\boldsymbol{\theta}$ can be initialized to a fixed value or drawn at random. In our experiments, the policies are usually sampled from Uniform or Gaussian distributions, depending on the case.

Policy Gradient Algorithm

As already mentioned in the previous chapter, this work restricts the analysis to meta-MDPs with a learner f in the form of a Policy Gradient algorithm. In its simplest form, a PG approach improves the policy $\pi(\boldsymbol{\theta})$ according to a return function $J(\boldsymbol{\theta})$, dependent on the parameterization. The optimization is realized with a sequence of Gradient Ascent updates:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}); \tag{5.3}$$

where α is the step size and $\nabla J_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is the gradient of the value function. A complete explanation of the algorithm can be found in Section 2.3.1.

In our experiments, we implement f as a Natural Policy Gradient, a variation of the approach that operates in a "trust region" manner, in contrast to the "line search" technique adopted by the classic algorithm. In this version, each iteration constrains the KL divergence between two consecutive policies to be under a value δ. The update step, derived in Section 2.3.6, is expressed as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g, \tag{5.4}$$

where $g = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ and H is the Fisher Information Matrix, that measures curvature of the log probability of the policy. The expression H^{-1} is computed with a Conjugate Gradient method, also explained in Section 2.3.6.

Finally, we normalize the Natural Gradient dividing it by its L^2-norm. This operation preserves the direction of the update and delegates the decision about the step size entirely to the meta action h.

5.2 Experimental Methodology

We now describe the general methodologies adopted to evaluate the algorithm. For each experiment, we analyze the training environment, describing its difficulty level and its variation depending on the context. We present the performance of the algorithm, evaluating its convergence speed and overall stability.

We measure our result against a PG method trained with different fixed step sizes and linearly decreasing step sizes, verifying if our solution represents an improvement with respect to these approaches.

In conclusion, we evaluate various configurations of the algorithm, measuring the performance of different FQI iterations and different meta states implementations.

To ensure the fairness of the comparisons, all the methods presented are executed fixing the same random seeds that generates the tasks and trajectories. This way, we can measure the approaches in identical conditions.

5.3 Navigation 2D

For our first evaluation of the approach, we reproduce one of the experiment presented in (Finn et al., 2017), called Navigation2D. This environment consists of a 2D square space in which an agent, represented as a point, aims to reach a goal in the plane traversing the minimum distance.

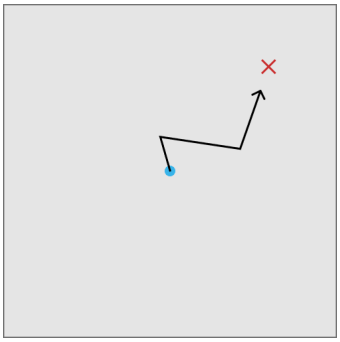


Figure 5.1: Nav2D - Reach a random point in a 2D space.

At the start of the episode, the agent is placed in the initial position $s_0 = (0, 0)$ of the Cartesian plane. Then, at each step t the agent observes its current position and performs an action a_t corresponding to movement speeds along the x and y axes:

$$a_t = (v_x, v_y), \text{ where } v_x, v_y \in [-v^{\max}, v^{\max}]. \tag{5.5}$$

According to this action, the agent can move in every direction of the plane, with a limit on the maximum speed $v^{\max} = 0.1$ allowed in a single step. This parameters determines the minimum number of steps necessary to reach the goal and can be varied to tune the difficulty of the environment.

At each step, the environment produces a reward equal to the negative Euclidean distance from the goal:

$$r_t = \sqrt{(x_t - x_{\text{goal}})^2 + (y_t - y_{\text{goal}})^2}. \tag{5.6}$$

An episode terminates when the agent is within a threshold distance d_{thresh} from the goal or when the horizon $H = 10$ is reached.

The distribution of tasks is implemented as a CMDP $\mathcal{M}(c)$ in which, at each episode, a different goal point is selected at random. The context c is given by a 2D vector, such that:

$$c = (x_{\text{goal}}, y_{\text{goal}}), \text{ where } x_{\text{goal}}, y_{\text{goal}} \sim U(-1, 1). \tag{5.7}$$

5.3.1 Policy Gradient Performance

We begin by analyzing the performance of a standard Natural Policy Gradient algorithm in the environment. The Figure 5.2 illustrates the results obtained by a fixed step size $\alpha = 5$ over 20 random tasks. For each trajectory, a step represents the cumulative reward collected by the policy in a episode of the specific context. The Policy Gradient update is applied once for each step.

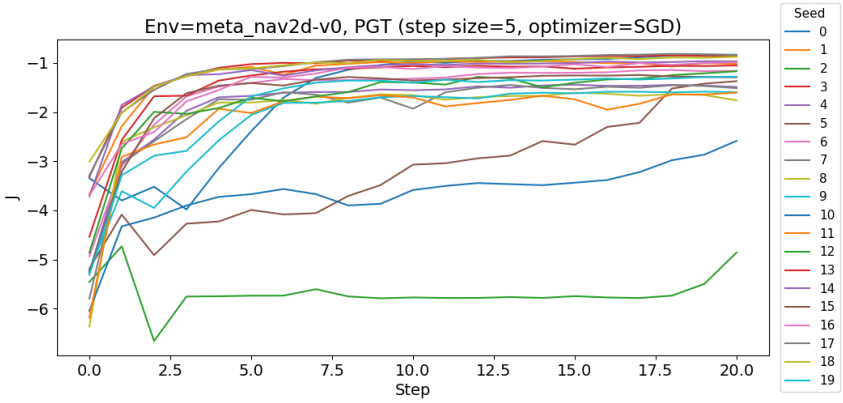


Figure 5.2: Nav2D - Fixed PG performance ($\alpha = 5$).

As we can note, the results are already acceptable: after the 8-th update step, most trajectories are able to reach the goal with a cost lower than 2. However, some tasks suffer of slow learning or never reach the optimal value.

We show this particular step size because it represents a good trade-off between speed of convergence and stability of improvement. Lower values of this parameter produce less variable results, but require a longer time to converge.

5.3.2 FQI Performance

The Figure 5.3 illustrates the results obtained by our approach in the same 20 random tasks. The first plot shows the cumulative rewards, the second indicates the dynamic step size $h \in \mathcal{H}$ selected at each step. As we can note, the algorithm is able to select higher step sizes w.r.t. the classic approach (up to 8), without suffering from any drop. The algorithm is able to calibrate its action, starting with larger improvements and slowing down once the policy obtains good results. In addition, all trajectories converge around the 5-th meta step, an improvement compared to Figure 5.2.

