ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
Data Science and Computation
CICLO XXXV

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI
Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Exploiting and Generalizing Epistemic Uncertainty in Reinforcement Learning and Planning

Presentata da: Amarildo Likmeta

Coordinatore Dottorato
Prof. Daniele Bonacorsi

Supervisore
Prof. Marcello Restelli

Esame finale anno 2024

Abstract

Solving general sequential decision-making problems with unknown complex and highly non-linear dynamics has been a central goal of Artificial Intelligence since the conception of the research field. Reinforcement Learning (RL) offers an extremely general framework for solving such problems. Its approach of solving sequential decision-making problems by direct interaction with the environment, allowing for speculation on the value of candidate solutions, testing, and counter-factual reasoning, has allowed researchers to achieve remarkable achievements in a multitude of challenging problems both simulated and real-world. Advances in deep learning have brought about the rise of the Deep Reinforcement Learning (DeepRL) field as a general framework for solving almost all sequential decision-making tasks. Nonetheless, the successful application of RL to new problems requires a large degree of task-specific tuning, and many tasks still represent a challenge for the current algorithms.

One of the main open challenges in RL remains the *exploration-exploitation* dilemma. An agent that optimizes a cumulative objective in an unknown environment while learning faces the question of whether to trust the current information gathered and exploit it by executing the best-known strategies or take explorative strategies to gather more information with the hope of finding better strategies. The exploration problem has been thoroughly studied in the literature, and a multitude of solutions have been given for tabular domains or continuous domains with known structure. However, when moving to complex domains where neural networks are employed as function approximators, deep and directed exploration is still a challenge. Numerous attempts have been made to extend provably efficient techniques from the tabular case to the DeepRL case with mixed results.

A general recipe for achieving efficient exploration has been to endow agents with an estimate of the uncertainty on the value functions to allow for decisions that account for the agent's lack of complete information. While these techniques allow for provable efficient exploration in tabular settings, their extension to DeepRL still faces numerous challenges. The main techniques to achieve uncertainty estimates in DeepRL are currently model-based methods where a distribution over the possible models of the environment is used to recover uncertainty estimates over the value function or ensemble methods where multiple value functions are trained to give an estimate of the variance of these estimates. While both methods have achieved some remarkable results in various domains, model-based methods require learning multiple models of the environment, increasing the computational domains of the methods and adding the complexity of learning the model, while ensemble methods are hard to justify theoretically.

In this dissertation, we tackle the exploration problem in RL by proposing Wasserstein TD-Learning (WTD), a novel framework that directly models the uncertainty over the value function estimates in a model-free manner and propagates it across the state-action space by employing inexpensive variational updates that allow us enough control over the updates to show some desirable theoretical properties in the tabular setting while allowing the method to be easily scalable in the DeepRL setting. The extension of WTD with function approximators allows us to generalize the uncertainty estimates across complex high-dimensional state-action spaces. This allows us to adapt WTD in a multitude of different settings by adapting algorithms from the literature to handle the distributional nature of our value function, allowing for deep and directed exploration.

Contents

CHAPTER 1

Introduction

When starting to write this dissertation, we took some time to decide on the initial sentence of the introduction. "It should be interesting", we thought. It should capture the reader, while also setting forth a train of thought that allows us to effectively express all the ideas presented in this work. The goal of the dissertation was clear. The tools at our disposal too, i.e., the English language vocabulary. Good then, we can now write the whole dissertation, we just need to decide the complete sequence of thousands of words that will effectively convey to the reader the information developed during our work. Now, if we actually took this approach to writing, we would probably require another four years to write about the work we developed in four years. Deciding beforehand the whole sequence of words of the dissertation would require us to *plan* ahead over an extremely large horizon, where in each step of this horizon we have a choice between a large list of words. Obviously, we opted for a different approach. We started with an "interesting" (hopefully interesting to the reader too) sentence and then continued the flow of ideas with a general goal in mind. This allowed us to write much faster, stopping from time to time to think ahead the next sentences or to review what we wrote before and correct some ideas if needed. Many times we *explored* a flow of thought that we considered promising but we did not know how to develop it further after some time, so

we came back and started rewriting, exploring other flows. In this way, we completed this dissertation, conveying to the reader all the information we aimed to convey, in a clear, straightforward manner. The complete dissertation might not be the best manuscript we could have written for our purposes. We might have written a clearer more concise thesis if we had taken the first approach of planning the text ahead without writing a single word first. We chose the second approach, opting for an approximate solution to our thesis writing problem to keep writing times manageable. We used our intuition to evaluate our writing at intermediate sections of the task to decide how to continue this chain of thought. Some times our intuition proved wrong and we had to go back and rewrite.

In our writing example, we already have all the information we need. We know all the information that needs to be conveyed to the reader, we know our contribution, the fundamentals of the field, the literature we build upon, and our conclusions. We learn no new information while writing. Furthermore, there is no uncertainty. We are not conversing with other people in this dissertation, so there is no uncertainty on their reply to our writing. Despite this complete information about the task, it was still beneficial to explore the solution space to reach a good enough solution in a reasonable time.

In our daily lives, we tackle a multitude of other sequential tasks where we do not have complete information. Consider the example of finding a parking space after reaching our destination, say a restaurant. After reaching the restaurant, we would like to find a parking space as close as possible, so that we minimize the walking time to and from our car. If all the parking spaces outside our destination are taken, we could just wait until one of them frees up, but we do not know how long that will take. Alternatively, we could continue looking for a space along the road. We reach a crossroads without finding a space. Should we turn left, right, or continue straight? Where are we most likely to find a space? Have we parked around here before? Was it at this time, at this day of the week? How much previous information can we use to inform this decision? We can see on the right side many empty spaces along the street. We turn right thinking we found our parking spot. Then we see that at each of those free spaces, there is a sign that says that parking is only allowed for residents of this street. We do not live around here so we cannot park. We could not have seen the signs from the crossroad. Differently from the writing example, we cannot just go back and turn left instead (driving backward on a single-lane street while technically possible is still not advisable). We need to continue in the street and hope we make a better decision (or be luckier) at the next crossroads. Meanwhile, we keep in mind that this street was not a good option to explore for the next time we are at that crossroad. We continue this exploration until we find a suitable parking spot and then walk to our destination. While finding the parking spot we took

many streets around the area. Some we had driven before, and some were new. We learned while doing the task at hand. This will be useful the next time we need to find parking in this area. We increased our knowledge of the area (we know not to try to find parking on the right of that first crossroad) but there are still many streets that we do not know. Even if we knew all of them, we still do not know how many (if any) free spaces are there going to be at any given time. Our next parking time will still be uncertain, but we have more information to exploit for next time. We enter the restaurant knowing next time we will still probably need to explore the area for 15 minutes before parking. Maybe we should leave our house a bit earlier to account for that.

These two examples share some characteristics with many tasks we face in our society. Sequential goal-maximizing decision-making tasks (whether with our without uncertainty) are ubiquitous. With the advances in technology, more and more of these tasks are being delegated to complex computer systems, shaping a society that leans heavily on computers for decision-making and action execution. The classical approach for building these computer systems has been to heavily shape their behavior, determining a fixed behavior in advance in response to environmental conditions. This approach yields agents that fail to adapt to unforeseen situations limiting their application to highly controlled environments. To be able to tackle complex tasks in the real world, autonomous agents need to be able to replicate our ability to learn and adapt to unforeseen situations. Learning agents need to continuously make decisions that impact their future, be able to reason over the long-term effects of their actions, and be proactive. Self-driving cars, with different levels of autonomy, are already on our streets. While they have access to a great amount of information, still they are far from being completely autonomous. In our parking example, while an autonomous car would have access to the map of the area, hence knowing not to take that right turn with the resident-only parking, it would still struggle more than us to find parking, because accounting for all the uncertainty in busy city streets is practically impossible beforehand. New solutions need to be discovered in real time, even when solutions to similar problems are known. Moreover, our agents need not only to estimate the value of possible approximate solution, but also their *uncertainty* on these values to be able to better determine which candidate solutions to explore next.

The writing and the parking example are obviously not the focus of this work. They represent useful examples of the kind of tasks we are interested in solving automatically. Reinforcement Learning (RL, Sutton and Barto, 2018) represents one of the most useful frameworks for solving sequential decision-making tasks where the optimal solution is not known apriori but a limited feedback signal is given as a response to the agent's actions. RL agents observe the *state* of the environment, which contains all the available information that is relevant to the task and the environment. The agent makes a decision

by executing one of the available *actions*, observes the evolution of the environment in the next state and receives feedback in the form of a scalar *reward* signal. By repeating this process, RL agents aim to maximize the (possibly discounted) sum of the gathered rewards during their activity and discover the *optimal* behavior as fast as possible. While learning the optimal behavior, RL agents will iterate many partial approximate solutions of the task. They continuously face the choice of whether to trust these partial solutions and follow them or explore new possible strategies with the hope that they are better than the previous strategies, or with the hope of gaining new knowledge of the environment to be used in the future. Going back to our parking example, do I turn left where I have found a parking space in the past or continue straight where I might find a spot even faster? In our writing example, given what I have written so far, what should I write next? How can I evaluate what I have written so far? These are instances of the famous *exploration-exploitation* dilemma (Sutton and Barto, 2018), which is still one of the most important open problems in the field. While we will not provide a definitive solution to this problem, we will make several contributions that bring us closer to autonomous learning agents. In this dissertation, a great focus will be given to devising methods that allow autonomous agents to estimate not only the value of candidate solutions but also our uncertainty over these estimates. By having estimates of how uncertain they are in any given situation, our agents can better decide which solutions to explore next. Going back again to our writing example, not only will we be able to evaluate our text partially with our "intuition" but we will also have some estimates of when to trust our intuition and when not to trust it.

1.1 Motivation

Reinforcement Learning has achieved incredible results in recent years in solving complex sequential decision-making tasks like video-game-playing (Mnih et al., 2015), board games (Silver et al., 2017a), robotics (Liu et al., 2022), autonomous driving (Likmeta et al., 2020) and financial trading (Bisi et al., 2021). These results have been achieved both in tasks where learning is done online while performing the task with no additional information on the environment (like in our parking example) or when a model of the environment is available and can be used to simulate experience before acting (like in our writing example). The combination of RL with deep learning allowed researchers to tackle many simulated or real-world tasks, where the number of states and actions is too large (or infinite) to represent solutions exactly. The combination of Deep RL with search methods like Monte Carlo Tree Search(MCTS, Browne et al., 2012) allowed the extension of these methods to domains with abysmally large search spaces like the game

of Go, previously thought as an unsolvable problem with the current methods. Despite these remarkable achievements, the application of RL to solve sequential decision-making problems still faces challenges.

The main challenge for the application of RL algorithms is the large *sample complexity* of current algorithms, i.e., the amount of experience needed to reach a desired level of performance. Deep RL algorithms are extremely data-hungry, and in RL, this data is usually generated online while interacting with the environment. A multitude of techniques have been proposed for increasing the efficiency of RL agents like pre-training agents with simulated environments, reusing samples from similar tasks through transfer learning (Zhu et al., 2023), training general agents in a multitude of similar tasks through meta-learning (Finn et al., 2017b) or maximizing the usage of the available experience samples through the study *update-to-data* ratio (Dorka et al., 2023). In this work, we focus on methods that decrease the sample complexity of RL agents by exploring the environment more efficiently.

The effective exploration problem has been extensively studied in the RL literature. Efficient exploration requires striking a balance between minimizing learning time and maximizing the agent's performance during the learning process. This balance necessitates a certain degree of exploration to uncover effective behaviors. However, identifying an optimal exploration strategy is a challenging task, given that what proves effective in one environment might not be suitable for another. The impact of exploration techniques on learning time varies significantly, highlighting the crucial need for adaptive and environment-specific strategies. RL researchers typically focus on tabular settings first, where the theoretical properties of algorithms are more tractable. These algorithms are subsequently extended for use with general function approximators, although the theoretical guarantees do not hold anymore. This emphasizes the need for general algorithms that are theoretically grounded but also easily scalable to the Deep RL setting.

A multitude of frameworks have been developed to improve the exploration of RL agents including *pseudo-counts* (Ostrovski et al., 2017), *intrinsic motivation* (Bellemare et al., 2016), *entropy maximization* (Mutti et al., 2021), *ensemble methods* (Osband et al., 2016a), *distributional RL*(Bellemare et al., 2017b). We will focus on methods that explore efficiently through the use of *epistemic uncertainty* estimates, i.e., measures of how confident we are on the estimated performance of the current (or optimal) policy. In this work, we introduce Wasserstein Temporal Difference learning (WTD), a novel framework for representing epistemic uncertainty, propagate it across the state-action space and exploit it to perform deep and directed exploration. We provide a provably efficient algorithm based on this framework in the tabular setting and then focus on

several extensions of the framework to tackle problems with large state spaces, continuous action spaces and settings when a forward model of the environment is available. This way, we make several contributions toward our goal of having autonomous agents that explore their environments more efficiently by exploiting estimates of the uncertainty of their current and future strategies.

1.2 Contributions and Overview

In Chapter 2, we start by introducing the fundamentals of Reinforcement Learning and planning for solving sequential decision-making problems. We spread the discussion over relevant background and literature on the use of uncertainty estimates in Reinforcement learning and efficient exploration in Chapter 4, Chapter 5, and Chapter 6 where we focus on the specific settings of tabular domains, domains with discrete actions, continuous control, and model-based methods.

Our original contributions start from Chapter 3, where we illustrate the exploration problem in Reinforcement Learning by studying the performance of modern Deep RL algorithms like AlphaZero in goal-directed tasks where the reward function is generally sparse and uninformative. We propose a simple solution to improve the sample complexity of the algorithm in this setting, based on the combination of AlphaZero with Hindsight Experience Replay (HER, Andrychowicz et al., 2017) and study its performance empirically in a number of simulated domains, including a novel application to quantum compiling. The material of this chapter has been published as a conference paper in ICLR2022 (Moro et al., 2022).

In Chapter 4, we take a step back and tackle the exploration problem in the model-free setting. We propose a novel framework, Wasserstein Temporal-Difference (WTD) learning that represents the epistemic uncertainty over the value function by means of parametrized posteriors over the possible value functions and propagates this uncertainty across the state-action space by employing barycenters in Wasserstein space to average the posteriors to account for aleatoric uncertainty. Furthermore, we present Wasserstein Q-learning, a model-free RL algorithm based on the WTD framework, and prove some desirable theoretical properties of WQL related to its sample complexity in tabular environments. Moreover, we propose a simple extension of the algorithm to the setting of large or continuous state spaces, where the use of function approximators is required, by combining it with the DQN algorithm (Mnih et al., 2015). Finally, we perform a thorough experimental campaign demonstrating the exploration capabilities of the proposed algorithm. The material of this chapter has been published as a conference paper in NeurIPS2019 (Metelli et al., 2019).

We continue in Chapter 5, where we extend the WTD framework to the setting of continuous control. We introduce Wasserstein Actor-Critic, a model-free algorithm that extends Soft Actor-Critic (SAC, Haarnoja et al., 2018) by employing distributional critics to represent and propagate epistemic uncertainty and exploit it to perform deep exploration by optimizing an optimistic objective. Moreover, we consider some practical issues that arise when generalizing uncertainty in the WTD framework and propose a simple regularization of the uncertainty estimates that proves effective in several hard exploration tasks. The material of this chapter has been published as a conference paper in AAAI2023 (Likmeta et al., 2023).

In Chapter 6, we present our final contribution, where we combine the WTD framework with MCTS to propose Wasserstein AlphaZero. We highlight the main differences compared to the base AlphaZero framework, related to the use of temporal difference estimates instead of Monte Carlo (MC), and observe empirically how this affects the exploration capabilities of the agent in several simulated domains.

We conclude our dissertation in Chapter 7, where we recap our contributions and discuss some interesting future directions related to the use of epistemic uncertainty in reinforcement learning.

CHAPTER 2

Reinforcement Learning and Planning

In this chapter, we introduce the fundamentals of Reinforcement Learning (RL) and planning as frameworks for solving sequential decision-making problems. We will restrain ourselves with a brief introduction of the main solution concepts and methods and refer the readers to (Puterman, 1994) for a more thorough read.

Reinforcement Learning is an integral framework of machine learning, heavily inspired by behavioral psychology. It focuses on determining the optimal behavior for *agents* within their environment to maximize a predefined utility function. The agent engages in exploration of its surroundings, gathering data from interactions to inform future decisions. This process can be viewed as similar to the way a baby explores its environment and plays in a playground. The driver of the agent's actions is the reward signal it receives from the environment. In the context of a baby exploring a playground, the motivation to engage in activities is tied to humans' intrinsic drive to understand and explore their surroundings—an internal, or intrinsic, reward signal. External, or extrinsic, reward signals are evident in scenarios such as teaching rats to navigate a maze for food upon reaching the end. Regardless of whether the reward is intrinsic or extrinsic, the reward function defines the learning task for the agent (Sutton and Barto, 2018). While we typically assume the agent receives rewards after each action, in complex situations,

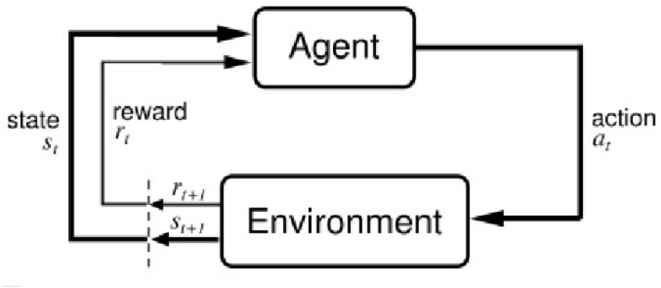


Figure 2.1: The Reinforcement Learning framework Sutton and Barto (2018).

actions may impact not only immediate rewards but also future ones. Consequently, the agent must possess the capability to connect environmental sensations (states) to actions, considering the potential repercussions on future rewards. The MDP framework is particularly suitable for formalizing such problems. The whole process is formalized in Figure 2.1.

Reinforcement Learning stands in stark contrast to Supervised Learning(SL). In SL, the training process relies on a labeled set of samples provided by an external authority, hence the term "supervised". Each sample encapsulates a scenario along with a specification of the system's correct behavior in that context. The objective is to generalize from this training dataset, enabling the system to apply appropriate behavior in novel situations beyond the training set. While crucial in machine learning (and as we will see also to reinforcement learning), supervised learning alone falls short in addressing control problems. In sequential decision-making problems, the agents need to identify the best possible behavior without observing it beforehand in a fixed dataset. Agents must possess the capability to generate their own datasets and learn from their own experiences. This leads us to a key aspect that sets reinforcement learning apart from other machine learning paradigms: the *exploration vs exploitation dilemma* (Sutton and Barto, 2018). It stands as one of the most significant challenges in the realm of reinforcement learning. Agents are tasked with selecting the "best" actions in each situation. To assess the value of these actions, they leverage knowledge gained from interacting with the environment. However, to uncover the optimal actions, they must explore the environment by trying out new actions. Exclusive reliance on either exploration or exploitation is inadequate for achieving the goal of maximizing rewards. Balancing the two becomes crucial. This predicament becomes even more crucial in stochastic tasks, where actions need to be attempted multiple times before a reliable estimate of their value can be obtained. Although extensively studied for decades, it still remains an open problem. The question of how to explore better while learning (or planning) is going to be crucial in this thesis, and we will introduce novel methods to explicitly quantify and leverage the agent's

uncertainty regarding the utility of its actions.

Reinforcement Learning seeks a global solution to control problems by learning a solution that performs well across the entire state space. While this appears desirable, finding a solution that generalizes effectively across diverse state regions can be challenging. Monte Carlo Tree Search (MCTS, Browne et al., 2012) algorithms have demonstrated remarkable efficacy in addressing sequential decision-making challenges, particularly in deterministic transition tasks like games. MCTS planners leverage a forward environment model to construct a search tree, assess the value of each action in the current state, and execute the best-estimated action. This process allows for the derivation of a "local" solution at each decision step by sampling potential future behaviors through the forward model. While this localized approach can be advantageous in certain contexts, it incurs a significant computational burden, as acting in the environment necessitates interleaved planning phases and potentially large search trees. This family of algorithms is based on some of the results presented in this thesis, so we formalize the method in this chapter.

In the remainder of the chapter, we start by giving a brief introduction of Markov Decision Processes in Section 2.1 as the mathematical tool used to formalize sequential decision-making and define the concept of solving an MDP. We continue by introducing solution methods when the model of the MDP is known in Section 2.2. Section 2.3 discusses model-free solution methods. Finally in Section 2.4, we formalize MCTS as one of the main "hybrid" methods where a model of the MDP is available but can only be used to generate samples.

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) offer a mathematical framework for modeling sequential decision-making problems in scenarios where outcomes result from a combination of randomness and the controlled actions of a goal-directed agent. This agent interacts with an environment, making decisions and receiving sensory feedback. This thesis focuses on discrete-time MDPs, where the agent's task involves selecting actions in each state to maximize their utility function. At each time step, the agent perceives the current state of the environment. Upon taking an action, the agent then observes the resulting state and receives an immediate reward associated with this transition. This immediate reward is a crucial element in decision-making. Typically, the agent's utility function involves a cumulative reward calculated over an extended time frame. While maximizing immediate rewards is important, it may not suffice for maximizing cumulative rewards. Consequently, the agent often needs to make decisions that involve sacrificing immediate gains to secure long-term benefits.

More formally: A Markov Decision Process is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu, \gamma)$, where:

- $\mathcal{S}$ is a the set of states , called *state space*;
- $\mathcal{A}$ is the action set , called *action space*;
- $\mathcal{P}$ is a function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathcal{S})$[1] called *transition model*, that for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}$, assigns $\mathcal{P}(\cdot \mid s, a)$, a probability measure over $\mathcal{S}$;
- $\mathcal{R}$ is a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathbb{R})$ called *the reward function*, that for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}$, assigns a probability measure over $\mathbb{R}$, $r(s, a) = \mathop{\mathbb{E}}_{R \sim \mathcal{R}(\cdot|s,a)}[R]$ is the state-action expected reward;
- μ is a distribution over states in $\mathcal{S}$, called the *initial state distribution*;
- $\gamma \in [0, 1]$ is the *discount factor*.

The agent's sensor and actuator capabilities are defined by the state space $\mathcal{S}$ and action space $\mathcal{A}$. These spaces can take on either finite or infinite, discrete or continuous forms. The immediate payoff is modeled by means of a scalar reward that, given the current state s and the current action a, assigns a scalar reward sampled from $\mathcal{R}(s, a)$. $R(\cdot|s, a)$ is the corresponding density function. In most common applications the state-action expected reward $R(s, a)$ is used, defined as the expected value of the reward taken over all next states s' and all real rewards r. We assume that the reward function is upper bounded in absolute value, i.e., $\| R \|_\infty = \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \mid R(s, a) \mid \leq M < +\infty$.

Time is represented as a discrete set of time steps $\mathcal{T} = 0, 1, \ldots, T$, where T is referred to as the *horizon*. The horizon can be either finite ($T \in \mathbb{N}$) or infinite ($T = \infty$). In the former case, the MDP is termed *finite horizon*, while in the latter case, it is labeled *infinite horizon*. An MDP is episodic if its state space includes a *terminal* (or absorbing) state, denoted as s, from which no other states can be reached for any action $a \in \mathcal{A}$ ($P(s|s, a) = 1$). It is conventionally assumed that actions performed in a terminal state yield zero reward.

The discount factor γ, typically set to a value strictly less than 1, plays a crucial role in attenuating the weight of rewards perceived in the distant future. While this is particularly relevant in infinite horizon MDPs, it can also prove beneficial in finite horizon scenarios. An illustrative example involves financial applications, where receiving monetary rewards in the present generally holds more value than receiving them in the future, owing to interest rates.

To summarize, the dynamics of MDPs unfold as follows. The process begins with an initial state $s_0 \sim \mu$. At each time step, the agent selects an action from the available set

[1] Given a set $\mathcal{X}$, we denote with $\mathscr{P}(\mathcal{X})$ the set of all probability measures over $\mathcal{X}$

of actions $\mathcal{A}$. Following the transition model $\mathcal{P}(\cdot \mid s, a)$, the agent transitions to state s' and observes the reward signal r. By iteratively repeating these actions, the agent moves through a sequence of states, accumulating a series of rewards. The agent's objective is to maximize the discounted cumulative sum of these rewards.

2.1.1 Policies

The set of rules that map an agent's history to actions are referred to as *policies*. Due to the Markovian property inherent in Markov Decision Processes (MDPs), the current state s_t alone suffices to determine the subsequent action a_t (Sutton and Barto, 2018) for most objectives. Such policies are termed Markovian Policies. Additionally, if the actions mapped are independent of the time step t, the policy is labeled as stationary. For simplicity in this thesis, when we mention the term "policy", we specifically refer to Markovian Stationary Policies. More formally:

A Markovian Stationary Policy π is a function $\pi : \mathcal{S} \to \Delta(A)$ that for every state $s \in \mathcal{S}$ maps a probability distribution, $\pi(\cdot \mid s)$, over the action space $\mathcal{A}$. If the policy is deterministic then $\pi : \mathcal{S} \to \mathcal{A}$.

The objective of reinforcement learning (loosely defined) is to identify the policy that maximizes the accumulated reward. An MDP combined with a policy π gives rise to what is commonly referred to in the literature as a "Markov Reward Process," denoted by the tuple $(\mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \mu, \gamma)$. For all states $s \in \mathcal{S}$, $\mathcal{P}^\pi(\cdot \mid s)$ represents a probability measure over $\mathcal{S}$. P^π is the probability density function derived by marginalizing the transition model of the MDP over the actions:

$$P^\pi(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) P(s'|s, a), \qquad \forall s, s' \in \mathcal{S}. \tag{2.1}$$

$P^\pi(s'|s)$ gives the probability of ending up in state s', starting from state s, in one time step. Similarly,$\mathcal{R}^\pi(\cdot|s, s')$ for all $s, s' \in \mathcal{S}$ is a probability measure over $\mathbb{R}$ with the corresponding density function R obtained from the reward model of the MDP as:

$$R^\pi(r|s, s') = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}(r|s, s', a), \qquad \forall s, s' \in \mathcal{S}. \tag{2.2}$$

(MRPs) serve as effective models for uncontrolled processes where the concept of action is irrelevant. They find utility in representing stochastic processes occurring in the environment, yielding rewards, e.g., evaluation of a fixed policy π. When the notion of reward is eliminated from MRPs, as previously noted, we are then left with Markov Chains or Markov Processes.

Value Functions

As discussed earlier, the goal of reinforcement learning is to find the optimal policy the agent can play to maximize the sum of rewards in the long term. The most trivial way to do this is to list all the possible behaviors the agent can exhibit and choose the one with the highest utility. Fortunately, we can do better than this. A better way is to use the value function of each state, defined as : The value function in state $s, V^{\pi}(s)$, of an MDP is the expected return starting from state s and following policy π:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) | s_0 = s\right], \qquad \forall s \in \mathcal{S}, \tag{2.3}$$

where the expectation is taken w.r.t. trajectories generated by following policy π starting from state s. The problem of finding the optimal policy becomes finding the optimal value function and determining the optimal behavior from it.

For control purposes, it is often more beneficial to define the action-value function, $Q^{\pi}(s, a)$, where instead of taking the first action from policy π, we fix it to a, and then follow the policy for the remainder of the trajectory. Formally: The action-value function in state s executing action a, $Q^{\pi}(s, a)$, of an MDP is the expected return starting from state s, applying action a and then following policy π:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right], \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.4}$$

There is a clear relationship between the value function and the state value function. The former is computed by averaging the latter over the actions taken by policy π:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)}\left[Q^{\pi}(s, a)\right], \qquad \forall s \in \mathcal{S}. \tag{2.5}$$

Sometimes it is useful to estimate the *advantage function*, $A^{\pi}(s, a)$, which represents how much a given action, a, is better in state s, w.r.t., the average utility of that state. $A^{\pi}(s, a)$ is defined as:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s), \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.6}$$

Bellman Equations and Operators

We defined the state value function, $V^{\pi}(s)$, as the expected return collected starting from state s and then following policy π. The value function also admits a recursive definition, decomposing it further as the sum of the immediate reward in state s with the expected discounted reward in the following state. This recursive definition will show

itself as useful in solving MDPs. The *Bellman Expectation Equation* (Bellman, 1957) is defined as :

$$\begin{aligned} V^{\pi}(s) &= \mathop{\mathbb{E}}_{\pi}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi}(s') \right). \end{aligned} \tag{2.7}$$

The action-value function can be decomposed in the same way as :

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

When the MDP is finite, by using the *Markov Reward Process* induced by policy π, we can write the Bellman Expectation Equation in a concise matrix form of the equation as follows:

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi}, \tag{2.9}$$

which yields the solution:

$$V^{\pi} = \left(I - \gamma P^{\pi}\right)^{-1} R^{\pi}. \tag{2.10}$$

While this is an exact solution of the MDP , unfortunately, inverting the matrix $(I - \gamma P^{\pi})$ has high computational complexity. A way to solve the MDP while avoiding the complexity of the matrix inversion is to use the *Bellman Expectation Operator* (Bellman, 1957) defined as $\mathcal{T}^{\pi} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|)}$:

$$(T^{\pi} V)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right), \qquad \forall s \in \mathcal{S}. \tag{2.11}$$

This operator maps value functions to value functions. It can be shown that the operator is a γ-contraction and the value function V^{π} is the unique fixed point of the operator, T^{π} , i.e., it satisfies $T^{\pi}[V^{\pi}] = V^{\pi}$ (Puterman, 1994).

We can define the Bellman Expectation Operator for the action-value function as well ,$T^{\pi} : R^{|\mathcal{S}| \times |\mathcal{A}|} \to R^{|\mathcal{S}| \times |\mathcal{A}|}$, defined as:

$$(T^{\pi} Q^{\pi})(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^{\pi}(s',a'), \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{2.12}$$

Like for the value function, Q^{π} is the unique fixed point of T^{π} , i.e., $T^{\pi}[Q^{\pi}] = Q^{\pi}$.It is worth noticing that both operators are linear. Furthermore, they satisfy the contraction property in $\mathcal{L}_{\infty}$ norm, i.e., $\|T^{\pi} f_1 - T^{\pi} f_2\|_{\infty} \leq \gamma \|f_1 - f_2\|_{\infty}$, thus the repeated application

of T^π makes any function converge to the value function. Value functions are very important in reinforcement learning. They define a partial order over policies. Namely for any two policies π, π':

$$\pi \geq \pi' \quad if \quad V^\pi(s) \geq V^{\pi'}(s), \qquad \forall s \in \mathcal{S}.$$

2.1.2 Optimality Conditions

We have stated repeatedly that the goal of reinforcement learning algorithms is that of identifying the *optimal* policy of the MDP, i.e., the policy that maximizes the agent's utility function. It can be shown that such optimal policy, is endowed by the *optimal value function*. The optimal value function maximizes the expected return for every state.

If Π is the set of all possible Markovian stationary policies then the optimal value function,V^* is defined as:

Given an MDP $\mathcal{M}$, the optimal value function in any state $s \in S$ is given by :

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s), \qquad \forall s \in \mathcal{S}. \tag{2.13}$$

The optimal value function specifies the best possible performance an agent can reach in any state of the MDP. Similarly we can define the optimal action-value function, $Q^*(s,a)$ as:

Given an MDP $\mathcal{M}$, the optimal action-value function in any state-action pair $(s,a) \in S \times A$ is given by :

$$Q^*(s,a) = \max_{\pi \in \Pi} Q^\pi(s,a), \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{2.14}$$

The optimal value function also accepts a recursive definition:

$$\begin{aligned} V^*(s) &= \max_{a \in A} Q^*(s,a) \\ &= \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right). \end{aligned} \tag{2.15}$$

Note that there is no specific dependence on a policy π. The Bellman optimality operator, $T^* : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ is defined as:

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

The optimality operator has the same properties as the Bellman Expectation Operator mentioned before; namely, it is a contraction w.r.t. $\|\cdot\|_\infty$, and the optimal value function V^* is a unique fixed point of the operator. Differently from the Expectation operator, it is not a linear map, so it does not accept a closed-form solution in matrix form as in Equation 2.10.

Optimal Policies

After establishing the optimal value function and its significance, the question arises: is there any policy, denoted as π^*, whose associated value function aligns with the optimal value function? (Puterman, 1994) gives us the following result:

For any Markov Decision Process

- There exists an optimal policy π^* that is better than or equal to all other policies $\pi \in \Pi$;
- All optimal policies achieve the optimal 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 MDP.

The last result is extremely important. As a result of the theorem above we can find the optimal policy of an MDP by finding the optimal Q function, Q^*, and taking the "greedy" action in each state. More formally :

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a), \qquad \forall s \in \mathcal{S}. \tag{2.17}$$

It is noteworthy that having knowledge of the optimal action-value function Q^* is sufficient for computing an optimal policy in a model-free manner. In contrast, obtaining the optimal value function V^* necessitates knowledge of the transition model $\mathcal{P}$. This is the main reasoning of Model-free Reinforcement Learning algorithms which are designed to estimate Q^* from trajectories and use it to derive the optimal policy.

2.2 Solving MDPs

Knowing all the elements of the tuple enables us to solve the MDP and determine the optimal policy without actually executing any actions in the environment (Mausam and Kolobov, 2012). This ability to resolve a decision-making problem without actively making decisions is referred to as "planning." In the subsequent sections, we will delve into the explanation of several key planning algorithms.

2.2.1 Dynamic Programming

The primary objective when solving an MDP is to identify an optimal policy. Specifically, our interest usually lies in finding a singular optimal policy rather than exploring the entire space of optimal policies. The conventional approach involves determining the optimal value function (or action-value function) and subsequently deriving the optimal

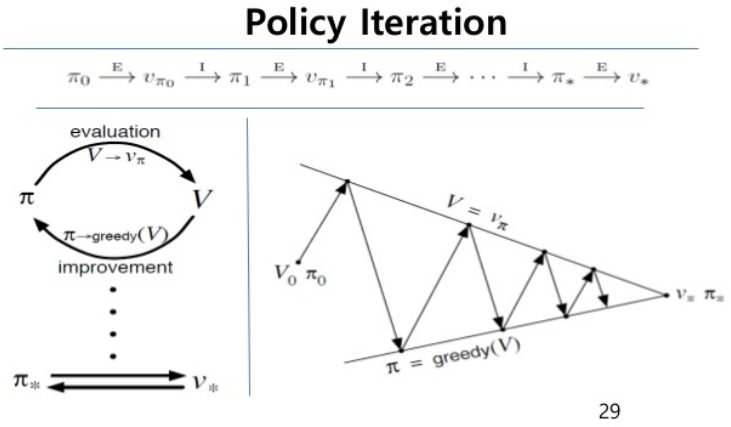


Figure 2.2: Policy iteration algorithm (Sutton and Barto, 2018).

policy from it. When the state transition model $\mathcal{P}$ is known, the prevalent method for solving MDPs is through *Dynamic Programming* (Bellman, 1957).

Dynamic programming is a widely employed technique for solving problems that can be broken down into subproblems. It is particularly effective for problems that exhibit two key properties:

- Optimal Substructure: The solution can be decomposed into subproblems.
- Overlapping Subproblems: Subproblems recur frequently, allowing solutions to be cached and reused.

This recursive approach aligns with the principles of the Bellman operator, as described earlier. Notably, MDPs possess both of these properties:

- The Bellman equation provides a recursive decomposition.
- The value function stores and reuses solutions.

A cornerstone algorithm of the dynamic programming family for solving MDPs is Policy Iteration. This method tackles the problem by iteratively engaging in *policy evaluation* and *policy improvement* phases (Howard, 1960). Figure 2.2 illustrates the Policy Iteration algorithm, which commences with an initial policy, π_0. The policy evaluation phase seeks to estimate the value function of the current policy. Multiple methods can achieve this, as discussed in the preceding section. While a closed-form solution (refer to Equation 2.10) offers precision, it comes with substantial computational costs. Alternatively, a recursive application of the Bellman Expectation Operator provides an approximation of the true value function for the policy by exploiting the contraction property. In many applications, the focus is on obtaining a sufficiently

accurate approximation of the value function. Indeed, this allows the algorithm to be applied also in settings where the full model is not available but only the ability to sample from it as well as in settings with infinite state spaces where approximate policy evaluation schemes can be employed. Following the policy evaluation phase, policy improvement generates the greedy policy from the value function. In each state, it selects the action maximizing the value function.

$$\pi^{t+1}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi^t}(s, a), \qquad s \in \mathcal{S}. \tag{2.18}$$

The policy generated is deterministic, and moreover, it is guaranteed to be an improvement over the previous one (Puterman, 1994). While we can perform the policy evaluation case even without access to the full model of the MDP, the policy improvement phase is different. When deriving the greedy policy for the next phase after policy evaluation, the state transition model $\mathcal{P}$ becomes a prerequisite. However, in scenarios where the model is not accessible, Q-iteration emerges as a viable alternative. In this approach, instead of evaluating the value function of the policy, we direct our focus toward the action-value function, undertaking a similar evaluative process. In summary, while policy iteration necessitates the availability of the state transition model for policy improvement, Q-iteration offers a solution in cases where the model is unavailable by concentrating on the action-value function with the added cost of estimating the value for all actions separately.

While policy iteration explicitly represents polices and iterates through them, Value iteration is a method that seeks the optimal value function without relying on intermediate policies. This technique relies on iteratively applying the Bellman Optimality operator, discussed in the preceding section by exploiting the contraction policy. The algorithm begins with an initial (random) value function, denoted as V^0, and iteratively applies the Bellman operator until a stopping condition is met. The stopping condition can be determined by reaching a maximum number of iterations or based on a minimal metric of the distance between two consecutive estimations of the value function. In the end, the optimal policy is obtained in a manner similar to policy iteration, by adopting the greedy policy induced by the optimal value function. It's important to note that, once again, the state transition model is required; otherwise, we need to estimate the optimal action-value function. The convergence to the optimal value function is guaranteed (Puterman, 1994). While policy iteration explicitly represents the policy, value iteration concentrates solely on the value function. This implies that intermediate value functions may not necessarily correspond to any policy. Both methods exhibit polynomial time complexity for Markov Decision Processes (MDPs) with a fixed discount factor Mansour and Singh (2013). In terms of a single iteration, policy iteration is computationally more demanding than

Algorithm 1 Value Iteration

1: **Input:** initial estimate Q_0
2: **for** $t = 0, 1, ...,$ until convergence **do**
3: $\quad Q_{t+1} \leftarrow \mathcal{T}^*(Q_t)(s,a) \qquad \forall s, a \in \mathcal{S} \times \mathcal{A}$
4: **end for**
5: $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q_t(s,a), \qquad s \in \mathcal{S}$
6: **Output:** π^*

value iteration since it necessitates evaluating the policy and performing the greedy improvement. However, policy iteration tends to converge in fewer iterations. Besides Dynamic Programming (DP), Linear Programming (LP) can also be employed to recover the optimal value function. Nevertheless, LP becomes impractical for a much smaller number of states compared to DP methods. the pseudocode for action-value iteration is given in Algorithm 1.

2.3 Reinforcement Learning

A significant drawback of Dynamic Programming (when employed for solving MDPs) lies in its dependency on the knowledge of the model. The necessity of the state transition model to resolve the MDP poses a limitation, as this model is frequently unavailable in real-life scenarios. Moreover, Dynamic Programming becomes impractical swiftly as the action-state spaces of the MDP expand, rendering it clearly unsuitable for infinite MDPs. In Reinforcement Learning we aim to transform Dynamic Programming algorithms into a sample-based nature. By adapting the algorithms in a sample-based manner, we can often marginalize the effects of the environment by sampling from it in the absence of the transition model.

2.3.1 A Taxonomy of Reinforcement Learning Algorithms

Reinforcement Learning, as a prominent paradigm within Machine Learning, encompasses a vast array of algorithms that can be classified in several dimensions:

- Model-based vs Model-free:
  - Model-based techniques involve estimating the state transition probabilities, denoted as $\mathcal{P}$. These methods first seek to estimate the transition dynamics and reward function and then apply Dynamic Programming to derive the optimal policy.

- In contrast, model-free techniques aim to directly discover the optimal policy from samples of trajectories without explicitly estimating the transition probabilities and reward function.

- On-Policy vs Off-Policy:
  - On-policy algorithms estimate the value function associated with the policy used to collect data (behavioral policy). Consequently, they apply some form of policy improvement steps to generate a new policy.
  - Conversely, off-policy algorithms learn the value function of a target policy (often the optimal policy) while using a different policy to collect the samples. This comes with additional issues related to the difficulty of evaluating the target policy from samples of one or several behavior policies. This generally makes off-policy algorithms more unstable than on-policy algorithms.

- Online vs Offline:
  - Online methods engage in learning while actively collecting samples. This requires the ability to be able to sample from the environment but generally makes the algorithm more robust since the policy collecting the data is the same (or at least similar in the off-policy case) as the policy being optimized.
  - On the other hand, offline methods conduct learning after acquiring all the data. This is generally done by one or more *behavioral* policies beforehand. While offline algorithms are considered safer as the behavioral policies are generally safe, they often struggle due to the limited exploration of the state action space and the inability to further explore and test the estimates created during learning.

- Value-based vs Policy-based vs Actor Critic:
  - Value-based algorithms focus on estimating the value function of the optimal policy and derive the optimal policy from it. No explicit policy is ever maintained by the algorithm.
  - Policy-based algorithms attempt to directly estimate the optimal policy by iterating multiple steps of policy optimization.
  - Actor-critic methods combine two components: an actor (or policy) that collects samples from the environment and a critic (value function) that estimates the value function of the actor. The actor also uses the estimates by the critic to perform policy improvement steps.

- Tabular vs Function Approximation:
  - Tabular methods explicitly store the value function for each state in a table. This approach is applicable solely in finite MDPs.
  - Function approximation algorithms leverage approximators such as neural networks to estimate the value function, allowing for more scalability in handling larger state spaces. This often comes with a loss of convergence guarantees to the optimal policy.

This short classification provides a basic understanding of the various Reinforcement Learning algorithms and their distinctive characteristics. In the following sections, we present some important model-free algorithms for prediction and control.