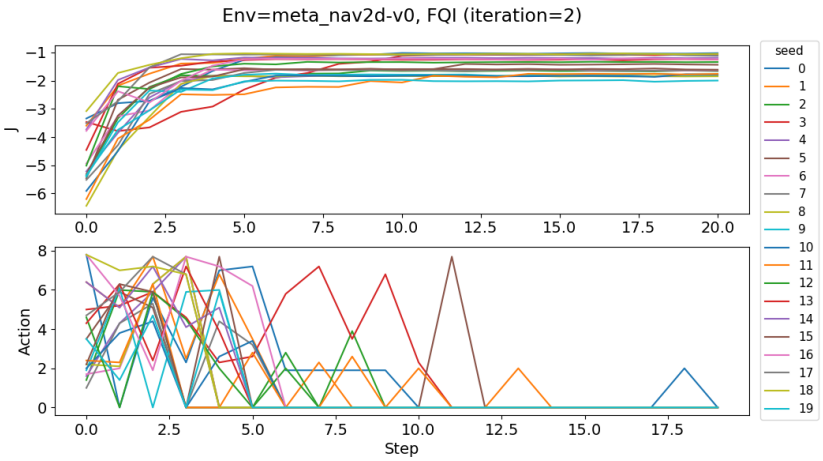


Figure 5.3: Nav2D - Dynamic PG performance ($\alpha \in [0, 8]$).

This evaluation is accomplished training the algorithm with the following parameters:

- number of estimators = 50, minimum samples split = 0.01;
- meta state $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i), \phi_{\mathcal{M}}(\mathcal{M}_i) \rangle$;
- step size $h \in [0, 8]$;
- policy parameter initialization $\boldsymbol{\theta} \sim \mathcal{N}(0, 0.1)$.

If we compare different iterations of the approach, we can observe that the performance starts to degrade after the second, due to the effect of compounding approximation errors.

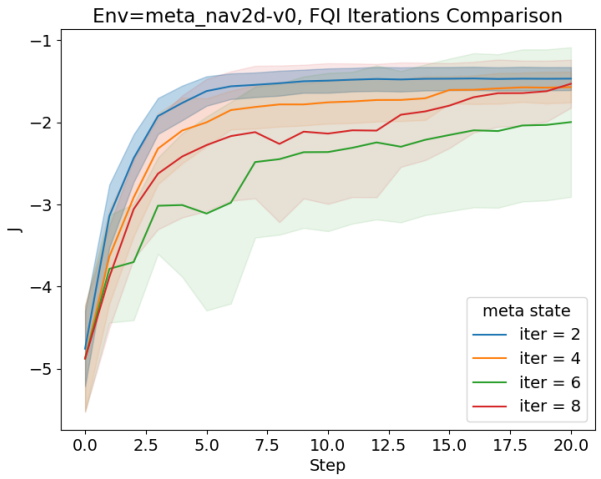
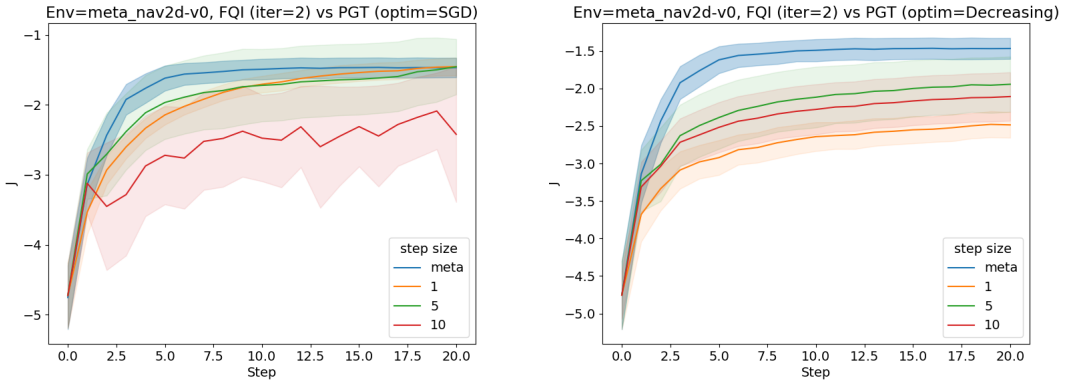


Figure 5.4: Nav2D - FQI iterations comparison.

5.3.3 FQI vs Policy Gradient

We now evaluate the results obtained by FQI against different configurations of the classic Natural PG approach. In particular, we compare the performance against different fixed step sizes α and linearly decreasing step sizes $\frac{\alpha}{t}$. Figure 5.5 illustrates the outcome of the comparisons. Our approach is able to reach faster convergence, while maintaining higher stability w.r.t. the benchmarks.



(a) Dynamic PG vs Fixed PG

(b) Dynamic PG vs Linearly Decreasing PG

Figure 5.5: Nav2D - Dynamic PG trained with FQI vs Benchmarks.

5.3.4 Considerations

In light of the outcomes showed above, we perform a further analysis to understand which factors contributed the most to the result. Our algorithm induces the approximation $\hat{Q}$ from a set of inputs (x, h), where x is the meta state and h is the step size. In this experiment, we adopted the most informative version of x among the ones presented in the previous chapter, i.e. $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i), \phi_{\mathcal{M}}(\mathcal{M}_i) \rangle$, where $\phi_{\mathcal{M}}(\mathcal{M}_i) = c$. The parameters $\boldsymbol{\theta}$ and the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$ are both represented by 6-dimensional vectors, while the context c is 2-dimensional. In Figure 5.6, we illustrate a ranking of these features based on the importance attributed to them by the ExtraTrees regressor.

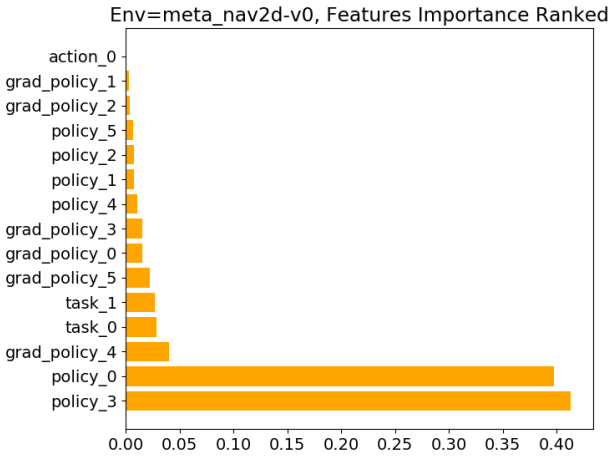


Figure 5.6: Nav2D - Ranking of features importances in ExtraTrees regressor.

As we can see, the most important factors belong to the policy, followed by elements of the gradient and information about the tasks. Interestingly, the action is considered the least important feature in the prediction. This fact could be explained by the relative simplicity of this environment. It is reasonable to say that the main challenge of this meta-MDP is the identification of the task, that is deducing the position of the goal from variations in the Euclidean distance and pointing the agent in the correct direction. Once the task is correctly identified, the action to take in the environment remains the same until the goal is reached, hence reducing the importance of the step size feature in the regression.

Another aspect that we can gather from the image is that the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ and the explicit task parameterization provide a comparable amount of information. Given this fact, we can evaluate the performance of a second model that can't observe the explicit parameterization of the current task and learns by only using the implicit information included in the gradient. The meta state in this case is $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i) \rangle$. The Figure 5.7 shows a comparison between these configurations:

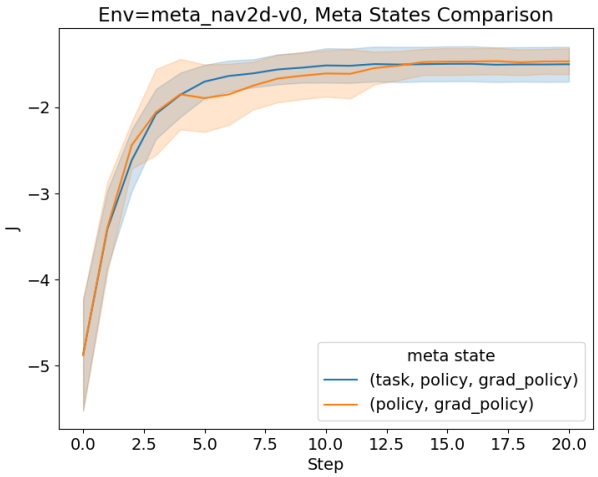


Figure 5.7: Nav2D - Meta States implementations comparison.

As we can observe, the two meta states are comparable, with a slight increase in variance for the version with less information. This result provides a first example that supports the use of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ as an appropriate implicit representation of the context space.

5.4 Minigolf

The environment for our second experiment is inspired by the article "The physics of putting" (Penner, 2002), that models the dynamics of a rolling ball on golf greens of various characteristics.

In our implementation, we consider the ideal scenario of a perfectly flat surface, in which a golf putter hits a ball with radius r in the direction of a hole with diameter D. The objective is to place the ball inside the hole in the minimum number of strokes.

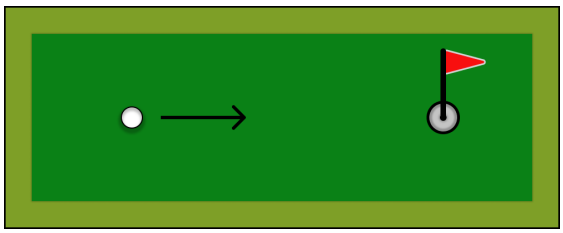


Figure 5.8: Minigolf - Hit the golf ball and center the hole in the green.

The friction imposed by the green surface is modeled by a constant deceleration

$d = \frac{5}{7}\rho g$, where ρ is the dynamic friction coefficient between the ball and the ground and g is the gravitational acceleration.

At the start of each episode, the ball is placed at a random distance x_0 from the hole between $0m$ and $20m$. At each step, the agent selects the angular velocity ω of the putter swing, producing the initial ball velocity $v_0 = \omega l$ (where l is the length of the putter). As described by the original paper, a shot can have three outcomes:

- if $v_0 < v_{\min} = \sqrt{\left(\frac{10}{7}\right)\rho g x_0}$ the shot is too weak and the ball doesn't reach the hole. The environment returns a reward of -1 and the episode continues for another attempt. In the next step, the putter strikes from a distance x_1, derived according to a Uniformly Decelerated Motion as $x_1 = x_0 - v_0 t + \frac{1}{2}dt^2$, where the variable time is $t = \frac{v_0}{d}$;
- if $v_0 > v_{\max} = \sqrt{(2D - r)^2 \frac{g}{2r} + v_{\min}^2}$ the shot is too strong and the ball overcomes the hole, losing the game. The environment returns a reward of -100 and the episode ends with a failure;
- if $v_{\min} < v_0 < v_{\max}$ the ball has the adequate velocity to reach the hole without overcoming it, winning the game. The environment returns a reward of 0 and the episode ends with a success.

Given these properties, an optimal policy for the Minigolf environment is one that reaches the hole with one stroke from a generic distance. It follows that the optimal cumulative reward reachable in an episode is -1.

In order to add some uncertainty to the game, the actual velocity v_0 produced by the swing of the putter is given by $v_0 = \omega l(1 + \epsilon)$, where $\epsilon \sim \mathcal{N}(0, 0.25)$. The addition of this multiplicative noise creates a variance in the action proportional to ω (and to l, if we consider it as a variable). This discourages the agent to attempt a one-shot hole and incentives a multi-step approach to the goal.

During the experiment, we set the environment parameters to imitate the dynamics of a realistic shot in a minigolf green, within the limits of our simplified simulation. This is the complete configuration adopted:

- horizon $H = 20$;
- discount factor $\gamma = 0.99$;
- angular velocity $\omega \in [1 \times 10^{-5}, 10]$;
- initial distance $x_0 \in [0, 20]$ meters;
- ball radius $r = 0.02135$ meters;
- hole diameter $D = 0.10$ meters;

- gravitational acceleration $g = 9.81 \frac{\text{meters}}{\text{second}^2}$.

The distribution of tasks is built as a CMDP $\mathcal{M}(c)$, induced by a set of parameters $c = (l, \rho)$. At each meta episode, a new task is sampled from a Multivariate Uniform distribution within this ranges:

- putter length $l \sim U(0.7, 1)$ meters;
- friction coefficient $\rho \sim U(0.065, 0.196)$.

Selecting these random parameters, the meta-MDP simulates the activity of hitting a golf ball with different putters and in different greens.

5.4.1 Policy Gradient Performance

We begin the analysis by studying how the meta-MDP varies as the context c changes. Since both the action $a_t = \omega_t$ and the state $s_t = x_t$ are scalars, the linear policy adopted in our experiments reduces to a single 2-parameters equation:

$$\pi(s_t, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top s_t = w s_t + b. \tag{5.8}$$

Figure 5.9 shows different edge cases of the context space. In each subplot, the cumulative reward J is represented as a function of the policy parameters $\boldsymbol{\theta} = (w, b)$. The performance is graded by a color scale, in which the darker tones indicate the optimal values of the policy.

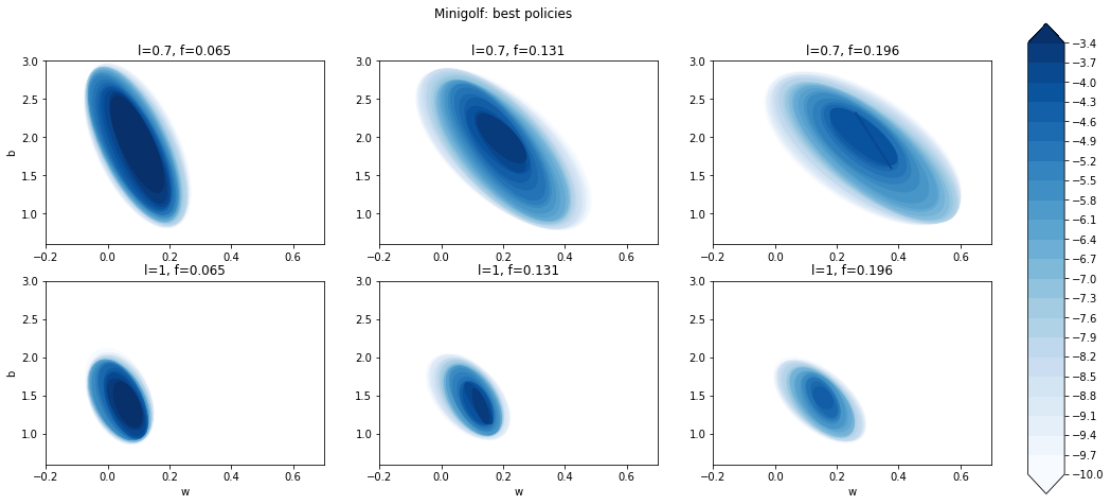


Figure 5.9: Minigolf - Evaluation of the optimal region as function of the policy.