2.3.2 Policy Evaluation

Before delving into approximate optimization algorithms for solving MDPs, it is advantageous to introduce the concept of *model-free prediction*. This entails the accurate estimation, from samples, of the value (or action-value) function associated with a fixed policy π. Our focus will be on the *model-free* scenario, where the transition model of the MDP is neither known nor estimated. At this point, the process transforms into an MRP, considering the MDP in conjunction with a policy. Our attention will be directed towards two prominent classes of algorithms: *Monte Carlo approaches* (MC) and *Temporal Difference approaches* (TD) (Sutton and Barto, 2018). Both methods are able to estimate the value function of a policy directly from episodes of experience, despite lacking knowledge of the transition model of the MDP. This is achieved through iterative updates to their estimations of the value function in each state:

$$V^{t+1}(s_t) = V^t(s_t) + \alpha^t \left(V_t^* - V^t(s_t)\right), \tag{2.19}$$

where v_t^* is an estimation of the value function in state s_t and α^t is the learning rate.

The two approaches differ on how the estimator V_t^* is calculated. The Monte Carlo estimator employs a straightforward approach where the target value function is the sum of (discounted) returns over samples of a trajectory generated using policy π, given by π, $V_t^{MC} = \sum_{i=t}^{T-1} \gamma^t r_{i+1}$. However, this estimator comes with the drawback of being applicable solely to episodic MDPs, meaning that all episodes must conclude for the value function to be estimated. Additionally, the MC estimator encounters challenges when states are visited multiple times within the same trajectory. In such cases, a decision must be made whether to compute the value function solely for the first visit of the state (first-visit MC) or for every instance the state is visited (every-visit MC). The choice

between these options hinges on the classical bias-variance trade-off (Sutton and Barto, 2018).

On the other hand, TD makes use of bootstrapping by using the current approximation of the value function as a prediction target. The estimator now adopts the form of the *temporal difference target*, expressed as $V_t^{TD} = r_{t+1} + \gamma V^t(s_{t+1})$ resulting in the temporal difference update rule:

$$V_{t+1}(s_t) = (1 - \alpha_t)\, V_t(S_t) + \alpha_t \left(R_{t+1} + \gamma V_t(s_{t+1})\right), \tag{2.20}$$

This approach offers the added advantage of facilitating online learning methods and is not confined to episodic MDPs. Bootstrapping in TD can be applied after multiple steps, rather than a single step, resulting in the emergence of $TD(n)$ methods where n denotes the number of true samples before the application of bootstrapping. The selection of n plays a role in the bias-variance trade-off; higher values mitigate the bias introduced by bootstrapping but elevate the variance due to the utilization of longer trajectories from the environment. Moreover, multiple n-steps estimators can be combined into the $TD(\lambda)$ class of algorithms. For further details, we refer to (Sutton and Barto, 2018).

2.3.3 Model-free Control

The prediction problem involves estimating the value function for a fixed policy. What makes RL algorithms particularly useful is their capacity to learn an optimal policy of an MDP from samples of experience, constituting the *model-free control problem*. We are interested in the setting in which this experience is generated sequentially from the environment, either in a single stream of experience or by collecting multiple (possibly partial trajectories) Control algorithms derived from the methods discussed in the previous section, integrating improvement steps in a Dynamic Programming (DP) fashion. A key distinction from previous DP algorithms is the inability to execute the policy improvement step simply by selecting the optimal action based on the current estimate. In the context of learning from samples and starting with an initial (potentially random) estimate, unlike value iteration, we cannot update value estimates across the entire state-action space since we lack the ability to query the environment for state-action pairs of our choosing. Our sampling is restricted to complete trajectories originating from initial states drawn from the initial state distribution μ. Some algorithms also assume that trajectories can be interrupted and reset from another initial state. Consequently, challenging-to-reach states are visited only if the correct sequence of actions leading to them is generated during the learning process. Opting for the optimal action based solely on our current estimate risks (almost surely) converging to sub-optimal policies, as the samples collected depend on our estimation of the Q function. This presents an instance of the *exploitation*

vs exploration problem, i.e., utilize existing information (exploitation) or choose new actions to acquire fresh information (exploration), enabling exploration of uncharted areas in the state-action space.

One option involves employing an ϵ-greedy policy, which is a stochastic policy selecting a random action from the available options with a probability of ϵ and choosing the current optimal action with a probability of $1 - \epsilon$. The ϵ-greedy Policy Improvement Theorem (Sutton and Barto, 2018) establishes that the new policy represents an improvement over the previous one. As we accumulate samples, our confidence in the estimate of the Q function grows, allowing for a reduction in exploration. This can be achieved, in its simplest form, by defining a schedule for the *exploration rate* ϵ that converges to zero. The update rule for the Q function can now be expressed as:

$$Q^{t+1}(s_t, a_t) = Q^t(s_t, a_t) + \alpha^t \left(v_t^* - Q^t(s_t, a_t) \right). \tag{2.21}$$

Again, different algorithms differ on their estimate of the value function, v_t^*. MC uses the estimator $v_t^{MC} = \sum_{i=t}^{T(\tau)-1} \gamma^t r_{i+1}$, while TD as before uses $v_t^{TD} = r_{t+1} + Q^t(s_{t+1}, a_{t+1})$.

The observations made in the previous section regarding the application of these algorithms remain pertinent in the control setting. It is crucial to note that both Monte Carlo (MC) and TD are on-policy methods, implying that they estimate the value function of a policy while actively following it. This is especially significant during bootstrapping in TD, as it dictates the selection of the next action, a_{t+1}, appearing in the TD target. The on-policy version of the TD algorithm is the well-known SARSA algorithm (Rummery and Niranjan, 1994). Both algorithms come with convergence guarantees, provided they adhere to the Robbins-Monro conditions on the learning rate:

$$\sum_{t=0}^{\infty} \alpha^t = +\infty \quad \text{and} \quad \sum_{t=0}^{\infty} \left(\alpha^t\right)^2 \leq +\infty. \tag{2.22}$$

and each state-action pair is visited infinitely many times (Jaakkola et al., 1994; Sutton and Barto, 2018), i.e., ϵ remains larger than 0. A simple case that respects these conditions is $\alpha^t = \frac{1}{t}$.

We conclude this section with the introduction of Q-Learning (Watkins, 1989a), an off-policy algorithm that extends Value Iteration to the model-free scenario. Q-learning employs a sample-based version of the Bellman Optimality Equation, enabling the learning of the optimal action-value function without the necessity of playing an optimal policy. The update rule for Q-learning is expressed as:

$$Q^{t+1}(s_t, a_t) = Q^t(s_t, a_t) + \alpha^t \left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^t(s_{t+1}, a') - Q^t(s_t, a_t) \right). \tag{2.23}$$

The pseudocode of the Q learning algorithm is shown in Alg. 2. Q-learning will serve as the basis for some of the algorithms developed in this thesis, extending it with more

sophisticated exploration schemes that leverage uncertainty estimates on the action-value estimates maintained during learning.

Algorithm 2 Q-learning

Input: States $\mathcal{S} = \{1, \ldots, n_s\}$, Actions $\mathcal{A} = \{1, \ldots, n_a\}$, Reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, Learning rate $\alpha \in [0, 1]$, Exploration rate $\epsilon \in [0, 1]$, Discount factor $\gamma \in [0, 1]$, initial state distribution μ.

Output: Q function

```
Initialize Q : S × A → R arbitrarily
while Q is not converged do
    Start in state s ~ μ
    while s is not terminal do
        Select action a accoring to ε-greedy policy
        Take a and receive the reward r and the new state s'
        Q(s, a) ← (1 − α) · Q(s, a) + α · (r + γ · max_a' Q(s', a'))
        s ← s'
        Update learning rate α according to learning rate schedule
        Update exploration rate ε according to exploration rate schedule
    end while
end while
return Q
```

2.4 Monte Carlo Tree Search

Sequential decision-making problems can be solved with two main approaches in a machine-learning context: Reinforcement Learning (RL) and online planning. RL algorithms solve these problems by interacting with the environment, or in the case of Batch RL by optimizing a policy in a fixed dataset of demonstrations from the environment. In both cases, the algorithm returns a global solution to the problem, which might be unfeasible or too expensive to do when the state-action space is too large or it is hard to generalize from. Online planning has proved itself to be a viable alternative to recover *local* solutions of the MDP while also leveraging function approximators to generalize across the state-action space (Silver et al., 2017a). The most used family of algorithms in the Online Planning realm is Monte Carlo Tree Search (MCTS). In MCTS a generative model of the environment is used to *simulate* future trajectories before taking a step in the true model. Specifically, a 1-step model of the environment and use it in each step to construct a search-tree to evaluate the different available actions. In this section, we

provide the general scheme of MCTS, focusing on 2 specific algorithms that are built upon in this thesis.

The family of MCTS algorithms integrates tree-search techniques with Monte Carlo sampling to progressively construct a search tree depicting potential future scenarios. This tree is then utilized to derive an estimate of the optimal value for each action in the current state of the environment. These algorithms follow a structured process generally consisting of four distinct phases:

- Selection: Commencing from the root of the planning tree, a *tree policy* is iteratively applied until an unexpanded node (a node with unvisited children) is reached.
- Expansion: One or more successors of the reached node are incorporated into the tree. A common best practice is to include only the first newly visited node.
- Simulation: A Monte Carlo simulation (*rollout*) is initiated from the expanded node to provide an initial estimate of the node's value.
- Backpropagation: The values of the states visited during the tree traversal and simulation are backpropagated up the tree until reaching the root, consequently updating the pertinent statistics.

In this thesis, our focus centers on Upper Confidence Tree (UCT)(Kocsis and Szepesvári, 2006). UCT employs the well-known Multi-Armed Bandit (MAB) algorithm, Upper Confidence Bounds (UCB1)(Auer et al., 2002), as a tree policy. At iteration n, UCB1 selects the action that maximizes a high-probability upper bound of the action values according to:

$$a_n = \underset{i=1..K}{\arg\max}\, \overline{X}_{i,T_i(n-1)} + C\sqrt{\frac{2\log n}{T_i(n-1)}}, \tag{2.24}$$

where K is the number of actions, C is a constant that regulates the exploration-exploitation tradeoff, $T_i(n-1)$ is the number of times action i has been played up to time $n-1$ and $\overline{X}_{i,T_i(n-1)}$ is the average payoff observed from arm i. UCT recursively updates the values of the nodes from the leaf to the root of the tree, during the backup phase. The algorithm is proved to be *consistent*, i.e., it converges to the optimal policy in the limit. In addition to the convergence guarantees of UCT, the algorithm also enjoys some other properties that have made it highly popular in the literature. First, the algorithm is *anytime*, which means it does not need to know the planning budget in advance and can return its best guess at any moment. This makes it particularly attractive for real-time scenarios like race-strategy identification (Piccinotti et al., 2021) or financial applications (Vittori et al., 2022). Secondly, UCT is *asymmetric*, which

means that it iteratively builds the search tree by favoring the most promising regions of the tree while still giving a probability of selection to all branches. Such propriety is paramount in some applications where the search tree is extremely large. This, in fact, is also one of the reasons why UCT has no theoretical guarantees on its *sample complexity*. This asymmetric tree building is based on the confidence bound kept from the UCB algorithm employed in each node, which, in turn, is not valid in high probability and might delay the discovery of the optimal paths if the reward function is misleading in the top levels of the tree.

Initially designed for deterministic environments where the variance of the transition and reward functions is 0, UCT has demonstrated success in numerous applications such as games and industrial optimization problems (Browne et al., 2012). However, its application becomes challenging when the environment involves a continuous action space or stochastic transitions with infinite state spaces. Various extensions have been proposed to address these cases (Browne et al., 2012).

In scenarios with a continuous action space, the application of UCB in the search tree nodes is not feasible due to the infinite number of actions and, consequently, the search tree itself is infinite. Progressive Widening (PW) (Couetoux, 2013) is a commonly used technique for handling MCTS in infinite search spaces. With PW, actions are explored progressively as the node visitation counts increase, based on the intuition that nodes visited more frequently are more promising. Specifically, when a node is visited for the n-th time, if the number of children of a node $|\mathcal{C}_n(\mathcal{N})|$ exceeds n^α, where $0 < \alpha < 1$, a new action is explored by sampling from the action space. Otherwise, one of the previously selected actions is explored, often chosen according to UCB1. It is crucial to note that PW determines when to add new nodes to the search tree but does not specify which new actions to explore. Typically, new actions are uniformly sampled in the action space in the literature, although the empirical performance of the algorithms heavily relies on the chosen action sampling distribution. In problems involving continuous state spaces and stochastic transition models, where the true search tree is infinite, the open-loop planning approach (Lecarpentier et al., 2018) is commonly employed. In this context, the goal is to find the optimal sequence of actions at the root state of the tree without considering the states visited during the search. This transformation turns the infinite search tree of the original problem into a finite tree with a branching factor equal to the number of actions. This comes at the cost of optimal performance since we are looking for the solution to a simpler problem by averaging between the states visited by a fixed sequence of actions.

2.4.1 Leveraging Generalization in MCTS

AlphaZero (Silver et al., 2016) bridges the gap between RL and MCTS by incorporating a parametric policy, π_θ, and a value function, v_ρ, with the aim of exploiting the local solution property of MCTS while exploiting the generalization of the global RL solution. The policy and value networks guide the MCTS algorithm to influence tree search, while the resulting search tree is used to generate training targets for the value and policy networks.

Contrary to the conventional MCTS, AlphaZero introduces modifications in the selection and evaluation phases. During the former, it employs the policy π_θ to bias exploration based on policy recommendations. Additionally, AlphaZero replaces the traditional UCT algorithm with PUCT (Rosin, 2011), adjusting the selection according to:

$$a_n = \underset{i=1..K}{\arg\max}\, B(a_i) = \overline{R}_{i,T_i(n-1)} + C\pi_\theta(s, a_i)\sqrt{\frac{n}{1 + T_i(n-1)}}. \tag{2.25}$$

By combining the PUCT confidence interval with the probability assigned to a_i by π_θ, AlphaZero prioritizes actions with high probability and low visit count initially, gradually favoring actions with high values as the search progresses. In the latter phase, AlphaZero evaluates leaf nodes using estimates from the value network v_ρ, eliminating the need for expensive simulations with a default rollout policy.

During each time step t, AlphaZero conducts M search iterations of MCTS, starting from the current environment state s_t, using π_θ in the selection phase and v_ρ during leaf evaluation. After the search phase, policy targets p_t are constructed based on visit counts at the root of the tree:

$$\boldsymbol{p}_t(a_i) = \frac{N_{t,i}}{\sum_j N_{t,j}} \quad \forall i = 1, \ldots, K, \tag{2.26}$$

where $N_{t,i}$ represents the visit count of the i-th action in the root of the tree at step t, and K is the number of actions. Subsequently, a_t is sampled according to $\boldsymbol{p}_t$ and executed, observing the next state s_{t+1} and the reward signal r_t. In practice, $\boldsymbol{p}_t$ is often times constructed as a greedy policy over the action counts, executing only the most explored action. At the end of each episode, the collected rewards contribute to building the value network targets R_t. The triples $(s_t, \boldsymbol{p}_t, R_t)$ are then added to a replay buffer $\mathbb{B}$ used for network training.

In the original AlphaZero paper, the policy and value networks share the parameter vector θ, and AlphaZero minimizes the following objective function:

$$\underset{(s,\boldsymbol{p},R)\sim\mathbb{B}}{\mathbb{E}} \left[(R - v_\theta(s))^2 - \boldsymbol{p}^T \log \pi_\theta(s) + c\|\theta\|^2\right] \tag{2.27}$$

where c controls the amount of L_2 regularization. Despite a lack of theoretical guarantees on convergence, even in simple cases, AlphaZero has demonstrated success across a variety of complex tasks, particularly in board games such as chess and, most notably, Go (Silver et al., 2017a). As we will explore in subsequent chapters, despite the enormous practical success in games, the algorithm encounters challenges in exploration-heavy environments where sparse feedback is received during optimization

CHAPTER 3

Planning Under Sparse Returns

3.1 Introduction

Monte Carlo Tree Search (MCTS) (Browne et al., 2012) algorithms have shown outstanding results in solving sequential decision-making problems, especially in deterministic transition tasks, such as games. MCTS planners use a forward environment model to build a search tree, estimate the value of each action in the current state, and execute the best-estimated one, respectively. Such a procedure allows finding a "local" solution to the decision problem in every decision step by sampling trajectories of possible future policies using the forward model. Although providing local solutions is advantageous in some contexts, it comes at a high computational cost since acting in the environment requires interleaved planning phases and possibly large search trees. These high computational costs have hindered applying MCTS to larger problems, especially those with long planning horizons and huge tree-branching factors (number of actions). However, MCTS does not require a training phase and can be deployed immediately.

On the other hand, Reinforcement Learning (RL) (Sutton and Barto, 2018) aims to find a global solution to the control problem by learning a policy that performs adequately in the whole state space. While this may seem a more desirable outcome, it can often

be hard to produce a policy that generalizes satisfactorily across different state regions. For this reason, MCTS algorithms have achieved tremendous success in a wide range of tasks (Enzenberger et al., 2010; Ikehata and Ito, 2011). The AlphaZero (Silver et al., 2016) family of algorithms made it possible to use sequential planners like MCTS in more challenging environments, such as the game of Go, where the AlphaGo agent defeated the world champion, achieving super-human performance. AlphaGo was, in turn, defeated by AlphaZero, a version of the algorithm without any heuristic related to the game of Go. AlphaZero combines MCTS with the ability of RL algorithms to generalize across the state-action space by keeping a parametrized policy and value network. Specifically, the policy network biases the exploration during the tree search, and the value network estimates the value of the states corresponding to the search tree leaves, replacing the trajectory-based evaluation usually performed with an ad-hoc evaluation policy called *rollout* policy. On the other hand, the MCTS algorithm acts as a *policy improvement* step. It takes as input the current state, as well as the policy and value networks, therefore improving the parametrized policy and generating the samples used to train the networks in a supervised manner. Although a tree search is still required, the deployment is cheaper since it usually requires a smaller planning budget, thanks to the policy and value networks that bias the search by making it more efficient. The algorithm has been successfully applied to different games such as Go, Chess, and Shogi (Silver et al., 2017a) without game-specific heuristics. However, despite its success, the algorithm suffers from a high sample complexity, especially prominent in tasks with sparse reward functions. Indeed, sparse return tasks still prove to be a challenge for RL algorithms in general, and in this chapter, we will investigate this challenge for the AlphaZero algorithm. We will consider goal-directed planning as a testbed for the exploration capabilities of AlphaZero.

In goal-directed tasks, the agent aims to reach a goal state, and in general, the policies take both the current state of the environment and the goal state as input. Usually, the actual reward function in these problems is zero for any transition except the one to the goal state, which gives a positive reward. RL algorithms struggle to optimize sparse reward functions since it might be hard to reach goal states, or even practically impossible during exploration if the task is quite challenging. Therefore, there is no reward signal to guide the exploration. While in practice, in specific tasks, it is possible to use more informative reward functions, such as a distance from the goal state, such a choice is not possible for every task. Furthermore, it might also lead to sub-optimal solutions since a reward function based on a state distance might generate local-optima in the policy space (Grzes and Kudenko, 2009). Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) is a straightforward method to tackle the problem of sparse-reward functions in goal-directed tasks. HER can be used with any *off-policy* RL algorithm to extend

the training dataset for the value networks. Practically, whenever the goal state is not reached during a training episode, the states visited during the episode are used as alternative goal states and are fed to the network during training. It allows reward signals to be given to the value networks and generalize them to reach the input goal state.

In this chapter, we investigate and identify exploration issues with the current performance of AlphaZero in sparse reward tasks and we consider the problem of applying HER to AlphaZero to tackle goal-directed tasks. We provide a scheme that does not involve computationally heavy tree re-weighting procedures or high additional computational costs. Finally, we benchmark the method with simulated environments, and we show a novel application to quantum compiling, where it is extremely hard to find unitary gate sequences from a set to approximate arbitrary single-qubit unitary operators.

3.2 Preliminaries

3.2.1 Goal-Directed Reinforcement Learning

In this chapter, we focus on goal-directed Reinforcement Learning problems. We will provide a brief introduction of goal-directed MDPs. In such a setting, an autonomous agent interacts with an environment to maximize the sum of observed reward signals. Formally, in deterministic goal-directed MDPs transition models, the environment consists of a state space $\mathcal{S}$, a set of goal states $\mathcal{G}$ (that can also be equal to $\mathcal{S}$), an action space $\mathcal{A}$, a goal-state dependent reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$, a transition model $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, a probability distribution over $\mathcal{S}$ for the initial state $s_0 \sim \mu$, and a probability distribution over $\mathcal{G}$ for the goal state $s_g \sim \nu$.

The behavior of the agent is described by a Markovian stationary goal-dependent policy, $\pi : \mathcal{S} \times \mathcal{G} \to \Delta(\mathcal{A})$, which takes as input the current state s and the goal state s_g, and outputs a probability distribution over the actions in $\mathcal{A}$. At the beginning of each episode, an initial state $s_0 \sim \mu$ and a goal state $s_g \sim \nu$ are sampled. Then, at time step t, the agent observes the current state s_t, selects an action $a_t \sim \pi(s_t, s_g)$, observes the next state $s_{t+1} = \mathcal{P}(s_t, a_t)$ and it gets the reward signal $r_t = r(s_t, a_t, s_g)$.