In the image, plots belonging to the same row have the same putter length, while plots belonging to the same column have the same friction. As we can note, the behavior

of the environment changes significantly in the different contexts. In particular, higher values of the putter length l increase the difficulty to reach the hole, thus reducing the dimension of the optimal area. On the other hand, increases in the friction coefficient ρ have the effect of shifting the orientation of the optimal area and reducing the maximum reachable return.

After this initial evaluation, let's observe the results obtained in the environment by a Natural PG algorithm. Figure 5.10 shows the outcome of 20 random tasks trained with a fixed step size $\alpha = 0.1$.

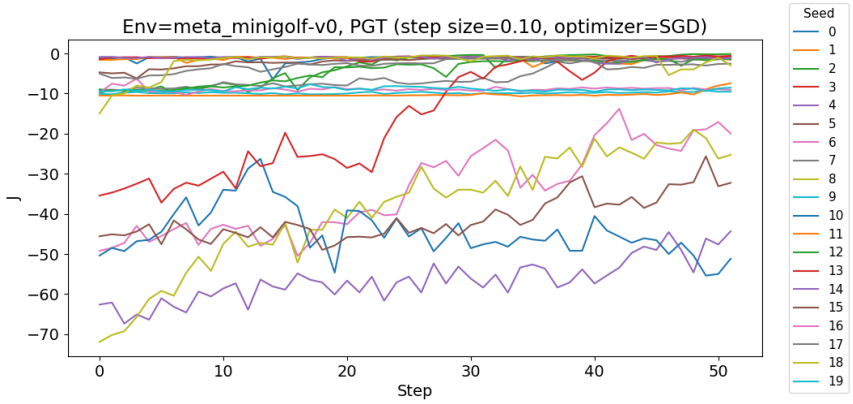


Figure 5.10: Minigolf - Fixed PG performance ($\alpha = 0.1$).

As we can see, the results in different context are quite varied. A portion of the trajectories starts from favourable conditions and it's able to reach the optimal value. On the other hand, policies that are randomly initialized to bad values struggle to converge and are affected by serious instability and frequent drops. In addition, there seems to be a local optimum that limits some trajectories at a return $J = -10$ and, most importantly, a serious variance problem.

5.4.2 FQI Performance

Figure 5.11 illustrates the performance of our approach in the same set of task. The first subplot shows that, using a dynamic step size higher than before (up to 1), the algorithm is able to consistently reach the optimal values, even for the trajectories that start from adverse positions. In addition, the convergence is achieved within 20 meta steps of training, a substantial improvement w.r.t. the instability seen in the classic PG approach. The only aspect that remains unsolved is the local optimum at $J = -10$. The subplot on the right highlights the parameters $\boldsymbol{\theta} = (w, b)$ updating towards the optimal area of the policy space.

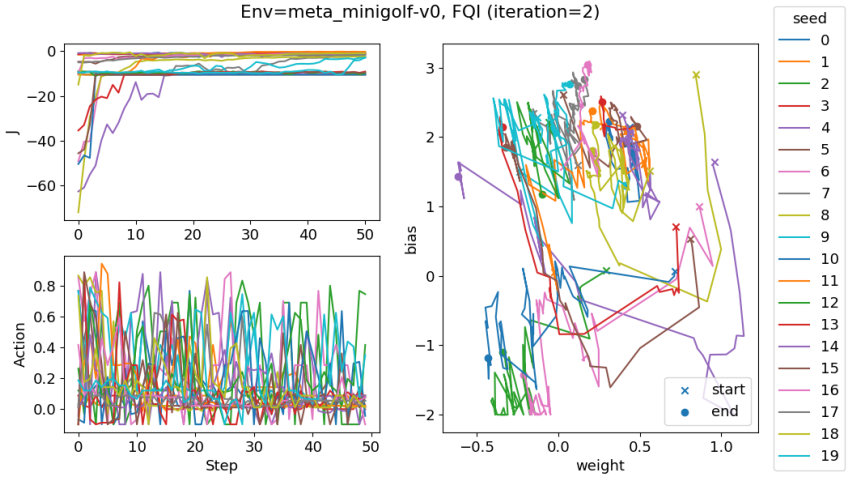


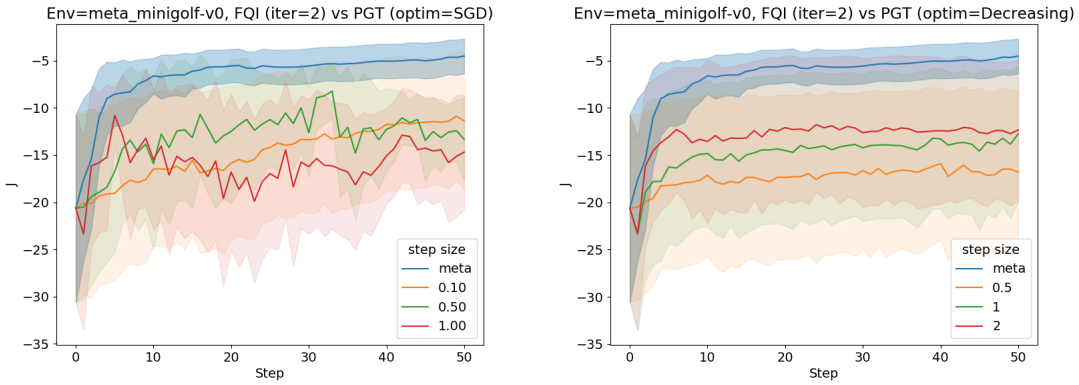
Figure 5.11: Minigolf - Dynamic PG performance ($\alpha \in [0, 1]$).

This evaluation is accomplished training the algorithm with the following parameters:

- number of estimators = 50, minimum samples split = 0.01;
- meta state $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i), \phi_{\mathcal{M}}(\mathcal{M}_i) \rangle$;
- step size $h \in [0, 1]$;
- policy parameter initialization $\boldsymbol{\theta} = (w, b) \sim U((-1, 2), (-2, 3.5))$. This range is wider than the one presented in Figure 5.9 to evaluate the convergence of policies that start from adverse positions.

5.4.3 FQI vs Policy Gradient

We proceed in the analysis by comparing the results against different PG configurations, as done in the previous experiment. In this environment, our approach shows an even bigger advantage, converging to a value much higher than its competitors and significantly reducing the variance in the optimization.



(a) Dynamic PG vs Fixed PG

(b) Dynamic PG vs Linearly Decreasing PG

Figure 5.12: Minigolf - Dynamic PG trained with FQI vs Benchmarks.

5.4.4 Considerations

Once again, we observe the importance assigned by the ExtraTrees algorithm to the meta features (x, h) and draw some additional insights about the result. In the meta space x adopted in the experiment, the context space parameterization c, the policy parameter $\boldsymbol{\theta}$ and the gradient $\nabla J_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ are all represented by 2-dimensional vectors.

The image below shows that the most important features are the gradients, followed by the meta action. Contrarily to the previous experiment, the tasks information is considered the least important to make a prediction.