The goal of the agent is to maximize the expected return $\mathbb{E}[R_t]$ defined as the expectation of the discounted sum of future rewards $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$ taken over the initial state and goal state. Given a policy π, the value of each state is encoded by the value function $V^{\pi}(s, s_g) = \mathop{\mathbb{E}}_{a_t \sim \pi}[R_0 | s_0 = s]$. Similarly, the action-value function is defined for each state-action pair, conditioning on the first action of the trajectories, $Q^{\pi}(s, a, s_g) = \mathop{\mathbb{E}}_{a_t \sim \pi}[R_0 | s_0 = s, a_0 = a]$. The goal of maximizing the return can be

expressed as finding an *optimal policy*, $\pi^* = \arg\max_\pi V^\pi(s, s_g), \quad \forall (s, s_g) \in \mathcal{S} \times \mathcal{G}$.

3.2.2 Hindsight Experience Replay

Hindsight Experience Replay (Andrychowicz et al., 2017) is a method of extending off-policy RL algorithms to improve sample efficiency even in the presence of sparse reward functions. HER requires parameterizing the reward, policy, and value as a function of the current and the goal state. The basic idea behind HER is to extend the replay buffer $\mathbb{D}$ after each episode $\{s_0, s_1, \ldots, s_T\}$ with the returns calculated based on a set of subgoals. While the main goal influences the agent's actions during training, it does not influence the state transitions. Consequently, we can generate additional training samples by considering a subset of the states visited during the episode as subgoals. This is extremely beneficial in cases with sparse rewards, such as a reward function of the type $r(s_t, a_t, s_g) = \mathbb{1}(s_{t+1} = s_g)$, where reward signals would be null until the goal state is visited by chance. However, HER enables the reward signals to be generalized across the state space.

3.3 AlphaZero with Hindsight Experience Replay

This section describes the AlphaZeroHER algorithm. AlphaZero has been successfully applied to challenging games such as Go and Chess with outstanding results. The complexity of these games stands in the vast state-action spaces and the highly sparse reward function, since the agents will only know at the end of the game whether they have won (a reward of $+1$), lost (a reward of -1) or drawn (a reward of 0). Although in the context of TD algorithms, Chess and Go technically fall under the definition of sparse reward environments, when considering returns observed at the end of the episode (Monte Carlo returns), such games have a clear reward function: the game's result is either a win, a loss or a draw. This reason may give the impression that AlphaZero doesn't suffer in a sparse reward setting. However, in goal-directed planning, the Monte Carlo returns are often sparse, in the sense that the whole episode might finish without a reward signal. This problem was also mitigated in the original AlphaGo Zero paper (Silver et al., 2017b), where the authors employed ad-hoc board evaluators to compute the Monte Carlo returns when episodes of Go were interrupted because they were too long. Due to the criticality of sparse return tasks, we extend AlphaZero with HER to tackle goal-directed tasks.

3.3.1 AlphaZeroHER

The basic idea of AlphaZeroHER, is to extend AlphaZero by using a goal-directed policy and value network and injecting HER into such a setting. Exploiting a goal-directed policy and value network in AlphaZero is effortless since the experience samples also include the goal state s_g. However, AlphaZero is not an off-policy algorithm since the value network is trained with the returns collected while playing the MCTS augmented policy. Moreover, it is not straightforward how to evaluate the new policy when considering a new goal $s_t \neq s_g$ without introducing additional prohibitive computational costs and tree re-weighting procedures. Ideally, we aim at estimating a new policy conditioned on a subgoal $s_t \neq s_g$. However, estimating such a policy requires building an additional tree since the policy targets in AlphaZero are a function of the action counts in the tree's root node, and this with an obvious additional computational cost. An alternative approach could keep the original goal's search tree and re-weight the statistics in the nodes conditioned on the reward function for the new goal s_t. However, such a procedure would require at least traversing the whole tree once, which would be computationally prohibitive for a large search tree. Such problems represent the main obstacle to the introduction of HER.

We propose a simple procedure that extends AlphaZero with HER without adding high computational costs. The basic idea is to neglect its on-policy nature, generating additional training samples at the end of each episode by sampling additional subgoals from the visited states. In fact, even though the MCTS augmented policy of AlphaZero did not reach the goal state, it successfully reached all the states visited during the episode. More precisely, we employ HER after finishing an episode of length T and having generated the sequence of states, reward, and policies, $\{s_t, \boldsymbol{p}_t, r_t\}_{t=1}^{T}$. The episode is retraced so that at each state s_t, M subgoals are sampled from future visited states, $\{s_i\}_{i=t+1}^{T}$. After selecting the subgoals states for state s_t, we need to compute these states' policy and value targets. This task is not straightforward since AlphaZero is an on-policy algorithm. If we computed, in some way, a different policy target from the actual policy played $\boldsymbol{p}_t$, these would generate a different sequence of state and rewards after timestep t, which are not available. We could use the built tree to evaluate these returns related to the subgoals, but these would come with heavy tree-reweighting schemes for an already computationally heavy algorithm such as AlphaZero. For this reason, we select as additional policy targets the policies played during the episode $\boldsymbol{p}_t$ since, although these are not the optimal policies that the MCTS agent would have played if the subgoal states were the goal during the search, they successfully reached the alternative goal states. Therefore, these samples still represent an improvement over the

Algorithm 3 AlphaZeroHER

Initialize memory buffer D
Initialize policy π_θ and value network v_θ
for epoch $= 1, \cdots, N$ **do**
  for episode $= 1, \cdots, M$ **do**
    experiences $\leftarrow \{\}$
    Sample initial state $s_t \sim \mu$
    while not done **do**
      $\boldsymbol{p}_t, a_t \leftarrow \text{MCTS}(s_t, \pi_\theta, v_\theta)$
      s_{t+1}, r_t, done $\leftarrow$ applyAction(a_t)
      experiences $\leftarrow$ experiences $\bigcup$ $(s_t, \boldsymbol{p}_t, r_t)$
      $s_t \leftarrow s_{t+1}$
    end while
    Store every experience $(s_t, \boldsymbol{p}_t, z_t)$ in D, where $z_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$
    for t in episode experiences **do**
      $G \leftarrow$ Sample k goals from future visited states s_j where $j > t$
      for $g \in G$ **do**
        $r_t^g \leftarrow r(s_t, a_t, g)$
      end for
      Store every $(s_t, \boldsymbol{p}_t, z_t^g)$ in D, where $z_t^g = \sum_{i=t}^{T} \gamma^{i-t} r_i^g$
    end for
    update π_θ, v_θ according to Equation 2.27
  end for
end for

current policy $\boldsymbol{p}_\theta$, which can be used as policy improvement steps. Finally, we compute the new returns based on the states visited only by computing the new reward signals for each new subgoal. This procedure is done once for each episode and involves negligible additional computation. We call this method AlphaZeroHER. Algorithm 3 shows the pseudocode of the proposed procedure. We notice that the main loop of AlphaZero, where target values are generated from the policy targets given from MCTS and the Monte Carlo returns observed during the episodes, is extended with a set of additional experiences based on the secondary goal states. In our implementation, the goal states are sampled uniformly from the states visited during each episode.

3.3.2 Motivating Example

To highlight the AlphaZero criticality in sparse return environments, we consider a simple BitFlip domain from (Andrychowicz et al., 2017), described in detail in Section 3.5. In this domain, the goal is to modify a long series of n bits to reach the desired bit

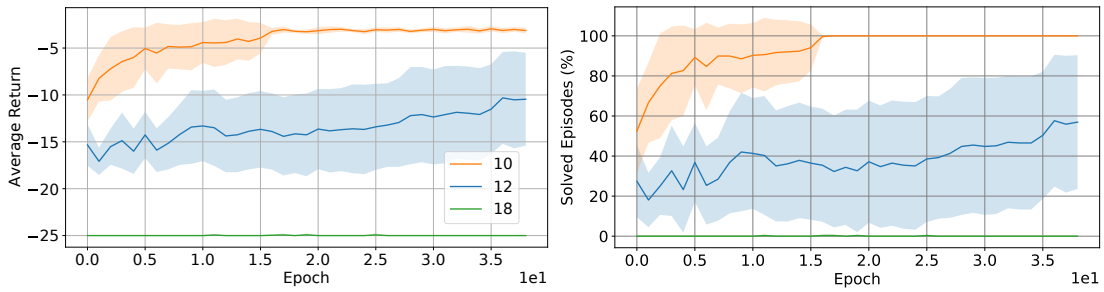


Figure 3.1: Performance of the AlphaZero agent in the BitFlip environment varying the number of bits using 20 search iterations. Average over 10 runs, 95% c.i.

configuration. While it might be easy to achieve the goal configuration by applying a random policy when considering a few bits only, it is practically impossible to reach the goal state if the bit length is increased.

Figure 3.1 shows the performance of plain AlphaZero in the bit-flip environment, where we have plotted the expected return and the percentage of solved episodes as a function of the training epochs for three different scenarios. While AlphaZero can consistently solve an "easy" scenario of 10 bits achieving almost perfect performance, it struggles with a modest increase in the number of bits, dramatically failing to solve the task with 18 bits.

3.4 Related Literature

Goal-Directed Reinforcement Learning has been extensively studied over the years. The optimization of multiple goals has been largely investigated in multi-goal policy optimization, curriculum learning, goal-directed planning, and multiple-task off-policy learning.

In the context of universal value function approximators, in (Schaul et al., 2015), the authors consider the problem of approximating multiple value functions in a single architecture. In this line of work, various works study the problem of compact representations of multiple tasks (Dhiman et al., 2018; Ghosh et al., 2018). In such context, the use of sub-goals (intermediate states between the current state and the goal state) has been studied to accelerate learning and generalize over the state-space (Nasiriany et al., 2019; Jurgenson et al., 2020). In (Parascandolo et al., 2020), the authors move the problem of goal-directed planning from the space of possible policies to the problem of finding suitable sub-goals. While this has some advantages in certain situations, it cannot be applied in large state spaces since the complexity of the algorithm scales linearly with the number of states. Approximate value iteration has also been used with great results

to solve the Rubik's cube with a goal-directed framework in (McAleer et al., 2019).

In the context of policy search, (Pinto and Gupta, 2016) and (Caruana, 1997) aim to learn policies to solve multiple tasks simultaneously. Meta-learning has also been extensively studied in recent years and can be seen as closely related to multi-task learning. In meta-learning, a meta-learner is trained to learn swiftly in ever-changing tasks to adapt to learning in new untested tasks quickly. While meta-learning has been investigated early in the literature (Schmidhuber, 1987, 2007), recent work has shown impressive results in the context of Deep RL (Finn et al., 2017a, 2018). Hierarchical Reinforcement Learning is also closely related to meta-learning (and our work). Here the goal is to (automatically) learn to perform multiple tasks by splitting the main problem into sub-tasks (Schmidhuber, 2002; Fruit et al., 2017). This is also closely related to the options framework (Sutton and Barto, 2018).

3.5 Experiments

In this section, we provide an experimental evaluation of AlphaZeroHER on simulated domains, including a novel application on a quantum compiling task, modeled as a deterministic goal-directed MDP. In the following, "search iteration" refers to a single application of the 4 MCTS phases. More details are given in Appendix B.1.1.

3.5.1 BitFlip

We consider a BitFlip environment where the individual bits of a long series of n bits are changed to reach a desired final configuration. More precisely, the state space is $\mathcal{S} = \{0,1\}^n$, as well as the goal space $\mathcal{G}$. The action space $\mathcal{A} = \{0, 1, \ldots, n-1\}$ specifies which bit of the current state changes from 0 to 1 or vice-versa. At the beginning of each episode, the starting bits and goal state are set randomly with uniform measures over the state space. We use a "sparse" reward of -1 for each transition unless the goal state is reached.

We tested such an environment as a motivating example for the application of HER in AlphaZero and a benchmark test since increasing the number of bits can vary the task's difficulty sensibly. In all the runs in BitFlip, we use 20 neurons for the shared layer, 8 neurons for the policy layer and 4 for the value layer. The network hyper-parameters were not optimized. Figure 3.2 reports the performance of AlphaZeroHER in the scenario of 70 bits for different numbers of subgoals sampled (0 subgoals refers to plain AlphaZero). While AlphaZero struggles with 12 bits and fails with 18 bits, AlphaZeroHER manages to achieve great results in the case of 70 bits. Moreover, the agent achieves better

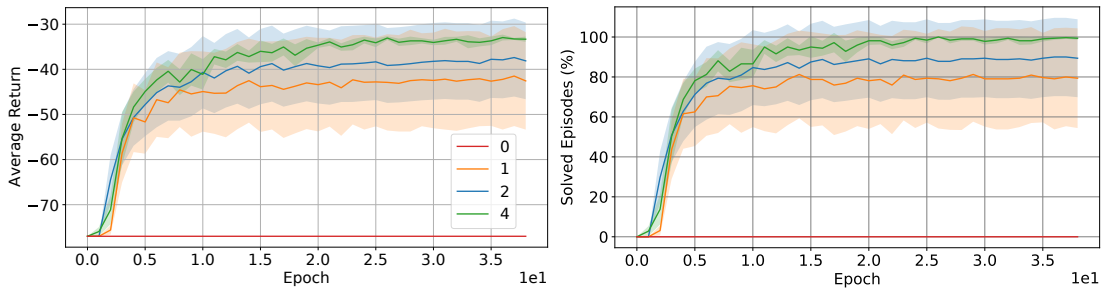


Figure 3.2: Comparison between AlphaZero (red line) and AlphaZeroHER (green, blue, and orange lines) in the BitFlip environment varying the number of sampled subgoals, using 20 search iterations and 70 bits. Average over 10 runs, 95% c.i.

performance by increasing the number of additional goals, getting perfect performance by using 4 subgoals only.

3.5.2 2D Navigation Task

In this section, we consider a 2D navigation task built on top of the Mujoco (Todorov et al., 2012) robotics simulator, called Point. The purpose of this experiment is to observe the performance of AlphaZero and AlphaZeroHER in a more challenging task, that requires a fair amount of exploration. More precisely, the agent's goal (orange ball) has to reach the goal state highlighted by the green rectangle as shown in Figure 3.4. The task is made more challenging by the presence of a wall in the center of the environment. At the beginning of each episode, the starting state and the goal state are sampled on the left and right sides of the wall, respectively. The agent observes its current position and its current velocities in both directions and the coordinates of the goal state, while the action space is represented as control over two actuators of the agent. Although the original action space is continuous over the domain $[-1, 1]^2$, we restricted the space to 9 discrete actions, representing the center (no action) and 8 points on the circle centered at action $(0, 0)$ with radius 1. The reward function is -1 at each step, making the optimal policy the shortest path that reaches the green rectangle while circumventing the wall.

We use a simple MLP with 20 neurons for the shared layer, 10 neurons for the policy layer, and 4 for the value layer. Figure 3.3 shows the results of the experiments in this domain. We use a large horizon of 200 steps, after which we interrupt if the goal state is not reached. While AlphaZero fails to solve the environment, only reaching the goal state in 10% of the episodes, by applying HER, we manage to solve most of the episodes, with only sampling 2 additional goals for each episode of experience.

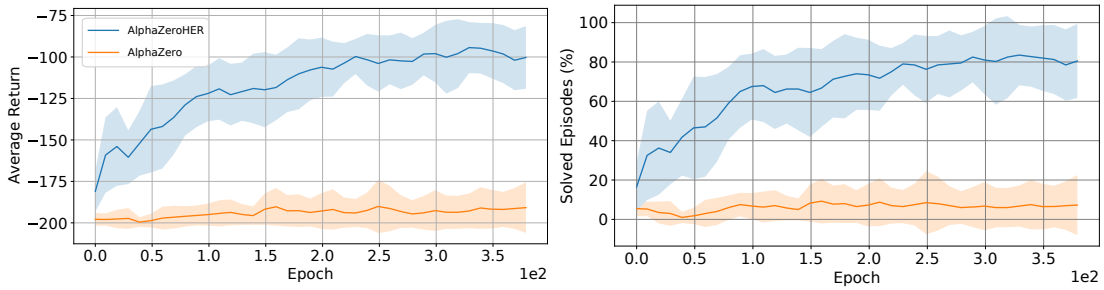


Figure 3.3: Comparison between AlphaZero and AlphaZeroHER in the 2D Navigation task using 70 search iterations. Average over 5 runs, 95% c.i.

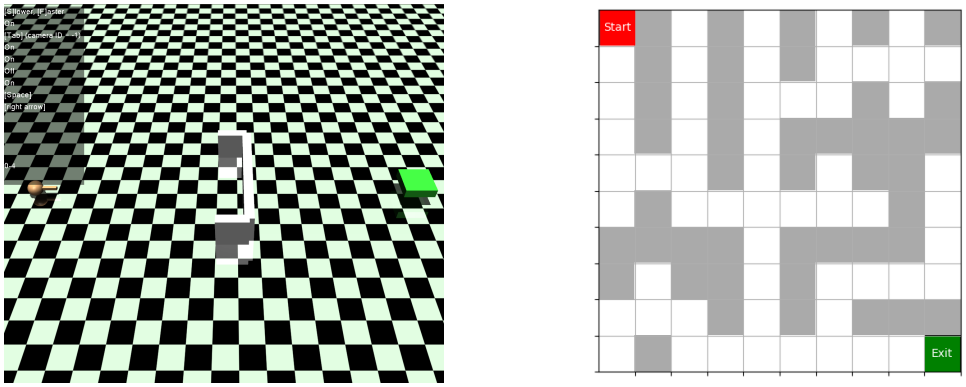


Figure 3.4: Visual representation of the Point (left) and Maze (right) environments.

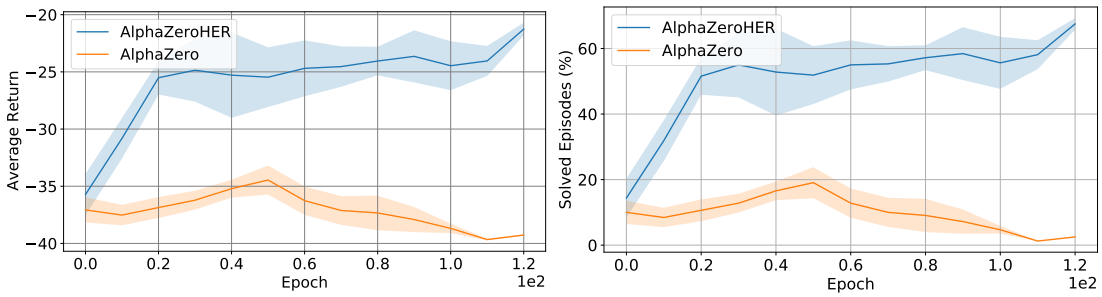


Figure 3.5: Comparison between AlphaZero and AlphaZeroHER in the 2D Maze task using 120 search iterations. Average over 5 runs, 95% c.i.

3.5.3 2D Maze

This section considers an environment of 2D procedurally generated mazes whose structure changes at each episode, as shown in Figure 3.4. This experiment aims to test the algorithm's performance in a challenging image task, where Convolutional Neural Networks represent the policy and value space. At each episode, the agent is spawned on a free cell (shown in red) and moves to reach the goal cell (shown in green). The action space consists of four directional movements, which move the agent to one of the adjacent cells. However, if the action points towards a wall, the agent does not move. The reward function employed in this environment is a constant reward of -1, prompting the agent to find the shortest path to the goal. The state space corresponds to a 2D image of the complete maze.

We employ the same network architecture in all the experiments, consisting of a 3-Layer CNN, with a kernel size of 3, a Layer Normalization after each convolutional layer, and strides of [1,1,2]. The policy and value network heads have two additional fully connected layers of 128 and 64 neurons each. We run experiments in 10x10 mazes, using a horizon of 60 steps, after which we interrupt if the goal state is not reached. Figure 3.5 shows the results of the experiments in this domain. We can see that AlphaZeroHER clearly outperforms plain AlphaZero in this environment, although it does not itself achieve a perfect score. We attribute this low general performance to the fact that the hyperparameters of the training process were not optimized, and in an image-based task, with CNNs as policy and value space, the architecture and training procedure are crucial. Nonetheless, AlphaZeroHER manages to substantially clearly improve over plain AlphaZero with only 2 additional goals.

3.5.4 Quantum Compiler Environment

Gate-model quantum computers achieve quantum computation by applying quantum transformations on quantum physical systems called qubits (Nielsen and Chuang, 2002). Due to hardware constraints and quantum disturbances (Linke et al., 2017; Maslov, 2017; Leibfried et al., 2007; Debnath et al., 2016), quantum computers require compilers to approximate any quantum transformations as ordered sequences of quantum gates that can be applied on the hardware. In this work, we employ AlphaZeroHER to address the problem of quantum compilation. We consider a Quantum Compiler (QC) environment consisting of a sequence $U_n = \prod_{j=1}^{n} A_j$ of quantum gates A_j that starts empty at the beginning of each episode, as fully described in Moro et al. (2021). Such sequence is built incrementally at each time-step by the agent, choosing from a finite set of quantum gates $\mathcal{B}$ corresponding to the action space. More specifically, we chose the base composed of six finite rotations, i.e $\mathcal{B} = \big(R_{\hat{x}}(\pm\frac{\pi}{128}),\ R_{\hat{y}}(\pm\frac{\pi}{128}),\ R_{\hat{z}}(\pm\frac{\pi}{128})\big)$.