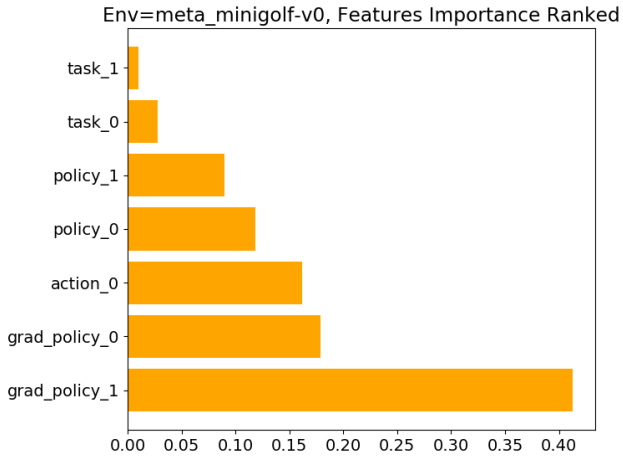


Figure 5.13: Minigolf - Ranking of features importances in ExtraTrees regressor.

To understand this ranking, we need to consider the preliminary analysis made on the environment and the performance achieved by the classic PGT algorithm. In Section 5.4.1, we showed that the meta-MDP is quite heterogeneous, a property that is in apparent contrast with the low importance assigned to the context parameterization c. A possible interpretation is that the gradient $\nabla J_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is able to capture some hidden information about the task that is not included in the explicit context parameterization. In addition, the poor results obtained by the various fixed approaches show that, for each task, this environment requires a dynamic update of the policy to achieve an optimal value, thus explaining the high importance attributed to the meta action h.

After these observations, we compare this model to a second one that doesn't include the context c in its meta state, expecting to obtain similar results. Figure 5.14 shows the outcome of the comparison.

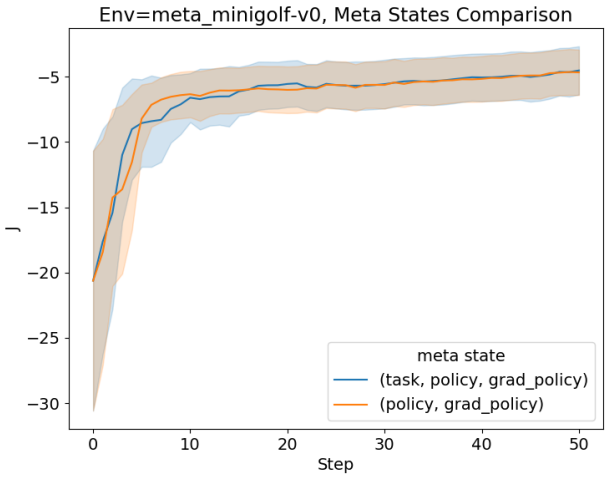


Figure 5.14: Minigolf - Meta States implementations comparison.

In accordance to our considerations, the second model is able to replicate the performance of the first, while having access to a smaller set of information.

5.5 CartPole

For our third experiment, we examine the CartPole balancing task (Barto et al., 1990), also known as the Inverted Pendulum problem. This environment is a classic example of control theory and a popular benchmark for Reinforcement Learning algorithms. The name derives from the most popular implementation of the problem, that consists in a pole attached to a cart by a non actuated joint, making it an inherently unstable

system. The cart can move horizontally along a frictionless track to balance the pole. The objective is to maintain the equilibrium as long as possible.

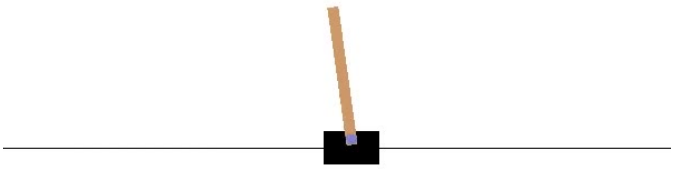


Figure 5.15: CartPole - Balance a pole on a cart.
Source: OpenAI.

Our experiments are conducted using a modified version of the "CartPole-v0" environment, provided by the OpenAI Gym library. In this implementation, an episode starts with the pendulum in vertical position. At each step, the agent observes the following 4-tuple of continuous values:

- cart position $x_{\text{cart}} \in [-4.8, 4.8]$;
- cart velocity $v_{\text{cart}} \in \mathbb{R}$;
- pole angle $\phi_{\text{pole}} \in [-0.418, 0.418]$ rad;
- pole angular velocity $\omega_{\text{pole}} \in \mathbb{R}$.

Given the state, the agent chooses an action between 0 and 1 to push the cart to the left or to the right. For each step in which the pole is in balance, the environment produces a reward of +1. An episode ends when the pole angle from the vertical position is higher than 12 degrees, or the cart moves more than 2.4 units from the center, or the horizon $H = 100$ is reached.

In our experiments, we set the environment parameters to these values:

- mass of the cart $m_{cart} = 1$ kg;
- length of the pole $l_{pole} = 0.5$ m;
- force applied the cart $F = 10$ N.

The CMDP $\mathcal{M}(c)$ is induced by varying two environment parameters, the gravity g and the pole mass m_{pole}, that form the context parameterization $c = (g, m_{pole})$. Each task in the meta-MDP is built by sampling c from a Multivariate Uniform distribution, within these ranges:

- gravity $g \sim U(9.8, 12.8)$ $\frac{\text{m}}{\text{s}^2}$;
- pole mass $m_{pole} \sim U(0.1, 3)$ kg.

5.5.1 Policy Gradient Performance

As done for previous experiments, we begin the analysis by observing the performance of a classic PG algorithm. In this case, the best results were obtained with a fixed step size $\alpha = 2.5$. As we can see from the Figure 5.16, most trajectories begin with rapid improvements, achieving acceptable returns in less than 5 steps. However, none of them are able to consistently reach the optimum of $J = 100$ (equivalent to balancing the pole for the entire time horizon) and drop to lower returns by the end of the episode. The trajectories that struggle to converge correspond to higher values of the parameters g and m_{pole}, that naturally make the balancing task more difficult.

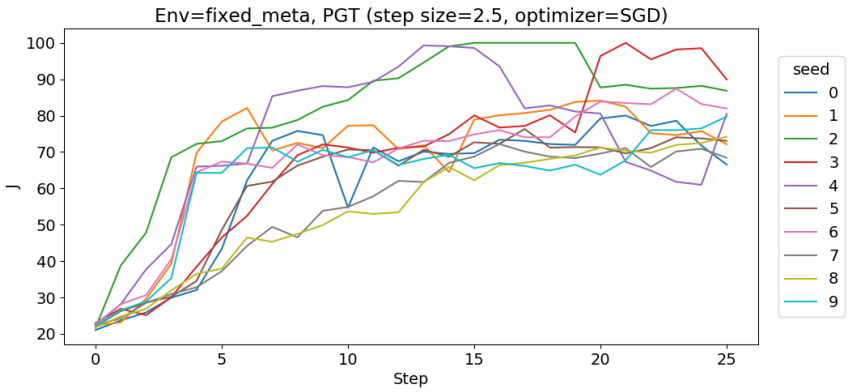


Figure 5.16: CartPole - Fixed PG performance ($\alpha = 7.5$)

5.5.2 FQI Performance

We now evaluate the performance of our approach in the same set of tasks. As showed in Figure 5.17, the convergence speed is comparable to the fixed approach, with most trajectories reaching a local optimum within 5 update steps. On the other hand, we can note an overall improvement in the training stability: In this case, a larger number of trajectories reaches a return close to $J = 100$, without suffering of any drop. There is

still a considerable variance in the final return obtained by the trajectories, due to the different difficulty levels of the various contexts.