The goal of the agent consists of approximating a single-qubit unitary transformation $\mathcal{U}$ that is chosen at each episode from Haar matrices. Pictorially, sampling Haar unitary matrices can be seen as choosing a number from a uniform distribution (Russell et al., 2017). The observation used as input at time-step t corresponds to the vector of the real and imaginary parts of the elements of the matrix O_n, where $\mathcal{U} = U_n \cdot O_n$. Such representation encodes all the useful information required to achieve the task, i.e., the unitary transformation to approximate $\mathcal{U}$ and the current sequence of gates. We exploited average gate fidelity (AGF) (Nielsen, 2002) as a metric to evaluate the distance between the target gate $\mathcal{U}$ and the current sequence of gates U_n. The task is solved whenever the agent reaches a distance equal to or greater than 0.99 AGF. The base of gates $\mathcal{B}$ allows defining a distance-based reward, which allows solving the problem with relative ease, although it leads to sub-optimal solutions as shown in Moro et al. (2021). However, in this work, we employ a sparse reward equal to -1 regardless of the action performed by the agent. For such reason, the task is very challenging since a high number of steps are required to approximate Haar unitary targets on average.

In all the experiments, we use the same network architecture, consisting of a simple MLP with one initial layer of 16 hidden neurons. The policy and value network heads have an additional layer of 8 and 4 hidden neurons respectively. Figure 3.6 shows the results of the experiments in the QC task. In this task, the planning horizon is substantially longer than in previous environments. We interrupted the episodes at 300 steps since a perfect policy achieves an average episode length of 180 steps and 95% of the solution can be achieved using less than 200 steps (Moro et al., 2021). AlphaZero shows a slow performance improvement, but after 100 training epochs it fails to learn the optimal

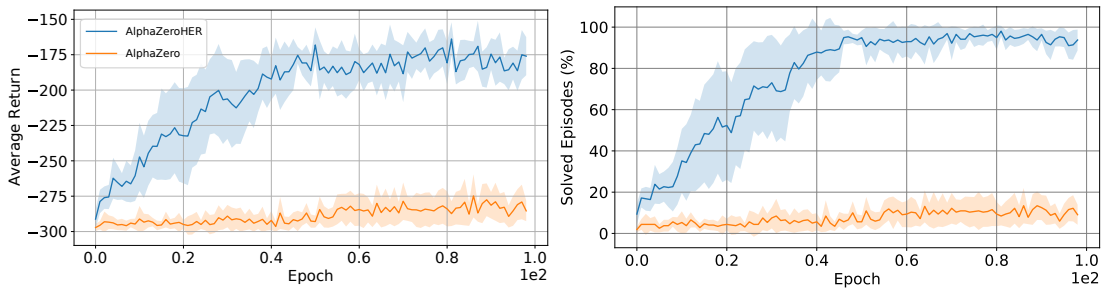


Figure 3.6: Comparison between AlphaZero and AlphaZeroHER in the quantum compiling environment with 20 search iterations. Average over 10 runs, 95% c.i.

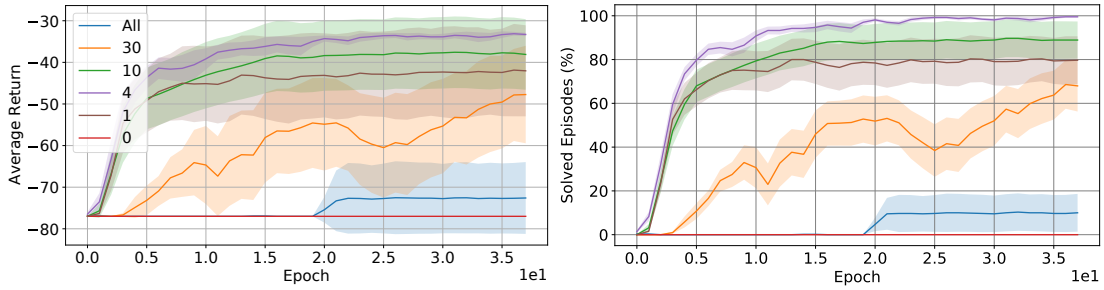


Figure 3.7: Varying the number of subgoals in the BitFlip environment. Average of 10 runs, 95% c.i..

policy. On the other hand, AlphaZeroHER consistently improves the performance and achieves almost perfect resolution of the problem.

3.5.5 Dependece on Number of Subgoals

We briefly study the effect of increasing the number of sampled subgoals in AlphaZeroHER. Figure B.1 and Figure B.2 show the results of varying the number of subgoals in the BitFlip and quantum compiling environments, respectively. We can see that in both enviroments the performance increases as we increase the number of subgoals k until we reach a (problem dependent) threshold, after which the performance starts falling until it reaches the lower levels when we use as subgoals, all the available ones (label All). This is in line with the results presented in the original HER paper (Andrychowicz et al., 2017).

3.5.6 Comparison with DQN + HER

In this section, we compared the proposed AlphaZeroHER with DQN+HER used in the original HER paper. The goal of this experiment is to answer whether using HER in an MCTS method like AlphaZero was needed, or using "more traditional" HER

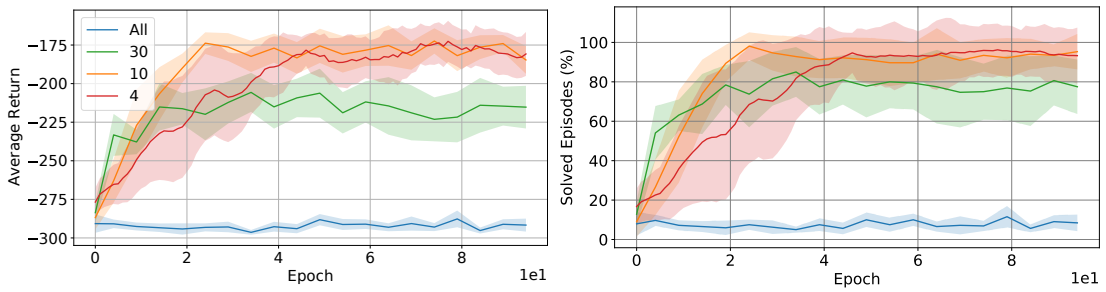


Figure 3.8: Varying the number of subgoals in the quantum compiling environment. Average of 10 runs, 95% c.i..

implementations, like DQN, was enough to solve the considered environments. We use the implementation of DQN and HER given from *stable-baselines* [1]. To make the comparison fair, we first employed the same network architecture used by our agents. However, except for the bitflip domains of less than 20 bits, the DQN agents could not solve any of the domains. For this reason, in the following results, we have employed a more complex network architecture for all domains (except Maze where we use the same CNN), namely an MLP with a single hidden layer of 128 neurons. We optimized the DQN hyperparameters using *hyperopt*[2]. The best hyperparameters used in all domains are listed in Table B.3. It is worth noting that the hyperparameters for AlphaZero and AlphaZeroHer presented in Section 5 were not tuned.

Figure 3.9 shows the results of DQN+HER in the Bitflip domain for different numbers of bits. We used four additional subgoals like in the AlphaZero case but with a more complex network architecture. We can reproduce the results presented in the HER paper, as the top-performing policies solve the problems 100% of the time. When comparing the average performance, though, DQN+HER still performs worse than AlphaZeroHER, only solving on average 80% of the episodes up to 60 bits, and even less when increasing to 70 bits.

We also achieved satisfactory results in the quantum compiling domain, solving around 80% of the episodes within the given horizon, yet still less than the 95% achieved from the AlphaZeroHER agent presented in Section 5. Figure 3.10 shows the performance in this domain. There, a more complex network structure was needed to achieve this performance too. DQN+HER starts to fail in the 2D navigation task. Figure 3.11 presents the results in this domain. Even after tuning the DQN parameters, we only can achieve around 5% of solved episodes on average, in contrast to the 80% achieved by AlphaZeroHER. Finally, in the Maze domain, even after tuning, DQN+HER could

[1] `https://github.com/hill-a/stable-baselines`

[2] `http://hyperopt.github.io/hyperopt/`

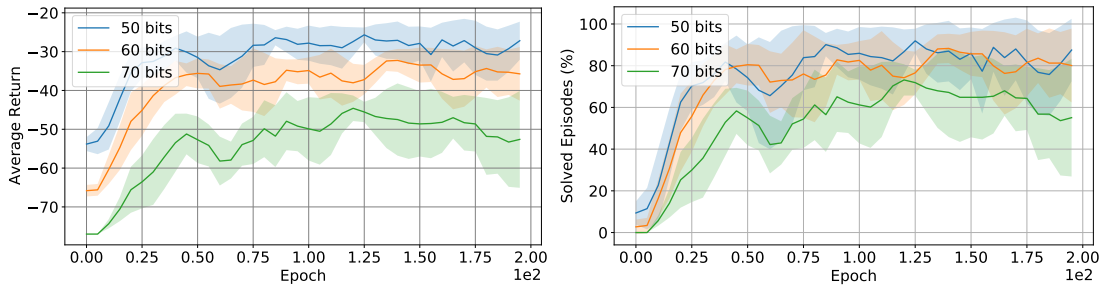


Figure 3.9: Performance of DQN+HER in Bitflip by varying the number of bits. Average of 5 runs, 95% c.i..

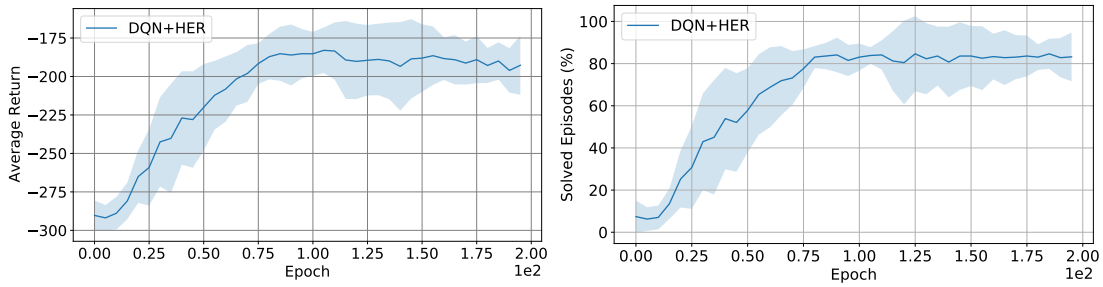


Figure 3.10: Performance of DQN+HER in QC using 4 additional subgoals. Average of 5 runs, 95% c.i..

not resolve a single episode of the task against the 60% solve rate of AlphaZeroHER. These experiments confirm again that sparse return tasks remain a problem not only for an MCTS algorithm like AlphaZero but also for a TD value-based algorithm like DQN, which despite the additional samples from HER, still fails to solve most of the tasks we investigated.

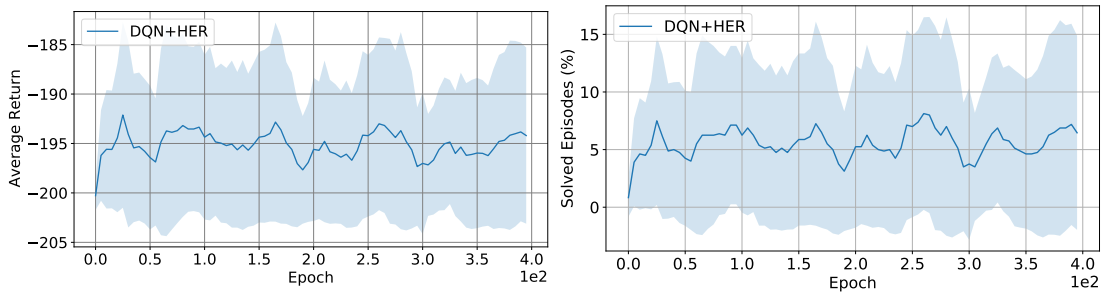


Figure 3.11: Performance of DQN+HER in the 2D navigation task. Average of 5 runs, 95% c.i..

3.6 Discussion

In this chapter, we identified how sophisticated Deep Reinforcement Learning algorithms like AlphaZero and DQN still can struggle in sparse return tasks, even in limited planning horizons. We illustrated this in the context of goal-directed tasks, where a single positive reward is received when reaching a goal state. For the specific case of goal-dependent tasks, we introduced a novel algorithm, consisting of the extension of AlphaZero with the Hindsight Experience Replay (HER) method to overcome the issues caused by sparse reward functions typical of goal-directed planning. We provide a straightforward procedure that does not involve high computational costs by sampling other goals from the visited states and addressing the intrinsic on-policy nature of AlphaZero. The proposed approach outperforms AlphaZero in several test domains, including a novel application to quantum compiling, with negligible additional computation compared to plain AlphaZero.

While this simple method was enough to improve performance in the considered goal-based benchmarks the exploration problem remains when considering classical RL tasks where the reward function is not goal-dependent. In goal-dependent tasks, we can generate new samples by changing the goal states and generalizing across them. In tasks where we cannot choose our own goals or we do not have access to the reward function we cannot apply HER to augment our samples. In these kinds of tasks, AlphaZero or any other MCTS or RL algorithm is completely dependent on randomly reaching the goal without any additional information.

In the following chapter, we will investigate novel approaches for improving exploration in RL. We will investigate epistemic uncertainty, as a tool to direct the exploration of our agents toward more promising regions of the state-action space. In sparse return settings, this would translate to incentivizing "new" regions of the state space, since they would come with a higher uncertainty even though no reward is collected, avoiding the undirected exploration we observed in this chapter. Specifically, we aim to devise methods that generalize uncertainty estimates, with the goal of improving exploration even in planning algorithms like AlphaZero, biasing the search in the tree towards promising regions of the state-action space, instead of towards regions favored by the policy as in Equation 2.25.

CHAPTER 4

Propagating Uncertainty in TD-Learning

4.1 Introduction

As discussed and illustrated in previous chapters, effectively balancing exploration and exploitation is a key challenge in Reinforcement Learning. When an agent takes decisions under uncertainty, it faces the dilemma between *exploiting* the information collected so far to execute what is believed to be the best action or to choose a possibly suboptimal action to *explore* new portions of the environment and gather new information, leading to more profitable behaviors in the future. Traditional exploration strategies, such as ϵ-greedy and Boltzmann exploration (Sutton and Barto, 2018), inject random noise into the action-selection process, i.e., the policy, to guarantee that each action is tried often enough. Although these methods allow RL algorithms to learn the optimal value function under mild assumptions (Singh et al., 2000), they are not *efficient*, since exploration is random and not driven by confidence on the value function estimate. Therefore, they might converge towards the optimal behavior after an exponential number of steps (Kakade et al., 2003).

The exploration-exploitation dilemma has been extensively analyzed in the RL community, focusing on the definition of proper indices for *provably-efficient* exploration

and devising algorithms with strong theoretical guarantees (Kearns and Singh, 2002; Brafman and Tennenholtz, 2002; Jaksch et al., 2010a; Osband et al., 2016a). Most of these algorithms are inherently model-based, i.e., they need to maintain and update estimates of the environment dynamics and the reward function during the learning process. For this reason, model-based methods are rather unsuited to problems with large state spaces and inapplicable to continuous environments. Apart from rare exceptions (Strehl et al., 2006), the RL community has only recently focused on devising efficient model-free exploration strategies. Some works have succeeded in obtaining provably-efficient algorithms (Pazis et al., 2016; Osband et al., 2016b; Jin et al., 2018); whereas others are more empirically-oriented (Osband et al., 2016a, 2018; Azizzadenesheli et al., 2018).

A fundamental step towards efficient exploration is the quantification of the *uncertainty* of the value function. The notion of uncertainty is formalized in Bayesian statistics by means of a *posterior* distribution. Bayesian Reinforcement Learning incorporates the Bayesian inference tools to provide a principled way to address the exploration-exploitation dilemma (Ghavamzadeh et al., 2015). However, these methods rarely exploit the specific way in which the uncertainty propagates through the Bellman equation. Recently, in (O'Donoghue et al., 2018) a partial answer has been provided, proposing an uncertainty Bellman inequality; although no posterior distribution is explicitly considered.

In this chapter, we propose a novel Bayesian framework to address the problem of exploration using posterior distributions over the value function. Specifically, we focus on how to *model* directly this epistemic uncertainty and *propagate* it when performing temporal-difference learning (Section 4.3). Moreover, we show how to use this uncertainty information to effectively explore the environment. Finally, we combine these elements to build our algorithm: *Wasserstein Q-Learning* (WQL, Section 4.4). Similarly to Bayesian Q-Learning (Dearden et al., 1998), we equip each state-action pair with an approximate posterior distribution (named *Q-posterior*), whose goal is to quantify the uncertainty of the value function. Whenever a transition occurs, we update our distribution, in a temporal difference (TD, Sutton and Barto, 2018) fashion, in order to incorporate all sources of uncertainty: i) the one due to the sample estimate of the reward function and environment dynamics; ii) the uncertainty injected using the estimate of the next-state value function. Rather than employing a standard Bayesian update, we resort to a variational approach to approximate the posterior distribution, based on Wasserstein barycenters (Agueh and Carlier, 2011). Recently, several works have embedded into RL algorithms notions coming from *Optimal Transport* (OT, Villani, 2008), especially the Wasserstein metric, to improve the learning abilities of policy search algorithms (Pacchiano et al., 2019) or in the field of robust RL (Abdullah et al., 2019). We will show that, when paired with proper choices of the approximate posterior distribution model, this method

has remarkable properties and allows accounting also for the correlation of samples. This will prove important, as we avoid costly updates while still being able to provide theoretical guarantees on the algorithm's performance. These variational updates also make it easy for us to extend the algorithms to the use of function approximators as discussed in Section 4.7. This will prove important, as it allows us to generalize our uncertainty estimate and use them in a variety of algorithms and settings, including continuous action spaces which are out of the scope of this chapter and will be addressed in Chapter 5 and online planning addressed in Chapter 6. In this way, we propose a novel framework that is theoretically grounded with provable guarantees in the tabular setting but can also be easily extended to DeepRL with limited additional computational costs.

Indeed, we prove in Section 4.5, that a slight modification of WQL, in tabular domains, is *PAC-MDP in the average loss setting* (Strehl and Littman, 2008). After examining the related literature (Section 4.6), we present an experimental evaluation on tabular environments to show the effectiveness of WQL, compared to the classic RL algorithms, some of which specifically designed for exploration (Section 4.7.1). Finally, we provide some preliminary results on the application of WQL to deep architectures (Section 4.7.2). The proofs of all results are reported in Appendix A.2. The implementation of the proposed algorithms can be found at `https://github.com/albertometelli/wql`.

4.2 Preliminaries

In this section, we recall some notation and basic notions we will use in the following. Given a measurable space $(\mathcal{X}, \mathscr{F})$, where $\mathcal{X}$ is a set and $\mathscr{F}$ is a σ-algebra over $\mathcal{X}$, we denote by $\mathscr{P}(\mathcal{X})$ the set of all probability measures over $(\mathcal{X}, \mathscr{F})$.

4.2.1 Wasserstein Barycenters

Let $(\mathcal{X}, d)$ be a complete separable metric (Polish) space and $x_0 \in \mathcal{X}$ be an arbitrary point. For each $p \in [1, +\infty)$ we define $\mathscr{P}_p(\mathcal{X})$ as the set of all probability measures μ over $(\mathcal{X}, \mathscr{F})$ such that $\mathbb{E}_{X\sim\mu}[d(X, x_0)^p] < +\infty$. Let $\mu, \nu \in \mathscr{P}_p(\mathcal{X})$, the L^p-Wasserstein distance between μ and ν is defined as (Villani, 2008):

$$W_p(\mu, \nu) = \left(\inf_{\rho \in \Gamma(\mu,\nu)} \mathop{\mathbb{E}}_{X,Y\sim\rho} [d(X, Y)^p] \right)^{1/p}, \tag{4.1}$$

where $\Gamma(\mu, \nu)$ is the set of all probability measures on $\mathcal{X} \times \mathcal{X}$ (couplings) with marginals μ and ν. With little abuse of notation, we will indicate with $W_p(X, Y) = W_p(\mu, \nu)$, whenever clear from the context. The Wasserstein distance comes from the optimal transport community. Intuitively, it represents the "cost" to move the probability mass

to turn one distribution into the other. Given a set of probability measures $\{\nu_i\}_{i=1}^n$, belonging to the class $\mathscr{N}$, and a set of weights $\{\xi_i\}_{i=1}^n$, $\sum_{i=1}^n \xi_i = 1$ and $\xi_i \geq 0$, the L^2-Wasserstein barycenter is defined as (Agueh and Carlier, 2011):

$$\overline{\nu} = \arg\inf_{\nu \in \mathscr{N}} \left\{ \sum_{i=1}^{n} \xi_i W_2(\nu, \nu_i)^2 \right\}. \tag{4.2}$$

4.3 How to Model and Propagate Uncertainty?

In this section, we introduce a unifying Bayesian framework for exploration in RL that employs (approximate) posterior distributions to *model* uncertainty of value functions (Section 4.3.1) and Wasserstein barycenters to *propagate* uncertainty when performing TD updates (Section 4.3.2). Furthermore, we discuss how to leverage on the Q-posteriors to estimate the action that attains the maximum return in each state (Section 4.3.3) and to effectively explore the environment (Section 4.3.4).