An important aspect to highlight is that, once again, our approach is able to dynamically select high step sizes (up to $\alpha = 10$) when the return function offers large margins of improvement, to then resort to smaller actions when the optimal area is reached, guaranteeing the stability of the result.

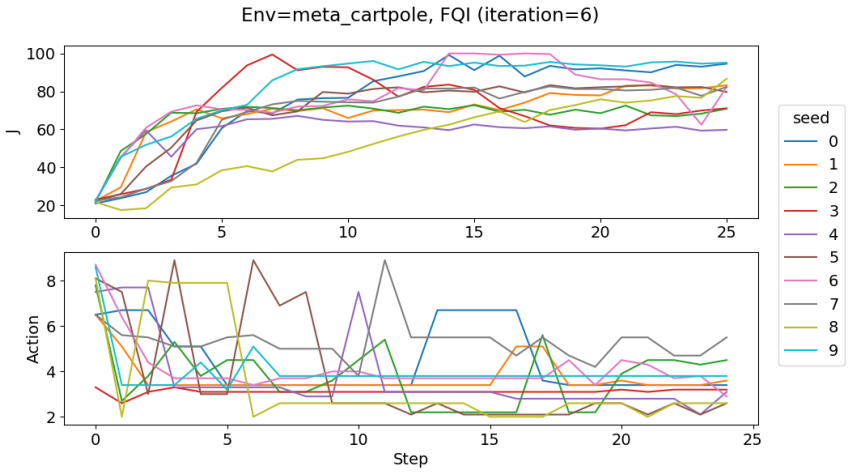


Figure 5.17: CartPole - Dynamic PG performance ($\alpha \in [0, 10]$)

This evaluation is accomplished training the algorithm with the following parameters:

- number of estimators = 150, minimum samples split = 0.025;
- meta state $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i), \phi_{\mathcal{M}}(\mathcal{M}_i) \rangle$;
- step size $h \in [0, 10]$;
- policy parameter initialization $\boldsymbol{\theta} \sim \mathcal{N}(0, 0.1)$.

We now compare the results obtained by different iterations of the FQI algorithm. Differently from previous experiments, in which we were able to obtain good results already from the second or third iteration, this experiment requires a long-sighted estimate of the Q function. In particular, we obtain the best performance at the 6-th iteration, after an initial sequence of iterations characterized by slower improvements. This is due to a tendency of the algorithm to prioritize stability over speed of convergence, causing the selection of step sizes lower than necessary. Later iterations are able to correct this behavior, but suffer of higher variance.

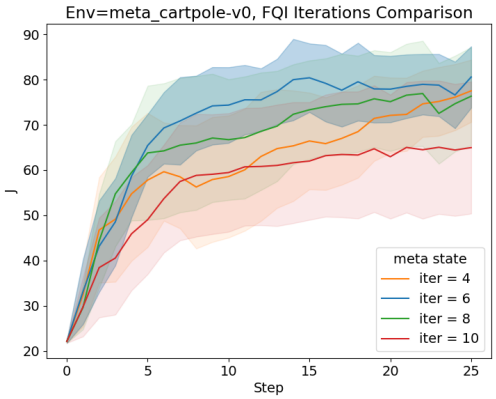
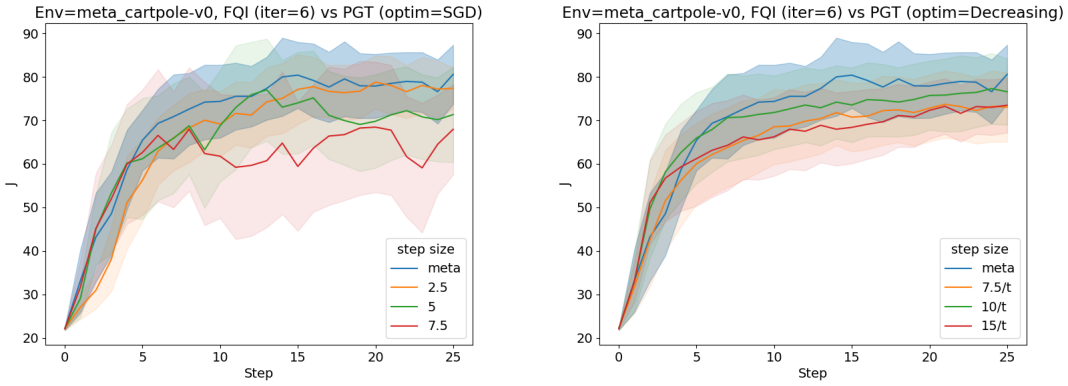


Figure 5.18: CartPole - FQI iterations comparison.

5.5.3 FQI vs Policy Gradient

In the Figure 5.19, we illustrate an evaluation of the approach against different fixed configurations of the PG algorithm. In this environment, our method shows a small advantage in terms of performance, converging slightly faster than the benchmarks, but obtaining a final return comparable to the best fixed approaches. An interesting aspect to note is the significantly higher stability of the Linearly Decreasing PG w.r.t. to the Fixed PG, testifying that this environment could be suited to such configuration.

Aside from the modest improvements, this result can still be considered a success from the perspective of an automatic hyperparameter tuning task. Our approach is able to autonomously select a sequence of step sizes that match the performance of the best fixed configurations identified after a manual fine tuning of the parameter.



(a) Dynamic PG vs Fixed PG

(b) Dynamic PG vs Linearly Decreasing PG

Figure 5.19: CartPole - Dynamic PG trained with FQI vs Benchmarks.

5.5.4 Considerations

In conclusion, we observe the importance assigned to each feature by the ExtraTrees regressor to gain some additional insights. The first element we can gather from Figure 5.20 is that, for the first time, the gradients $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ are considered the least informative factors for the prediction. This could be due to a high level of similarity between the J functions of the various tasks, that reduces the amount of information captured by the gradient. A second aspect to note is that, in this experiment, one feature belonging to the explicit task parameterization is the most important in the ranking (by far). In particular, this element is associated to the mass of the pole m_{pole}. On the other hand, the value of the gravity is considered not important. During our tests, we attempted many configurations of this environment, varying the mass of the cart, the length of the pole and the force applied to the cart. All these features were considered not important w.r.t. to m_{pole}.

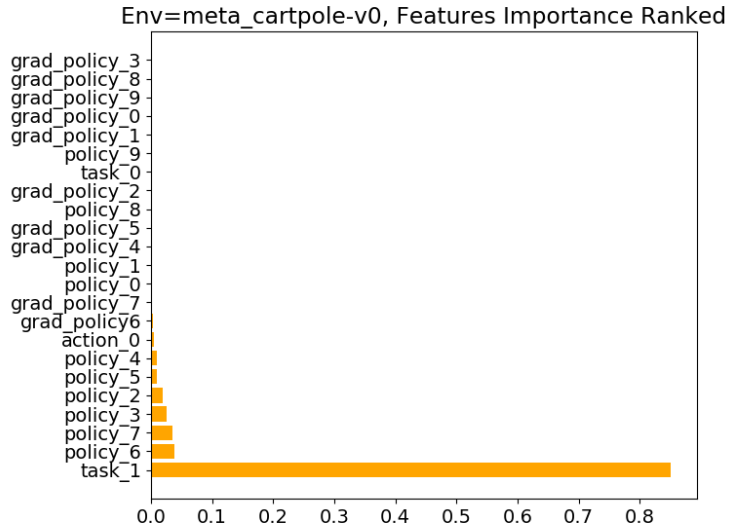


Figure 5.20: CartPole - Ranking of features importances in ExtraTrees regressor.

The outcome of this ranking allows us to evaluate the reduced meta state $x = \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i) \rangle$ in a scenario in which the gradient doesn't seem able to represent the differences between the various contexts. In Figure 5.21, we illustrate a comparison of the two meta states proposed in this thesis:

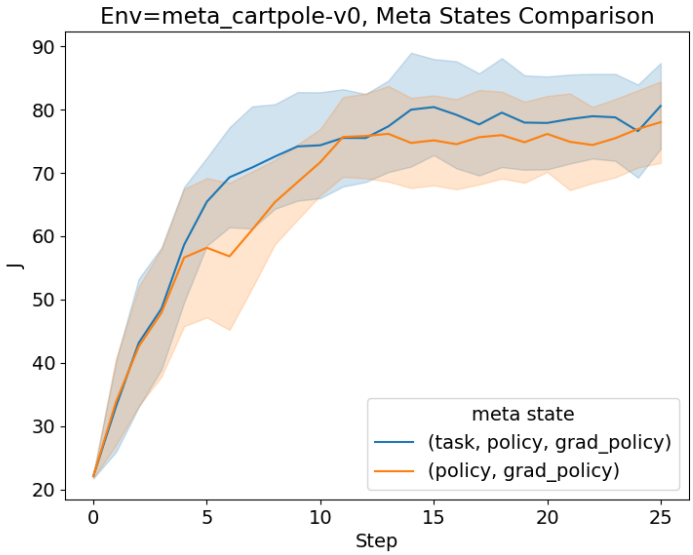


Figure 5.21: CartPole - Meta States implementations comparison.

In this case, the approach trained with the reduced meta state converges with lower speed and suffers of higher instability w.r.t. the complete version. On the other hand, the final returns achieved are comparable, meaning that the ExtraTrees regressor was able to extract implicit information about the tasks from the other features, i.e. the parameters $\boldsymbol{\theta}_i$ and the gradient $\nabla J_{\boldsymbol{\theta}}(\boldsymbol{\theta}_i)$.

Chapter 6

Conclusions

After an empirical evaluation of the performance, we conclude our work with some considerations about the results achieved and we reflect on future improvements.

In this thesis, we focused on Meta Reinforcement Learning, the problem of crafting models that can quickly learn new skills and adapt to unseen RL environments. After an introduction to the background knowledge needed to understand the topic, we presented a formalization of the meta-RL problem, known as meta-MDP. This is a general framework to solve a set of RL tasks, represented as a Contextual Markov Decision Process. We described the most important elements of the model, i.e. the meta state x, the meta action h and the meta objective function $\mathcal{L}$. We discussed various possible implementation of the meta state, providing the motivations for our choices. Then, we analyzed the case of Lipschitz meta-MDPs, deriving some general guarantees that hold if the tasks in the model are Lipschitz continuous.

Finally, we devised an algorithm to learn in meta-MDPs, focusing on the instances in which the learning algorithm is a Policy Gradient approach. In these cases, the meta action h reduces to the choice of the step size of the Gradient Ascent update. The idea of the approach is to apply the Fitted Q Iteration algorithm to build an approximation of the action-value function $\hat{Q}$, on the basis of the meta features observed (which depend on the current policy adopted) and the step size h. This estimate is used to solve the two main challenges posed by a meta-RL problem: the identification of the current task $\mathcal{M}_i$ and the improvement of the policy π.

The approach has been evaluated in different settings that highlighted its strengths, but also showed some current limitations. In particular, we observed that the algorithm can outperform usual PG approaches thanks to the generalization capabilities of the ExtraTrees regressor. In two cases analyzed, our approach achieved significant improve-

ments in terms of total reward obtained and convergence speed, while guaranteeing a higher stability compared to the benchmarks. These experiments also showed that the same results are reacheable without an explicit parameterization of the task in the meta state, demonstrating that the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ can be a valid candidate to capture information about the context. This is a crucial aspect in perspective of the scalability of the approach, that may suffer if the feature space becomes too large.

In the last experiment, the approach obtained minimal advantages, reaching comparable results to the best fixed step size we could identify. Possible causes are an insufficient number of samples in the dataset or, to give a simpler explanation, that a standard approach is sufficient to reach an optimal solution in the environment. Aside from improving the performance, the outcome can be still considered a success from the point of view of automatic hyperparameter selection. In all experiments, the approach was able to dynamically identify good step size values within the given range. This aspect can be seen as a small step in the AutoML direction, in which a practitioner could set up a maximum value for the step size, run the algorithm and, with some guidance, obtain an acceptable performance without the need of manual fine tuning.

The algorithm also suffers of some limitations that are worth mentioning. First of all, the regression still requires a considerable amount of data to perform adequately. To solve this issue, we need to study a better way to collect training samples, instead of simply drawing random actions from a uniform distribution.

Second, the approach tends to avoid risks, sometimes selecting actions smaller than necessary to prioritize stability over speed of convergence. This is due to a common issue of FQI, that can be quite short-sighted in its first iterations. As the training proceeds, the algorithm improves its predictions and starts recommending higher step sizes. However, this comes at the price of compounded approximations in the estimates, that could become a problem when the data set is small. Since reducing the amount of samples is also an objective, these problems are in conflict and need further addressing.

At the moment, we are analysing a possible improvement, denoted as FQI-KDE, in which we run a Kernel Density Estimation on the sampled policies between each iteration of the algorithm, to induce some knowledge about their distribution and perform an update with additional insight. The tests are being conducted on simulated environments provided by the Meta-World (Yu et al., 2019) library, to evaluate the sample efficiency of the approach in larger settings.

Finally, there are some considerations that step outside the restricted scope of learning the step size, and apply to the general meta-MDP framework. Often, the most significant improvements in a problem aren't achieved by improving the optimization technique, but by identifying the correct formulation. The challenge of finding a good

framework to solve meta-RL tasks is still very open, also considering that the field is a niche in the broader space of RL.