4.3.1 Modeling Uncertainty via Q-Posteriors

Taking inspiration from Bayesian approaches to RL (Dearden et al., 1998; Ghavamzadeh et al., 2015), for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ we maintain a probability distribution $\mathcal{Q}(s,a)$, which we call *Q-posterior*, representing a (possibly approximate) posterior distribution of the Q-function estimate. This distribution will depend on the underlying MDP, in particular, the environment dynamics $\mathcal{P}$ and reward model $\mathcal{R}$, and on the updates of the Q-function estimates performed. Given a learning algorithm, at each iteration t, the estimation of the Q-function will follow a certain distribution that is the result of two contributions: i) the underlying MDP, in particular the environment dynamics $\mathcal{P}$ and reward model $\mathcal{R}$; ii) the learning algorithm itself, especially the policies played and the update rules executed up to time t. Furthermore, when considering TD algorithms, these distribution are likely to be dependent across different state-action pairs, since samples are shared when performing TD update rules (2.20). As in a model-free scenario we cannot represent such distribution exactly, we employ a class of approximating probability distributions $\mathscr{Q} \subseteq \mathscr{P}(\mathbb{R})$. Similarly to usual value functions, we introduce the *V-posterior* $\mathcal{V}(s)$ which represents the (possibly approximate) posterior distribution of V-function, that combines the uncertainties modeled by the Q-posteriors $\mathcal{Q}(s,a)$. Furthermore, being the V-function defined, in the usual framework, as the expectation of the Q-function over the action space, i.e., $V^\pi(s) = \mathbb{E}_{A\sim\pi}[Q^\pi(s,a)]$, it is natural to define, in our setting, the V-posterior $\mathcal{V}(s)$ as the Wasserstein barycenter of the Q-posteriors

$\mathcal{Q}(s,a)$.[1]

Definition 4.3.2 (V-posterior)**.** *Given a policy $\overline{\pi}$ and a state $s \in \mathcal{S}$, we define the* V-posterior $\mathcal{V}(s)$ *induced by the Q-posteriors $\mathcal{Q}(s,a)$ with $a \in \mathcal{A}$ as the Wassertein barycenter of the $\mathcal{Q}(s,a)$:*

$$\mathcal{V}(s) \in \arg\inf_{\mathcal{V} \in \mathscr{Q}} \left\{ \mathop{\mathbb{E}}_{A \sim \overline{\pi}(\cdot|s)} \left[W_2 \left(\mathcal{V}, \mathcal{Q}(s,A) \right)^2 \right] \right\}. \tag{4.4}$$

When the policy $\overline{\pi}$ is known, the expectation over the action space can be computed as we are assuming that $\mathcal{A}$ is finite. In a prediction problem, policy $\overline{\pi}$ is a fixed policy, whereas, in a control problem, $\overline{\pi}$ is a policy aimed at properly selecting the best action in state s accounting for the uncertainty modeled by the Q-posterior (see Section 4.3.3). Moreover, when $\mathcal{Q}(s,a)$ are deterministic distributions, $\mathcal{V}(s)$ is a deterministic distribution too centered in the mean of the $\mathcal{Q}(s,a)$. In this way, we obtain the usual V-function definition (see Proposition A.1.3).

It is important to stress that our approach is rather different from *Distributional Reinforcement Learning* (Bellemare et al., 2017a; Dabney et al., 2018a; Rowland et al., 2018). Indeed, we employ a distribution to represent the *uncertainty* of the Q-function estimate and not the *intrinsic randomness* of the return. The two distributions are clearly related and both depend on the stochasticity of the reward and of the transition model. However, in our approach the stochasticity refers to the uncertainty on the Q-function estimate which reduces as the number of updates increases, being a sample mean.[2]

4.3.2 Propagating Uncertainty via Wasserstein Barycenters

In this section, we discuss the problem of uncertainty propagation, i.e., how to deal with the update of the Q-posteriors when experiencing a transition $(S_t, A_t, S_{t+1}, R_{t+1})$. Whenever a TD update is performed, there are two sources of uncertainty involved. First, we implicitly estimate the environment dynamics $\mathcal{P}(\cdot|S_t, A_t)$ and the reward model $\mathcal{R}(\cdot|S_t, A_t)$ using a set of sampled transitions $(S_t, A_t, S_{t+1}, R_{t+1})$. Second, when using the V-function estimates of the next states $V_t(S_{t+1})$ we bring into $Q_{t+1}(S_t, A_t)$ part of the uncertainty of $V_t(S_{t+1})$ and they become correlated. This phenomenon poses new challenges in the correct use of Bayesian approaches to model the uncertainty of the Q-function estimates. For this reason, the standard Bayesian posterior update, used for instance in Bayesian Q-learning (Dearden et al., 1998), becomes rather inappropriate

[1] The Wasserstein barycenter can be regarded as a way of averaging distributions (Agueh and Carlier, 2011).

[2] A notable difference w.r.t. the distributional RL is that the variance of our posterior distribution $\mathop{\mathbb{V}\text{ar}}_{Q \sim \mathcal{Q}(s,a)}[Q]$ vanishes as the number of updates grows to infinity.

as it assumes that the samples are independent, which is clearly not true. We argue that, rather than using a Bayesian update, when we have a Q-posterior $\mathcal{Q}_t(S_t, A_t)$ and a V-posterior $\mathcal{V}_t(S_{t+1})$ we can combine them using a notion of barycenter, which does not require the independence assumption. We formalize this idea in the following update rule.

Definition 4.3.4 (Wasserstein Temporal Difference)**.** *Let $\mathcal{Q}_t$ be the current Q-posterior, given a transition $(S_t, A_t, S_{t+1}, R_{t+1})$, we define the* TD-target-posterior *as $\mathcal{T}_t = R_{t+1} + \gamma\mathcal{V}_t(S_{t+1})$. Let $\alpha_t \geq 0$ be the learning rate, we define the* Wasserstein Temporal Difference *(WTD) update rule as:*

$$\mathcal{Q}_{t+1}(S_t, A_t) \in \underset{\mathcal{Q}\in\mathscr{Q}}{\arg\inf} \left\{ (1-\alpha_t) W_2\left(\mathcal{Q}, \mathcal{Q}_t(S_t, A_t)\right)^2 + \alpha_t W_2\left(\mathcal{Q}, \mathcal{T}_t\right)^2 \right\}. \tag{4.6}$$

Therefore, the new Q-posterior $\mathcal{Q}_{t+1}(S_t, A_t)$ is the Wasserstein barycenter between the current Q-posterior $\mathcal{Q}_t(S_t, A_t)$ and the TD-target posterior $\mathcal{T}_t = R_{t+1} + \gamma\mathcal{V}_t(S_{t+1})$, which in turn embeds information of the current transition (i.e., the reward R_{t+1} and the next state S_{t+1}) and the next-state V-posterior $\mathcal{V}_t(S_{t+1})$. It is worth noting that the two terms appearing in Equation (4.13) account for all sources of uncertainty. Indeed, the first term $W_2\left(\mathcal{Q}, \mathcal{Q}_t(S_t, A_t)\right)$ avoids moving too far from the current estimation $\mathcal{Q}_t(S_t, A_t)$, as we are performing the update experimenting a single transition, whereas $W_2\left(\mathcal{Q}, \mathcal{T}_t\right)$ allows bringing in the new Q-posterior the V-posterior of the next-state $\mathcal{V}_t(S_{t+1})$ (including its uncertainty). It is worth noting, that other common distributional divergences are rather inappropriate in this context. For instance, the class of α-divergences (which include the widely used KL-divergence) are non-well defined for deterministic distributions. However, we want to be able to average deterministic distributions too. Indeed, as the learning process goes, we expect that the variance of our approximate posterior vanishes as we are gathering more information and reducing the uncertainty. We stress the analogy with the standard TD update in the following result.

Proposition 4.3.1. *If $\mathscr{Q}$ is the set of deterministic distributions over $\mathbb{R}$, then the WTD update rule (Equation* (4.13)*) has a unique solution that corresponds to the TD update rule.*

The choice of the prior for $\mathcal{Q}_0$ plays an important role, along with the learning rate schedule α_t. We will show in Section 4.5 that specific choices of $\mathcal{Q}_0$ and α_t, for a particular class of distributions $\mathscr{Q}$, allow achieving PAC-MDP property in the average loss setting.

4.3.3 Estimating the Maximum Expected Value

The TD-target-posterior $\mathcal{T}_t = R_{t+1} + \gamma \mathcal{V}_t(S_{t+1})$ is defined in terms of the next state V-posterior $\mathcal{V}_t(S_{t+1})$. In a control problem, we aim at learning the optimal Q-function Q^* and, thus, we are interested in propagating back to $\mathcal{Q}_{t+1}(S_t, A_t)$ a V-posterior $\mathcal{V}_t(S_{t+1})$ related to the optimal action to be taken in the next state.[3] This can be performed by a suitable choice of the policy $\overline{\pi}$, as in Definition 4.3.1.

A straightforward approach consists in propagating the Q-posterior $\mathcal{Q}(S_{t+1}, a)$ of the action with the highest estimated mean:

$$\overline{\pi}^M(\cdot|s) \in \mathscr{P}\left(\arg\max_{a\in\mathcal{A}}\{\mathop{\mathbb{E}}_{Q\sim\mathcal{Q}(s,a)}[Q]\}\right). \tag{4.8}$$

We refer to this approach as *Mean Estimator* (ME) for the maximum. However, when posterior distributions are available, we can use them to define a wiser way to estimate the V-posterior of the next state.[4] A first method based on Optimism in the Face of Uncertainty (OFU, Auer et al., 2002) consists in selecting the action that maximizes a statistical upper bound $u^\delta(s, a)$ of the Q-posterior, where $\delta \in [0, 1]$ is a confidence parameter andthe upper bound is defined as:

$$u_t^\delta(s, a) = \min\{F_{s,a}^{-1}(1-\delta), Q_{\max}\}, \tag{4.9}$$

where $F_{s,a}^{-1}$ is the quantile function of the Q-posterior $\mathcal{Q}(s, a)$. The policy in this case is given by:

$$\overline{\pi}^O(\cdot|s) \in \mathscr{P}\left(\arg\max_{a\in\mathcal{A}}\{u^\delta(s, a)\}\right). \tag{4.10}$$

We will refer to this method as *Optimistic Estimator* (OE). However, if we want to make full usage of the Q-posteriors, we can resort to the *Posterior Estimator* (PE) of the maximum, based on Posterior Sampling (PS, Thompson, 1933). In this case, each action contributes to the update rule weighted by the probability of being the optimal action:

$$\overline{\pi}^P(a|s) = \Pr_{Q_{s,a}\sim\mathcal{Q}(s,a)}\left(a \in \arg\max_{a'\in\mathcal{A}}\{Q_{s,a'}\}\right) \tag{4.11}$$

Table 4.1 reports the definitions of the policy $\overline{\pi}$ for the three presented estimators.

4.3.4 Exploring using the Q-posteriors

Up to now, we have shown how to use distributions to model the uncertainty in the Q-function estimate and how to use the notion of Wasserstein barycenter to propagate

[3] We stress that we are uninterested in modeling the distribution $\max_{a\in\mathcal{A}}\{\mathcal{Q}(s, a)\}$, but rather in exploiting the uncertainty modeled by $\mathcal{Q}(s, a)$ to properly perform the computation of the optimal action.

[4] This problem was treated in RL, without distributions, proposing several estimators, such as the double estimator (Van Hasselt, 2010) and the weighted estimator (D'Eramo et al., 2016, 2017).

Table 4.1: Definition of the three policies $\overline{\pi}$ to be used in Definition 4.4 for computing the Mean (ME), Optimistic (OE) and Posterior (PE) estimators of the maximum.

Maximum Estimator	*Policy*
Mean (ME)	$\overline{\pi}^M(\cdot\|s) \in \mathscr{P}\left(\arg\max_{a\in\mathcal{A}} \mathbb{E}_{Q\sim\mathcal{Q}(s,a)}[Q]\right)$
Optimistic (OE)	$\overline{\pi}^O(\cdot\|s) \in \mathscr{P}\left(\arg\max_{a\in\mathcal{A}} u^\delta(s,a)\right)$
Posterior (PE)	$\overline{\pi}^P(a\|s) = \Pr_{Q_{s,a}\sim\mathcal{Q}(s,a)}\left(a \in \arg\max_{a'\in\mathcal{A}} Q_{s,a'}\right)$

Table 4.2: Probability density function (pdf), Wasserstein Temporal Difference (WTD) update rule and computation of the V-posterior for Gaussian and Particle posterior distributions.

$\mathscr{Q}$	pdf	WTD and V-posterior
Gaussian	$\frac{\exp\left\{-\frac{1}{2}\left(\frac{x-m(s,a)}{\sigma(s,a)}\right)^2\right\}}{\sqrt{2\pi\sigma^2(s,a)}}$	$m_{t+1}(S_t,A_t) = \alpha_t m_t(S_t,A_t) + (1-\alpha_t)\left(R_{t+1} + \gamma m_t(S_{t+1})\right)$ $\sigma_{t+1}(S_t,A_t) = \alpha_t\sigma_t(S_t,A_t) + (1-\alpha_t)\gamma\sigma_t(S_{t+1})$ $m(s) = \mathbb{E}_{A\sim\overline{\pi}(\cdot\|s)}\left[m(s,A)\right]$ $\sigma(s) = \mathbb{E}_{A\sim\overline{\pi}(\cdot\|s)}\left[\sigma(s,A)\right]$
Particle	$\sum_{j=1}^M w_j\delta(x - x_j(s,a))$ $x_1(s,a) \le ... \le x_M(s,a)$ $\sum_{i=j}^M w_j = 1$ and $w_j \ge 0$	$x_{j,t+1}(S_t,A_t) = \alpha_t x_{j,t}(S_t,A_t) + (1-\alpha_t)\left(R_{t+1} + \gamma x_{j,t}(S_{t+1})\right)$ $x_j(s) = \mathbb{E}_{A\sim\overline{\pi}(\cdot\|s)}\left[x_j(s,A)\right],\ j = 1,2,...,M$

uncertainty across state-action pairs when performing TD learning. In this section, we show that we can use these distributions to devise also exploration policies. Clearly, the choice of a suitable policy depends on the goal of the experiment. We might be interested in learning the Q-function as fast as possible with no need of showing a good on-line performance or we might be interested in provably efficient algorithms.

In the previous section, we have introduced two approaches that exploit the Q-posterior to properly define the V-posterior of the next state, using specific policies $\overline{\pi}$. These policies can also be used to implement effective exploration strategies aware of the uncertainty. Using the optimistic policy $\overline{\pi}^O$ in each state, we play (deterministically) the action that maximizes the statistical upper bound on the estimated Q-function $u^\delta(s,a)$, we call this strategy *Optimistic Exploration* (OX). Instead, we can directly use the posterior policy $\overline{\pi}^P$ to sample the action from the Q-posterior $\mathcal{Q}(s,a)$. Thus, in *Posterior Exploration* (PX), each action is played with the probability of being optimal.

4.4 Wasserstein Q-Learning

Algorithm 4 Wasserstein Q-Learning.

Input: a prior distribution $\mathcal{Q}_0$, a step size schedule $(\alpha_t)_{t\geq 0}$, an exploration policy schedule $(\pi_t)_{t\geq 0}$
Initialize $\mathcal{Q}(s,a)$ with the prior $\mathcal{Q}_0$
for $t = 1, 2, ...$ **do**
    Take action $A_t \sim \pi_t(\cdot|S_t)$
    Observe S_{t+1} and R_{t+1}
    Compute $\mathcal{V}_t(S_{t+1})$ using Equation (4.4)
    Update $\mathcal{Q}_{t+1}(S_t, A_t)$ using Equation (4.13)
end for

The ideas presented so far can be combined in an algorithm, *Wasserstein Q-Learning* (WQL), whose pseudocode is reported in Algorithm 4.

We developed our approach for a generic class of distributions $\mathscr{Q}$, however, in practice, we focus on two specific classes: *Gaussian* posteriors (G-WQL) and *Particle* posteriors (P-WQL), i.e., a mixture of $M > 1$ Dirac deltas. For both classes the Wasserstein Barycenter is unique and can be computed in closed form (see Appendix A.1.3).[5] In Table 4.2, we summarize the main relevant features of these distributions classes. It is worth noting, that when using policy π^{OFU} paired with OO maximum estimator and when using policy π^{PS} paired with PE, we are using the same policy to explore the environment and to select the action whose approximate posterior will be propagated. In a sense, this can be thought as a distributional version of Expected SARSA (Van Seijen et al., 2009).

WQL simply needs to store the parameters of the Q-posterior for every state-action pair ($m(s,a)$ and $\sigma(s,a)$ for G-WQL and $x_j(s,a)$ for P-WQL). Therefore, unlike the majority of provably-efficient algorithms, it can be extended straightforwardly to continuous state spaces as long as we adopt a function approximator for the parameters of the posterior. For instance, we could approximate $m(s,a)$ and $\sigma(s,a)$ or the particles $x_j(s,a)$ using a neural network with multiple heads. For this reason, our method easily applies to deep architecture by adopting a network that directly outputs the posterior parameters, instead of the value function (see Section 4.7.2).

[5] It is worth noting that, even for the Gaussian case, using the standard Bayesian posterior update is inappropriate, as the independence of the Q-function estimates across state-action pairs cannot be assumed.

Algorithm 5 Modified Gaussian Wasserstein Q-Learning

1: **Input:** m_0, σ_0, σ_b
2: **for** $t = 1, 2, ...$ **do**
3: Take action $A_t \in \arg\max_{a \in \mathcal{A}} u_t^{\delta}(S_t, A_t)$
4: Observe S_{t+1} and R_{t+1}
5: Update the posterior distribution

$$
\begin{aligned}
m_{t+1}(S_t, A_t) &= (1 - \alpha_t)\widetilde{m}_t(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma m_t(S_{t+1})\right) \\
\widetilde{\sigma}_{t+1}(S_t, A_t) &= (1 - \alpha_t)\widetilde{\sigma}_t(S_t, A_t) + \alpha_t \gamma \sigma_t(S_{t+1}) \\
\sigma_{t+1}(S_t, A_t) &= \widetilde{\sigma}_{t+1}(S_t, A_t) + \beta_t \sigma_b \\
n_{t+1}(S_t, A_t) &= n_t(S_t, A_t) + 1
\end{aligned}
$$

6: **end for**

4.5 Theoretical Analysis

In this section, we show that WQL, with some modifications, enjoys desirable theoretical properties in the tabular setting. We start providing a modification of the WTD update rule that will be used for the analysis; then we prove that with such modification our algorithm, under certain assumptions, is PAC-MDP in the average loss setting (Strehl and Littman, 2008).

Definition 4.5.2 (Modified Wasserstein Temporal Difference)**.** *Let $\mathcal{Q}_t$ be the current Q-posterior and $\mathcal{Q}_b$ be a zero-mean distribution, given a transition $(S_t, A_t, S_{t+1}, R_{t+1})$, we define the* TD-target-posterior *as $\mathcal{T}_t = R_{t+1} + \gamma \mathcal{V}_t(S_{t+1})$. Let $\alpha_t, \beta_t \geq 0$ be the learning rates, we define the* Modified Wasserstein Temporal Difference *(MWTD) update rule as:*

$$
\begin{aligned}
\widetilde{\mathcal{Q}}_{t+1}(S_t, A_t) &\in \inf_{\mathcal{Q} \in \mathscr{Q}} \left\{ (1 - \alpha_t) W_2 \left(\mathcal{Q}, \widetilde{\mathcal{Q}}_t(S_t, A_t)\right)^2 + \alpha_t W_2 \left(\mathcal{Q}, \mathcal{T}_t\right)^2 \right\}, \\
\mathcal{Q}_{t+1}(S_t, A_t) &\in \inf_{\mathcal{Q} \in \mathscr{Q}} \left\{ W_2 \left(\mathcal{Q}, \widetilde{\mathcal{Q}}_{t+1}(S_t, A_t) + \beta_t \mathcal{Q}_b\right)^2 \right\},
\end{aligned}
$$

We will denote the algorithm employing this update rule as *Modified Wasserstein Q-Learning* (MWQL). We provide the pseudocode for this modified version in Algorithm 5. The reason why we need to change the WTD lies in the fact that the uncertainty on the Q-function value (the Q-posterior) is, as already mentioned, the contribution of two terms: i) the uncertainty on the reward and transition model; ii) the uncertainty on the next-state Q-function. These terms need to be averaged into the Q-posterior at different speeds. If $n_t(s, a)$ is the number of times (s, a) is visited up to time t, (i) has to reduce proportionally to $1/\sqrt{n_t(s, a)}$ being a sample mean, while (ii) is averaged with coefficients proportional to $1/n_t(s, a)$. Therefore, we should keep the two sources of

uncertainty separated. To this end, we use an additional distribution $\mathcal{Q}_b$ to prevent the uncertainty from reducing too fast.

The notion of *PAC-MDP in the average loss setting* (Strehl and Littman, 2008) is a relaxation of the classical PAC-MDP notion introduced in (Kakade et al., 2003), in which we consider the actual reward received by the algorithm while learning, instead of the expected values over future policies. We recall the definitions given in (Strehl and Littman, 2008).

Definition 4.5.4 (Definition 4 of (Strehl and Littman, 2008))**.** *Suppose a learning algorithm $\mathfrak{A}$ is run for T steps. Consider partial sequence $S_0, R_1, ..., S_{T-1}, R_T, S_T$ visited by $\mathfrak{A}$. The* instantaneous loss *of the agent at time t is $il_{\mathfrak{A}}(t) = V^*(S_t) - \sum_{i=t}^{T} \gamma^{i-t} R_{i+1}$. The quantity $\mathcal{L}_{\mathfrak{A}} = \frac{1}{T} \sum_{t=1}^{T} il_{\mathfrak{A}}(t)$ is called the* average loss.

Then, a learning algorithm $\mathfrak{A}$ is PAC-MDP in the average loss setting if for any $\epsilon \geq 0$ and $\delta \in [0, 1]$, we can choose a value T, polynomial in the relevant quantities $(1/\epsilon, 1/\delta, |\mathcal{S}|, |\mathcal{A}|, 1/(1-\gamma))$, such that the average loss $\mathcal{L}_{\mathfrak{A}}$ of the agent (following the learning algorithm $\mathfrak{A}$) on a trial of T steps is guaranteed to be less than ϵ with probability at least $1 - \delta$.

In the following, we will restrict our attention to MWQL with Gaussian posterior, optimistic estimator (OE) and optimistic exploration policy (OX). We leave the analysis of the posterior sampling exploration (PX) as future work. To prove the main result we need an intermediate result.

Theorem 4.5.1. *Let $S_0, ..., S_{T-1}, S_T$ be the sequence of states and actions visited by MWQL with Gaussian posterior, OE and OX. Then, there exists a prior $\mathcal{Q}_0$ and a zero-mean distribution $\mathcal{Q}_b$ and a learning rate schedule for $(\alpha_t, \beta_t)_{t \geq 0}$ (whose values are reported in Appendix A.2.1), such that for any $\delta \in [0, 1]$, with probability at least $1 - \delta$ it holds that:*[6]

$$\sum_{t=1}^{T} [V^*(S_t) - V_{\mathfrak{A}}(S_t)] \leq \mathcal{O}\left(\frac{Q_{\max}}{(1-\gamma)^{\frac{3}{2}}} \sqrt{|\mathcal{S}||\mathcal{A}| T \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} \right), \tag{4.14}$$

where $V_{\mathfrak{A}}$ is the value function induced by the (non-stationary) policy played by algorithm $\mathfrak{A}$.

From this result, we can exploit an analysis similar to (Strehl and Littman, 2008) to prove that MWQL with Gaussian posterior, OE and OX is PAC-MDP in the average loss setting.

[6] This performance index resembles the *regret* (Jaksch et al., 2010a). However, it is a weaker notion, being defined in terms of the trajectory generated by algorithm $\mathfrak{A}$, instead of the trajectories of an optimal policy.

Theorem 4.5.2. *Under the hypothesis of Theorem 4.5.1, MWQL with Gaussian posterior, OE and OX is PAC-MDP in the average loss setting, i.e., for any* $\epsilon \geq 0$ *and* $\delta \in [0, 1]$*, after*

$$T = \mathcal{O}\left(\frac{Q_{\max}^2 |\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3} \log \frac{Q_{\max}^2 |\mathcal{S}|^2 |\mathcal{A}|^2}{\delta \epsilon^2 (1-\gamma)^3}\right)$$

steps we have that the average loss $\mathcal{L}_{\mathfrak{A}} \leq \epsilon$ *with probability at least* $1 - \delta$*.*

The per-step computational complexity of MWQL is $\mathcal{O}(\log |\mathcal{A}|)$ as we can maintain the upper bounds of the Q-function as a max-priority queue (Strehl et al., 2009) and the space complexity is $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$.

Despite the theoretical guarantees, MWQL turns out to be often impractical for two main reasons. First, MWQL cannot be extended to continuous MDPs, as α_t and β_t are defined in terms of number of visits $n(s, a)$ (Equation (A.14)), which can only be computed for finite MDPs. Second, as many provably efficient RL algorithms, MWQL is extremely conservative, leading to very slow convergence. This is why most provably efficient RL algorithms, when used in practice, are run with non-theoretical values of hyperparameters. In this sense, WQL can be seen as a "practical" version of MWQL in which α_t is treated as a normal hyper-parameter and $\beta_t = 0$. Although the PAC-MDP property in the average loss setting is a weaker notion of provable efficiency compared to the most used PAC-MDP or regret, we stress the fact that our main goal is not to provide an algorithm with tight theoretical guarantees rather to propose a theoretically grounded method that can be employed, with suitable modifications, beyond tabular domains.

4.6 Related Literature

A variety of approaches have been proposed in the RL literature to tackle the exploration-exploitation trade-off (Szepesvári, 2010). We recall here only those that do not assume the availability of a simulator of the environment (Koenig and Simmons, 1992). A first dimension of classification is the RL setting they consider: *finite-horizon*, *discounted* or *undiscounted*. Finite-horizon MDPs are a convenient framework to devise provably-efficient exploration algorithms with theoretical guarantees on the *regret* (Osband et al., 2013; Dann and Brunskill, 2015; Azar et al., 2017). Recently, in (Jin et al., 2018) it was shown that Q-learning, in the finite-horizon setting, can be made efficient by resorting to suitable exploration bonuses. Similar results have been proposed in the infinite-horizon undiscounted case. The main challenge of this class of problems is the connection structure of the MDP (Bartlett and Tewari, 2009). Early approaches (Kearns and Singh, 2002; Auer and Ortner, 2007; Tewari and Bartlett, 2008; Jaksch et al., 2010a) impose restrictive

requirements on either mixing/hitting times or diameter, which have been progressively relaxed (Fruit et al., 2018). A significant part of the early provably-efficient algorithms considers the discounted setting (Kearns and Singh, 2002; Brafman and Tennenholtz, 2002; Strehl et al., 2006; Szita and Szepesvári, 2010; Lattimore and Hutter, 2014). However, their theoretical guarantees are based on the notion of PAC-MDP (Kakade et al., 2003) rather than on regret.

Another relevant dimension is the kind of policy used for exploration. Taking inspiration from the Multi Armed Bandit (MAB, Berry and Fristedt, 1985) framework, two main approaches have been proposed: Optimism in the Face of Uncertainty (Auer et al., 2002) and Thompson Sampling (Thompson, 1933). Most exploration algorithms employ the optimistic technique, selecting actions from the optimal policy of an optimistic approximation of the MDP (Jaksch et al., 2010a) or of the value function directly (Strehl et al., 2006; Jin et al., 2018). Some methods, instead, use a posterior sampling approach in which either the entire MDP or a value function is sampled from a (possibly approximate) posterior distribution.

Inspired by these methods, numerous practical variants have been devised. Exploration bonuses, based on pseudo-counts (Bellemare et al., 2016; Ostrovski et al., 2017), mimicking optimism, have been applied with positive results to deep architectures. Likewise, with the idea of approximating a posterior distribution, *Bootstrapped DQN* (Osband et al., 2016a) and *Bayesian DQN* (Azizzadenesheli et al., 2018) succeeded in solving challenging Atari games. Recently, new results of sample-efficiency beyond tabular domains have been derived (Jiang et al., 2017).

4.7 Experiments

In this section, we provide an experimental evaluation of WQL on tabular domains along with some preliminary results on Atari games (implementation details are reported in Appendix B.2).

4.7.1 Tabular Domains

We evaluate WQL on a set of RL tasks designed to emphasize exploration: the Taxi problem (Dearden et al., 1998), the Chain (Dearden et al., 1998), the River Swim (Strehl and Littman, 2008), and the Six Arms (Strehl and Littman, 2008). We extensively test several WQL variants that differ on: i) the Q-posterior model (Gaussian G-WQL vs particle P-WQL); ii) the exploration strategy (optimistic OX vs posterior sampling PX), iii) the estimator of the maximum (ME, OE, and PE). We compare these combinations with the classic Q-learning (QL, Watkins, 1989b) (Boltzmann exploration), Bootstrapped

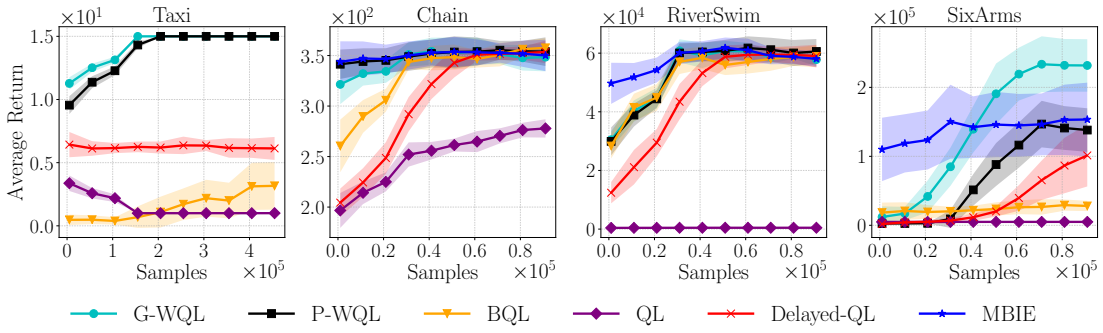


Figure 4.1: Online average return as a function of the number of samples, comparison of P-WQL and G-WQL with QL, BQL, Delayed-QL, and MBIE-EB. 10 runs, 95% c.i.

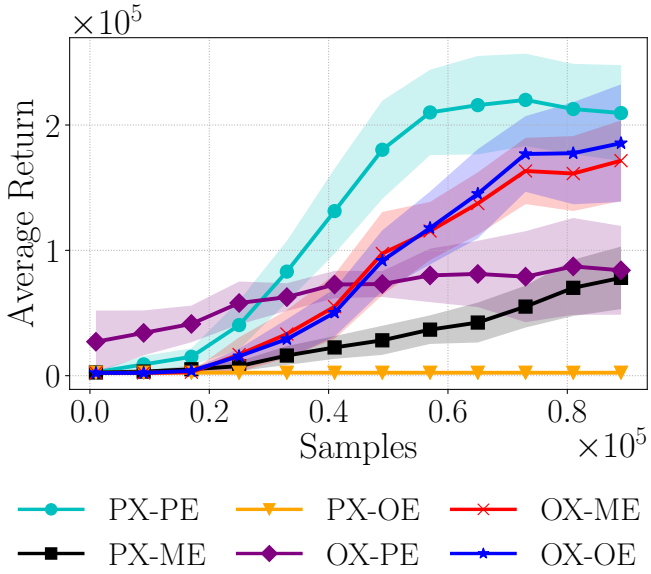


Figure 4.2: Online average return as a function of the number of samples for the different versions of G-WQL algorithm. 10 runs, 95% c.i.

Q-learning (BQL, Osband et al., 2016a) both with the double estimator (Van Hasselt, 2010), Delayed Q-learning (Delayed-QL, Strehl et al., 2006) and MBIE-EB (Strehl and Littman, 2008).[7]

Figure 4.1 shows the *online* performance on the considered tabular tasks. While we tried all the WQL variants, for clarity, we present here only the best combination of exploration strategy and maximum estimator for both Gaussian and particle models (complete results are reported in Appendix B.2). We can see that WQL learns substantially faster than classical approaches, like QL, in tasks that require significant exploration, such as Taxi, Six Arms, or River Swim. Our algorithm also outperforms BQL in most tasks, except in the River Swim, where performances are not substantially different. Finally, we can see that across all the tasks WQL displays a faster learning curve w.r.t. to Delayed-QL. MBIE-EB outperforms WQL in small domains like Chain and RiverSwim, but not in SixArms. MBIE-EB was not tested on the Taxi domain as the number of states (~ 200) makes the computational time demands prohibitive. We cross-validate the hyperparameters of Delayed Q-Learning and MBIE-EB.

Among the variants of WQL, we discovered that the choice of the exploration strategy and the maximum estimator are highly task dependent. However, we can see a general pattern across the tasks. As intuition suggests, being the exploration strategy and the maximum estimator closely related, the best combinations are: OX exploration with OE estimator and PX exploration with PE estimator. We illustrate in Figure 4.2 all the possible combinations of G-WQL on Six Arms, a domain in which exploration is essential. We can notice that the "hybrid" combinations, such as OX with PE and PX with OE are significantly outperformed by the more "coherent" ones.

Effect of initialization in Particle WQL

Finally, We explore the effect of the initialization of the prior distributions in the particle case. We argue that the particle algorithm performs better when used with particles equally spaced in a given interval $[Q_{\min}, Q_{\max}]$. To show this we added noise to these equally spaced particles and ran the learning algorithm in the same domain. Figure 4.3 shows our results as a function of α, denoting how spaced are the particles between each other. More specifically, $\alpha = 0$ means the particles were drawn uniformly random in the interval $[Q_{\min}, Q_{\max}]$ and $\alpha = 1$ means the particles are equally spaced in this interval. Any value in between is a combination of the two. For each α, we averaged the learning

[7] We are considering a discounted setting, thus, several provably efficient algorithms, like UCRL2 (Jaksch et al., 2010a), PSRL (Osband et al., 2013), RLSVI (Osband et al., 2016a), optimistic Q-learning (Jin et al., 2018) and UCBVI (Azar et al., 2017), cannot be compared as they consider either average reward or finite-horizon setting.

curves of the agent. It seems clear to us that using equally spaced particles to represent the prior yields better results.

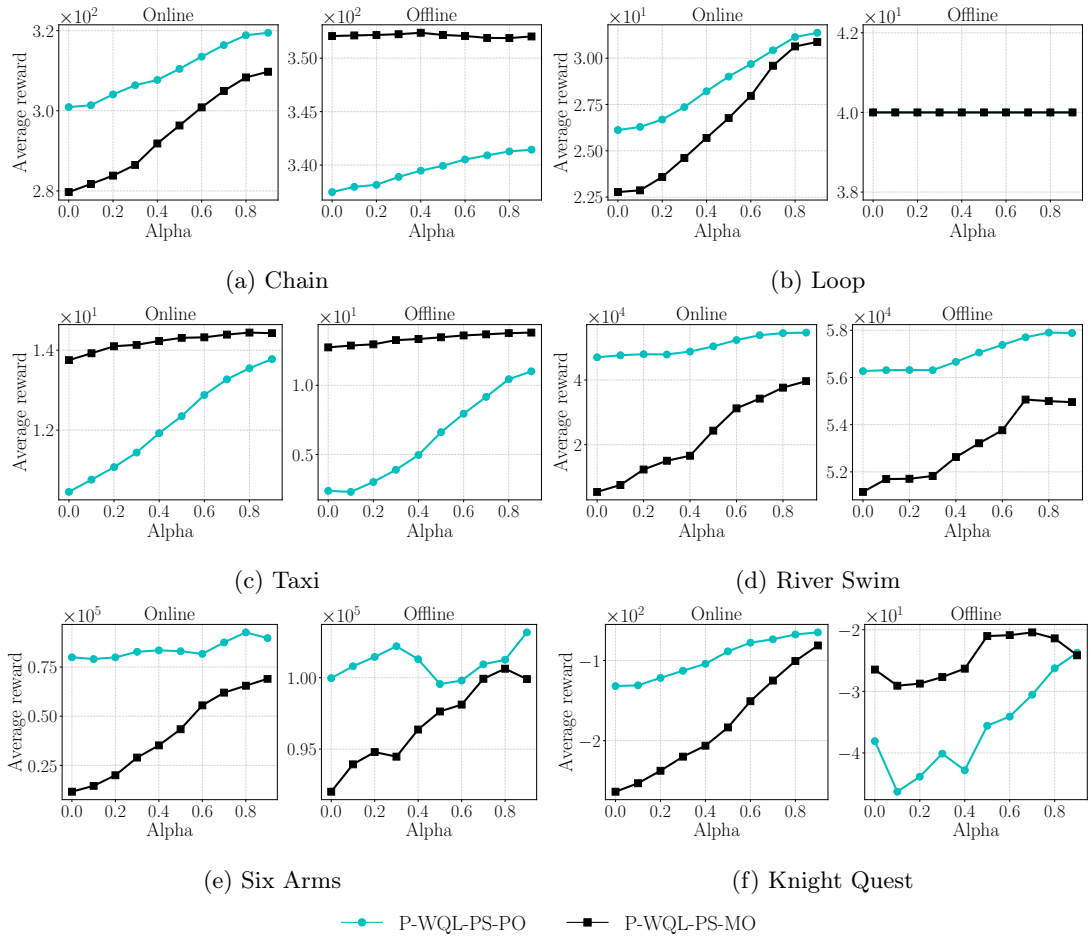


Figure 4.3: Effects of the initialization of the particles in Particle WQL. 10 runs.

4.7.2 Extending WQL to Deep Reinforcement Learning

As discussed earlier, the main advantage of modeling the epistemic uncertainty directly as posteriors over the value functions and applying variational updates is that this allows to model, propagate and generalize uncertainty in domains with continuous states where function approximation is required. For this reason, in this section, we present a simple adaption of WQL in the deep RL setting. We focus now only in the particle model, leaving the Gaussian model for the following chapter where we will better study the complications related to generalizing uncertainty over value functions.

We adapted WQL with the particle model to be used paired with deep architectures. For this purpose, similarly to Bootstrapped DQN (BDQN, Osband et al., 2016a), we

Algorithm 6 Particle DQN.

Input: a prior distribution $\{x_i\}_{i=1}^M$, a step size schedule $(\alpha_t)_{t\geq 0}$, an exploration policy schedule $(\pi_t)_{t\geq 0}$

Initialize a Q-function network with M outputs $\{Q_j\}_{j=1}^M$ and parameters $\boldsymbol{\theta}$ and the target network with parameters $\boldsymbol{\theta}^- = \boldsymbol{\theta}$

for $t = 1, 2, ...$ **do**

Take action $A_t \sim \pi_t(\cdot|S_t; \boldsymbol{\theta})$

Store transition $(S_t, A_t, S_{t+1}, R_{t+1})$ in the replay buffer

Sample random a batch of transitions $(S_l, A_l, S_{l+1}, R_{l+1})$ from the replay buffer

Compute targets $y_j(S_{l+1}) = \mathbb{E}_{A\sim\overline{\pi}(\cdot|S_{l+1})}[Q_j(S_{l+1}, A; \boldsymbol{\theta}^-)]$ for each output Q_j where $\overline{\pi} \in \{\overline{\pi}^M, \overline{\pi}^O, \overline{\pi}^P\}$ as in Section 4.3.3

Perform a gradient descent step w.r.t. $\boldsymbol{\theta}$ on the objective $\sum_{j=1}^M (y_j(S_{t+1}) - Q_j(S_l, A_l; \boldsymbol{\theta}))^2$ and using the step size α_t

Periodically update target network $\boldsymbol{\theta}^- = \boldsymbol{\theta}$

end for

use a network architecture with a head for each particle while the convolutional layers are shared among them. We compare the resulting algorithm, which we call *Particle DQN* (PDQN), with Double DQN (DDQN, van Hasselt et al., 2016), a classic benchmark in Deep-RL, and Bootstrapped DQN, specifically designed for deep exploration using Q-posteriors. To compare algorithms we consider *offline* scores, i.e., the scores collected using the current greedy policy. The goal of this experiment, conducted on three Atari games, is to prove that WQL, although designed to work in finite environments, can easily be extended to deep networks with potentially good results. We test the algorithms using the Arcade Learning Environment (ALE). Each step of the agent corresponds to four steps of the emulator, where the same action is repeated. The reward values observed by the agents are clipped between -1 and 1 for stability. We evaluate our agents and report performance based upon the raw scores and not the discounted scores. As it is common in literature, we do not show the online performance of the agent during training. We show the scores collected, when exploiting the greedy policies derived from the Q-function after each training period.

The *convolutional* part of the network used is identical to the one used in Osband et al. (2016a). The input to the network is $4 \times 84 \times 84$ tensor with a rescaled, grayscale version of the last four observations. The first convolutional layer has 32 filters of size 8 with a stride of 4. The second layer has 64 filters of size 4 with stride 2. The last layer has 64 filters of size 3. We split the network beyond the final layer into $M = 10$ distinct heads, each one is fully connected and identical to the network in Osband et al. (2016a). This consists of a fully connected layer to 512 units followed by another fully connected

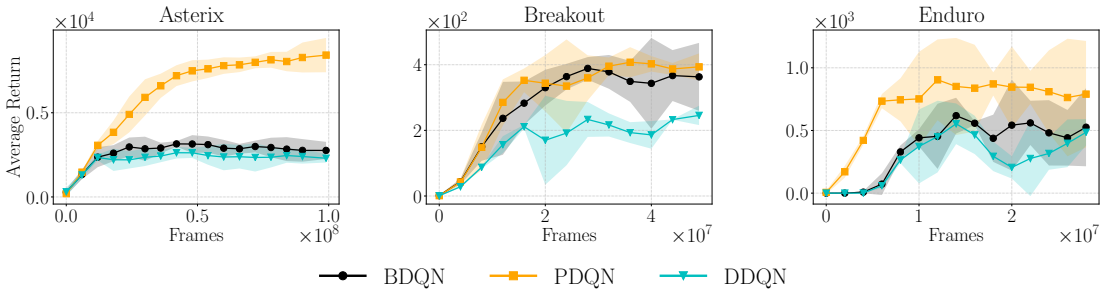


Figure 4.4: Offline average return of the greedy policy as a function of the number of collected frames, comparing PDQN, DDQN and BDQN on Asterix, Enduro and Breakout games. 5 runs, 95% c.i.

layer to the Q-Values for each action. The fully connected layers all use *Rectified Linear Units* (ReLU) as a *non-linearity*. We trained the networks with *RMSProp* optimizer. The discount was set to $\gamma = 0.99$, the number of steps between target updates was set to $\tau = 10000$ steps. The agents were evaluated every 1M frames. The *experience replay* contains the 1M most recent transitions. We update the network every 4 steps by randomly sampling a minibatch of 32 transitions from the replay buffer to use the exact same minibatch schedule as Bootstrapped DQN.