In our particular model, the elements that offer the largest room for improvement are the meta loss function and the meta space. Defining a loss function that can better incentive a fast and stable learning over different tasks would benefit the performance of any algorithm applied to the framework. Similarly, identifying a meta space that is generally informative without an explicit parameterization of the context is a crucial aspect to solve. In our experiments, the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ was used to reach these goals, but this solution is not guaranteed to succeed in the same measure for high dimensional contexts. At the moment, we are considering the introduction of an Autoencoder architecture to include a compressed representation of the context in the meta state. The hope is that this advancement can help the algorithm to tackle more ambitious problems.

Bibliography

Abadi, Martín et al. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv: 1603.04467 [cs.DC].

Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson (1990). "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems". In: *Artificial Neural Networks: Concept Learning*. IEEE Press, pp. 81–93. ISBN: 0818620153.

Botvinick, Matthew et al. (2019). "Reinforcement Learning, Fast and Slow". In: *Trends in Cognitive Sciences* 23.5, pp. 408–422. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2019.02.006. URL: https://www.sciencedirect.com/science/article/pii/S1364661319300610.

Brockman, Greg et al. (2016). *OpenAI Gym*. arXiv: 1606.01540 [cs.LG].

Brown, Noam and Tuomas Sandholm (2019). "Superhuman AI for multiplayer poker". In: *Science* 365.6456, pp. 885–890. ISSN: 0036-8075. DOI: 10.1126/science.aay2400. eprint: https://science.sciencemag.org/content/365/6456/885.full.pdf. URL: https://science.sciencemag.org/content/365/6456/885.

Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters (Aug. 2013). "A Survey on Policy Search for Robotics". In: *Found. Trends Robot* 2.1–2, pp. 1–142. ISSN: 1935-8253. DOI: 10.1561/2300000021. URL: https://doi.org/10.1561/2300000021.

Dhariwal, Prafulla et al. (2017). *OpenAI Baselines*. https://github.com/openai/baselines.

Duan, Yan et al. (2016). *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning*. arXiv: 1611.02779 [cs.AI].

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). "Tree-Based Batch Mode Reinforcement Learning". In: *Journal of Machine Learning Research* 6.18, pp. 503–556. URL: http://jmlr.org/papers/v6/ernst05a.html.

Finn, Chelsea (2020). "Meta Reinforcement Learning - Learning to explore". In: *CS 330 at Stanford University*. URL: https://cs330.stanford.edu/slides/cs330_metarl2_2020.pdf.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. arXiv: 1703.03400 [cs.LG].

Fujimoto, Scott, David Meger, and Doina Precup (2019). *Off-Policy Deep Reinforcement Learning without Exploration*. arXiv: 1812.02900 [cs.LG].

Garcia, Francisco M. and Philip S. Thomas (2019). *A Meta-MDP Approach to Exploration for Lifelong Reinforcement Learning*. arXiv: 1902.00843 [cs.LG].

Geurts, Pierre, Damien Ernst, and Louis Wehenkel (Apr. 2006). "Extremely randomized trees". In: *Machine Learning* 63.1, pp. 3–42. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1. URL: https://doi.org/10.1007/s10994-006-6226-1.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). *Neural Turing Machines*. arXiv: 1410.5401 [cs.NE].

Hallak, Assaf, Dotan Di Castro, and Shie Mannor (2015). *Contextual Markov Decision Processes*. arXiv: 1502.02259 [stat.ML].

Hui, Jonathan (2020). "Meta-Learning (Learning how to Learn)". In: *https://jonathan-hui.medium.com*. URL: https://jonathan-hui.medium.com/meta-learning-learn-how-to-learn-9095142ef3d6.

Kakade, Sham (2001). "A Natural Policy Gradient". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural*

and Synthetic. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, pp. 1531–1538.

Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov (2015). "Siamese Neural Networks for One-shot Image Recognition". In: *ICML Deep Learning Workshop*.

Levine, Sergey (2020). "Meta-Learning". In: *CS 285 at UC Berkeley*. URL: `http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-22.pdf`.

Li, Ke and Jitendra Malik (2016). *Learning to Optimize*. arXiv: `1606.01885 [cs.LG]`.

Li, Xiaoyu and Francesco Orabona (2019). *On the Convergence of Stochastic Gradient Descent with Adaptive Stepsizes*. arXiv: `1805.08114 [stat.ML]`.

Mnih, Volodymyr et al. (2013). *Playing Atari with Deep Reinforcement Learning*. arXiv: `1312.5602 [cs.LG]`.

Nichol, Alex, Joshua Achiam, and John Schulman (2018). *On First-Order Meta-Learning Algorithms*. arXiv: `1803.02999 [cs.LG]`.

Penner, A Raymond (2002). "The physics of golf". In: *Reports on Progress in Physics* 66.2, p. 131.

Pirotta, Matteo, Marcello Restelli, and Luca Bascetta (2015). "Policy gradient in lipschitz markov decision processes". In: *Machine Learning* 100.2-3, pp. 255–283.

Rachelson, Emmanuel and Michail G. Lagoudakis (2010). "On the Locality of Action Domination in Sequential Decision Making". In: *11th International Symposium on Artificial Intelligence and Mathematics (ISIAM 2010)*. Fort Lauderdale, US, pp. 1–8. URL: `https://oatao.univ-toulouse.fr/17977/`.

Sadeghi, Fereshteh and Sergey Levine (2017). *CAD2RL: Real Single-Image Flight without a Single Real Image*. arXiv: `1611.04201 [cs.LG]`.

Santoro, Adam et al. (2016). "Meta-Learning with Memory-Augmented Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, pp. 1842–1850.

Schulman, John et al. (2017). *Trust Region Policy Optimization*. arXiv: 1502.05477 [cs.LG].

Senior, Andrew W. et al. (Jan. 2020). "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577.7792, pp. 706–710. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1923-7. URL: https://doi.org/10.1038/s41586-019-1923-7.

Silver, David et al. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. arXiv: 1712.01815 [cs.AI].

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN: 0262039249.

Szepesvari, Csaba (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers. ISBN: 1608454924.

Tobin, Josh et al. (2017). *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. arXiv: 1703.06907 [cs.RO].

Wang, Jane X et al. (2017). *Learning to reinforcement learn*. arXiv: 1611.05763 [cs.LG].

Watkins, Christopher J. C. H. and Peter Dayan (May 1992). "Q-learning". In: *Machine Learning* 8.3, pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

Weng, Lilian (2018). "Meta-Learning: Learning to Learn Fast". In: *lilianweng.github.io/lil-log*. URL: http://lilianweng.github.io/lil-log/2018/11/29/meta-learning.html.

Weng, Lilian (2019a). "Domain Randomization for Sim2Real Transfer". In: *lilianweng.github.io/lil-log*. URL: http://lilianweng.github.io/lil-log/2019/05/04/domain-randomization.html.

— (2019b). "Meta Reinforcement Learning". In: *lilianweng.github.io/lil-log*. URL: http://lilianweng.github.io/lil-log/2019/06/23/meta-reinforcement-learning.html.

Wolpert, D. H. and W. G. Macready (1997). "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82. DOI: 10.1109/4235.585893.

Xu, Zhongwen, Hado van Hasselt, and David Silver (2018). *Meta-Gradient Reinforcement Learning*. arXiv: 1805.09801 [cs.LG].

Yu, Tianhe et al. (2019). *Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning*. arXiv: 1910.10897 [cs.LG].