An important question is how to initialize the heads of the deep network. In the tabular case, we initialized the particles equally spaced in the interval $[Q_{\min}, Q_{\max}]$. We found it is equally simple to extend this in the deep RL setting. We initialized the networks heads near the same interval by setting the bias of the last layer to the desired values. In Figure 4.4, we can see that PDQN, compared to BDQN and DDQN, manages to achieve higher scores in Asterix and Enduro, where exploration is needed, while achieving similar scores in Breakout. A relevant feature of PDQN is the particle initialization interval. Indeed, a narrower initial interval causes faster learning but might lead to premature convergence. In this sense, the initial interval becomes a hyperparameter of PDQN, which influences the amount of exploration and it is likely task-dependent. The pseudocode of PDQN is shown in Algorithm 6.

4.8 Discussion and Conclusions

In this chapter, we introduced Wassertein TD-Learning, a novel framework for TD-learning that accounts for and propagates epistemic uncertainty and Wasserstein Q-Learning, a novel reinforcement learning algorithm designed to address challenges associated with efficient exploration in model-free RL. Our approach revolves around modeling uncertainty in the estimated Q-function through approximate posterior distributions which we call Q-posteriors. We developed a variational method to propagate uncertainty across state-action pairs during Temporal Difference learning, leveraging Wasserstein barycenters and avoiding costly classical Bayesian updates which assume independence of samples. Despite potential theoretical challenges from a Bayesian perspective, this variational update method enabled explicit control over uncertainty updates, and it allowed us to demonstrate the convergence rate of a modified version of our algorithm. Additionally, the reduced computational cost facilitated the extension of our algorithm to incorporate function approximators.

The experimental evaluation allowed us to appreciate the properties of WQL. In tabular domains, whenever exploration is really necessary, our approach is able to significantly outperform TD methods even if designed specifically for exploration (e.g., Bootstrapped Q-Learning and Delayed Q-Learning). Although preliminary, the results on the Atari games are promising and need to be further investigated in order to make WQL scale on complex environments. We believe that our algorithm contributes to bridging the gap between theory and practice of exploration in RL. WQL is a theoretically grounded method, equipped with guarantees in the average loss setting, but, at the same time, it is a very simple algorithm, easily extensible to deal with continuous domains.

It is important to underline the differences between our proposed framework with more traditional model-based Bayesian RL methods. While we make use of probability distributions representing the uncertainty over the Q-value estimates, similar to Bayesian methods, WTD (and WQL consequently) should not be thought as a Bayesian method. Indeed, in the update rules derived in our work, we do not make use of the Bayes rule for updating our distributions as in classical Bayesian methods. These comes for two main reasons. Firstly, as discussed in Sections 4.4, we are interested in deriving online model-free algorithms that learn from samples of experience of the form $(s_t, a_t, r_{t+1}, s_{t+1})$ and apply bootstrapping to estimate the values in the next states s_{t+1}. The bootstrapping operation in the next states, makes all samples of experience intrinsically dependent which invalidates the assumptions of the update rule. Secondly, our variational updates allow us to effectively control the speed of the updates of the uncertainty estimates, allowing for an easier theoretical understanding of the proposed algorithm and its convergence rate. On

the other hand, in model-based Bayesian RL, a more rigourous approach is usually taken, where a prior belief over model parameters is used to derive a posterior over the value functions. This generally allows for tighter regret bounds and more efficient algorithms in terms of sample complexity, but generally comes with an increased computational complexity in simple cases or intractable updates in more complex scenarios. On the other hand, having an explicit dependence of the algorithm on the choice of the prior over the value function, generally gives more insight on the impact of the prior on empirical performance too. The main difference compared to WTD is that, given a prior over value functions, in Bayesian RL, we are tasked with computing the exact posterior over value functions after observing new experience. This is intractable in all but the simplest of cases, so generally this true posterior is approximated when devising practical algorithms. In the case of WTD, we take the approach of avoiding the computation of the exact posterior and substitute it with the Wasserstein Barycenter of the input priors over value functions. While this makes our uncertainty estimates different from typical bayesian epistemic uncertainty estimates, they still quantify the absense of information due to limited data, and degenerate to 0 as we collect more samples and estimate the correct value functions as demonstrated by our theoretical analysis. The question whether to take a more classical Bayesian approach or take a more variational approach on uncertainty estimates is still open and outside the scope of this thesis. Since our main goal remains to devise theoretically grounded algorithms that employ uncertainty estimates for faster learning, while also being easily extensible in domains where function approximation is needed, we further explore the use of the WTD framework as a means of more efficient exploration in RL.

In the next chapter, we investigate further how to extend the proposed algorithm to the continuous action setting. This will allow us also to better study the effect of our variational updates on the uncertainty estimates and whether the exploration properties that we achieve in tabular domains can be successfully transferred empirically in domains with complex state-action spaces where exploration is crucial. Moreover, we believe generalizing uncertainty estimates is fundamentally different from generalizing the value function estimates. For this reason, we expand our focus on continuous domains, where we can better study how our Q-posteriors generalize uncertainty across the state-action space.

CHAPTER 5

The Case of Infinite States and Actions

5.1 Introduction

Reinforcement Learning (RL, Sutton and Barto, 2018) is one of the most widely used frameworks for solving sequential decision-making problems, especially in model-free settings, where a model of the environment dynamics is not available. When an agent acts in an uncertain environment, it faces the choice between *exploring* with the hope of discovering more profitable behaviors or *exploiting* the current information about the actions' values. This exploration-exploitation dilemma is particularly challenging in *continuous-state* spaces, where *function approximation* is required to generalize across states, and an accurate estimate of the uncertainty on the value estimates is not available point-wise. *Continuous-action* tasks pose additional challenges since most exploration methods require the maximization of some objective (e.g., upper bound of the Q-value) over the action space. While in the discrete case, this maximization can be performed by enumeration, in the continuous case it requires solving a complex optimization problem, increasing the computational demands.

Actor-Critic (AC) methods (Haarnoja et al., 2018; Ciosek et al., 2019; Schulman et al., 2015) represent the current state of the art for continuous control. Despite their

widespread adoption, these methods still suffer from high sample complexity. Efficient exploration strategies have been extensively studied in the literature as a means of reducing sample complexity mainly in tabular domains (Auer et al., 2008; Ian et al., 2013; O'Donoghue et al., 2018; Metelli et al., 2019). Classical exploration strategies, like ϵ-greedy or Boltzmann (Sutton and Barto, 2018), inject noise around the current greedy policy to enforce exploration. Although in simple settings, this is enough to guarantee convergence (Szepesvári, 1997), this exploration strategy is not efficient in the general case.

Another line of approaches considers the *maximum entropy* setting to improve exploration and avoids the *deterministic collapse* of the policies, e.g., Soft Actor Critic (SAC, Haarnoja et al., 2018). In this setting, stochastic policies are preferred by optimizing the expected return regularized with an entropy term. Moreover, to improve stability and avoid over-estimation bias (Hasselt, 2010; Fujimoto et al., 2018) most methods use an ensemble of (at least 2) parameterized critics (trained with the same samples and differ only by the initialization of the parameters) in order to estimate a "lower bound" of the value functions. This also represents a form of *undirected* exploration since the policies are forced to be stochastic, thanks to the entropy bonus, but the induced noise does not consciously shift its focus towards promising regions of the state space.

A common trend in the RL literature consists of endowing existing methods with some form of uncertainty quantification and using it to perform *directed* exploration while focusing on the most promising regions. For instance, *optimistic* approaches have been applied to both Q-learning (Jin et al., 2018) and SAC. In particular, a recent extension of Soft Actor-Critic (SAC, Haarnoja et al., 2018), Optimistic Actor-Critic (OAC, Ciosek et al., 2019), proved to improve sample efficiency over the standard SAC. As we shall see, SAC is particularly impacted by local maximums of the return due to this undirected exploration, since it only depends on the entropy regularized objective for exploration. Indeed, uncertainty quantification is a fundamental step in defining efficient exploration strategies. Exploration strategies, coming from the Multi-Armed Bandit (MAB, Lattimore, 2020) literature, have been extended for the RL settings, starting from tabular domains (Auer et al., 2008; Ian et al., 2013; Metelli et al., 2019), with theoretical guarantees on the sample complexity and/or regret. In turn, they have been extended to the Deep Reinforcement Learning (DRL) settings too, but the guarantees no longer hold up. Ensemble methods allow quantifying the uncertainty but do not *propagate* it across the stat-action space when performing the critic updates. Uncertainty propagation is a fundamental tool of any principled uncertainty estimation approach since most AC methods rely on bootstrapping when updating the critics. This results in Q-value estimates that also incorporate uncertainty about the bootstrapped values.

Distributional RL (O'Donoghue et al., 2018) allows for uncertainty propagation but considers only aleatoric uncertainty, being aimed at estimating the full return distribution. These methods propagate uncertainty by means of the distributional Bellman equation, whose goal is to estimate the full return distribution. This is not straightforward in practice, and, furthermore, it is not strictly necessary in the classical RL setting where the goal is to maximize the expected return. To the best of our knowledge, the only method capable of propagating the epistemic uncertainty, without the need to learn the return distribution is Wasserstein Q-learning, introduced in the previous Chapter.

In this chapter, we address the problem of uncertainty estimation and propagation in the context of continuous-action RL. Starting from the methodology introduced in the previous chapter, we devise a novel actor-critic algorithm, *Wasserstein Actor-Critic* (WAC), which employs Q-posteriors both to quantify uncertainty on the critic estimates to drive exploration, as well as a tool to propagate it across the state-action space (Section 5.3). The Q-posteriors quantify the epistemic uncertainty of the Q-values and incorporate both the uncertainty due to the empirical estimate of the stochastic transition and immediate reward sample, as well as the Q-value uncertainty of the next states imported during the bootstrapping of the Temporal Difference (TD, Sutton and Barto, 2018) updates.

Furthermore, we consider some practical problems that arise while quantifying uncertainty by means of Q-posteriors coped with function approximators, especially neural networks. To this end, we propose a regularization approach for the uncertainty networks to avoid the collapse of the uncertainty estimates due to uncontrolled generalization (see Section 5.4). WAC uses the Q-posteriors to explore efficiently, by optimizing an upper bound of the Q-values. Unlike OAC, which employs bootstrapped uncertainty estimates from an ensemble of critics (two) to define the upper bound, we employ the Q-posteriors, which will eventually shrink to point estimates. Furthermore, WAC recovers SAC for a specific hyperparameter configuration and, more importantly, is able to explore more efficiently with negligible additional computational costs. After reviewing the related literature (Section 5.5), we present a thorough experimental evaluation over some simple 1D navigation domains, as well as some MujoCo (Todorov et al., 2012) tasks designed for exploration to assess the effect of uncertainty estimation and propagation on exploration and sample complexity (Section 5.6).

5.2 Preliminaries

In this section, we recall some fundamental notions from which we built upon.

5.2.1 Actor-Critic Methods

In large continuous domains, it is required to generalize across the state-action space by means of parameterized policies and value functions. One of the most used frameworks in this setting, especially for continuous action spaces, is the Actor-Critic (Schulman et al., 2015; Haarnoja et al., 2018; Ciosek et al., 2019) family of algorithms. In this framework, the agent maintains a parameterized value-function $Q_{\boldsymbol{\omega}}$ (*critic*) to estimate the value of the current (or a given target) policy, and a parameterized policy $\pi_{\boldsymbol{\theta}}$ (*actor*), trained through gradient descent. In particular, SAC, employs an *entropy-regularized* architecture, i.e., the agents optimize a modified objective regularized with the entropy of the policy favoring stochastic policies over deterministic ones, shown in Equation 5.2. Specifically, it maintains two parameterized action-value functions $\{Q_{\boldsymbol{\omega}_1}, Q_{\boldsymbol{\omega}_2}\}$ to estimate the entropy-regularized value function of policy $\pi_{\boldsymbol{\theta}}$. They are trained on the same samples and differ only on the initialization of $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$. The actor optimizes a "lower bound" of the action-value function, $Q_{LB}(s,a) = \min\{Q_{\boldsymbol{\omega}_1}(s,a), Q_{\boldsymbol{\omega}_2}(s,a)\}$. To update the critic, given a sample (s,a,r,s'), SAC uses the SARSA (Sutton and Barto, 2018) update rule, $Q_{\{\boldsymbol{\omega}_1,\boldsymbol{\omega}_2\}}(s,a) \leftarrow r + \gamma Q_{LB}(s',a')$, where $a' \sim \pi_{\boldsymbol{\theta}}(s')$. Specifically, SAC maintains experience collected with previous policies $\pi_{\boldsymbol{\theta}}$ in a *replay buffer* $\mathcal{D}$ (Sutton and Barto, 2018). The critic is trained to minimize the (entropy regularized) Bellman error over this replay buffer, as follows:

$$J_{\mathrm{C}}(\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2\}) = \mathop{\mathbb{E}}_{s,a,r,s' \sim \mathcal{D}} \left[\left(Q_{\{\boldsymbol{\omega}_1,\boldsymbol{\omega}_2\}}(s,a) - (r + \gamma \widetilde{Q}(s',a')) \right)^2 \right], \tag{5.1}$$

where $\widetilde{Q}(s,a) = \overline{Q}_{LB}(s,a) - \alpha \log \pi_{\boldsymbol{\theta}}(s',a')$, $\overline{Q}_{LB}(s,a) = \min\{Q_{\overline{\boldsymbol{\omega}}_1}(s,a), Q_{\overline{\boldsymbol{\omega}}_2}(s,a)\}$ is the lower bound of the Q-values given by two target networks which are updated slowly to improve stability (Mnih et al., 2015), $a' \sim \pi_{\boldsymbol{\theta}}(s')$, and $\alpha > 0$ specifies the level of entropy regularization. The actor-network is trained to optimize an entropy-regularized objective. Since the target Q-function is a parameterized function approximator, the policy can directly follow the gradient of the critic:

$$J_A(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ a_t \sim \pi_{\boldsymbol{\theta}}(s_t)}} \left[\log \pi_{\boldsymbol{\theta}}(s_t,a_t) - Q_{LB}(s_t,a_t) \right]. \tag{5.2}$$

For more details, we refer the reader to the original SAC paper (Haarnoja et al., 2018).

Algorithm 7 Wasserstein Actor-Critic.

Input: critic parameters $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2$, policy parameters $\boldsymbol{\theta}, \boldsymbol{\theta}^T$
Initialize $\mathcal{Q}_{\{1,2\}}(s,a)$ with the prior $\mathcal{Q}_0$
Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
for epoch $= 1, 2, ...$ **do**
  for $t = 1, 2, ...$ **do**
    Take action $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$
    Observe s_{t+1} and r_{t+1}
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_{t+1}, s_{t+1})\}$
  end for
  $\sigma_{\text{old}}^{\{1,2\}} \leftarrow \sigma_{\boldsymbol{\omega}_{\{1,2\}}}$
  for iteration $= 1, 2, ...$ **do**
    Update critic weights $\boldsymbol{\omega}_{\{1,2\}}$ using Equation (5.7)
    Update actor (resp. target) weights $\boldsymbol{\theta}$ (resp. $\boldsymbol{\theta}^T$) using Equation (5.5) (resp. Equation (5.6))
  end for
end for

5.3 Wasserstein Actor-Critic

In this section, we introduce Wasserstein Actor-Critic (WAC), which extends WQL to handle environments with continuous-action spaces. We present the algorithm, define the update rules, and a regularization for the uncertainty estimates.

5.3.1 Distributional Critic

For each state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, we maintain an approximate distribution $Q(s,a) \sim \mathcal{Q}(s,a)$, to model the uncertainty estimate on the value function. While these distributions will generally depend on the aleatoric uncertainty of the environment (state transition and reward), our updates will vanish the variance as we collect samples. This represents our main difference w.r.t. Distributional RL (Bellemare et al., 2017b), as we do not require learning the whole return distribution, while still propagating uncertainty across the state-action space. More specifically, given a replay buffer of past behavior $\mathcal{D}$, our critic minimizes the L_2-Wasserstein distance between the Q-posterior $\mathcal{Q}_{\boldsymbol{\omega}}$ and the target posterior $r + \gamma \mathcal{Q}_{\overline{\boldsymbol{\omega}}}$, defined through the target parameters $\overline{\boldsymbol{\omega}}$ and target policy $\pi_{\boldsymbol{\theta}^T}$:

$$J_C(\boldsymbol{\omega}) = \mathop{\mathbb{E}}_{s,a,s',r \sim \mathcal{D}} \Big[W_2 \Big(\mathcal{Q}_{\boldsymbol{\omega}}(s,a), \quad r + \gamma \mathcal{Q}_{\overline{\boldsymbol{\omega}}}(s', \pi_{\boldsymbol{\theta}^T}(s')) \Big)^2 \Big]. \tag{5.3}$$

Different flavors of the algorithm can be proposed, based on the combination of: (i) distribution classes $\mathscr{Q}$, (ii) behavioral policy $\pi_{\boldsymbol{\theta}}$, and (iii) target policy $\pi_{\boldsymbol{\theta}^T}$. We focus on

optimistic exploration, that requires optimizing upper bounds. Moreover, although other distribution classes, like particle models, could be employed, we limit our discussion to Gaussian posteriors, as their parametrization allows for direct control over the distribution variance.

Similar to WQL, we maintain a parameterized distributional critic using a function approximator (e.g., neural network) that outputs the parameters of the distribution. For the Gaussian case, $Q(s,a) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}(s,a), \sigma_{\boldsymbol{\omega}}(s,a))$, the Wasserstein distance has a closed form, and the critic objective becomes:

$$J_C(\boldsymbol{\omega}) = \mathop{\mathbb{E}}_{s,a,s',r\sim\mathcal{D}} \Big[\big(\mu_{\boldsymbol{\omega}}(s,a) - (r + \gamma \widetilde{\mu}_{\overline{\boldsymbol{\omega}}}(s', \pi_{\boldsymbol{\theta}^T}(s')))\big)^2 + \big(\sigma_{\boldsymbol{\omega}}(s,a) - \gamma \sigma_{\overline{\boldsymbol{\omega}}}(s', \pi_{\boldsymbol{\theta}^T}(s'))\big)^2 \Big], \tag{5.4}$$

where $\widetilde{\mu}_{\overline{\boldsymbol{\omega}}}(s,a) = \mu_{\overline{\boldsymbol{\omega}}}(s,a) - \alpha \log \pi_{\boldsymbol{\theta}^T}(s,a)$. In practice, $\mu_{\boldsymbol{\omega}}$ and $\sigma_{\boldsymbol{\omega}}$ can use either a shared network architecture or two different networks. We initialize the posterior networks using the bias of the last layer of the network. If the reward function is limited in the interval $[r_{\min}, r_{\max}]$, the Q values will be in the range $[q_{\min}, q_{\max}]$ with $q_{\min} = r_{\min}/(1-\gamma)$ and $q_{\max} = r_{\max}/(1-\gamma)$. We therefore initialize the uncertainty networks to $\sigma_0 = (q_{\max} - q_{\min})/\sqrt{12}$, i.e. variance of the Gaussian minimizing the KL divergence with the uniform distribution in $[q_{\min}, q_{\max}]$ (Metelli et al., 2019).

5.3.2 Policy Optimization

Behavioral Policy

The actor in WAC is updated by optimizing an *upper bound* $U^{\delta}_{\boldsymbol{\omega}}$ of the estimated Q-value, which we can efficiently compute using Gaussian posterior: $U^{\delta}_{\boldsymbol{\omega}}(s,a) = \mu_{\boldsymbol{\omega}}(s,a) + \sigma_{\boldsymbol{\omega}}(s,a)\Phi^{-1}(\delta)$, where Φ^{-1} is the quantile function of the standard normal and $\delta \in (0,1)$. When actions are finite, no actor is needed, as we can compute the maximum by enumeration. However, in the continuous-action case, we need an actor that follows $U^{\delta}_{\boldsymbol{\omega}}(s,a)$, which is differentiable in $\boldsymbol{\omega}$, leading to the minimization of the objective:

$$J_A(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{s_t\sim\mathcal{D} \\ a_t\sim\pi_{\boldsymbol{\theta}}(s_t)}} \Big[\log \pi_{\boldsymbol{\theta}}(s_t,a_t) - U^{\delta}_{\boldsymbol{\omega}}(s_t,a_t) \Big], \tag{5.5}$$

where $\boldsymbol{\theta}$ are the parameters of the behavioral policy.

Target Policy

We propose two alternatives for the target policy $\pi_{\boldsymbol{\theta}^T}$, corresponding to different estimators for the target posterior $\mathcal{T}_t$. First, we can use the same policy we use for exploration, i.e., $\boldsymbol{\theta} = \boldsymbol{\theta}^T$, like SAC. This has the advantage of not requiring a second parameterized

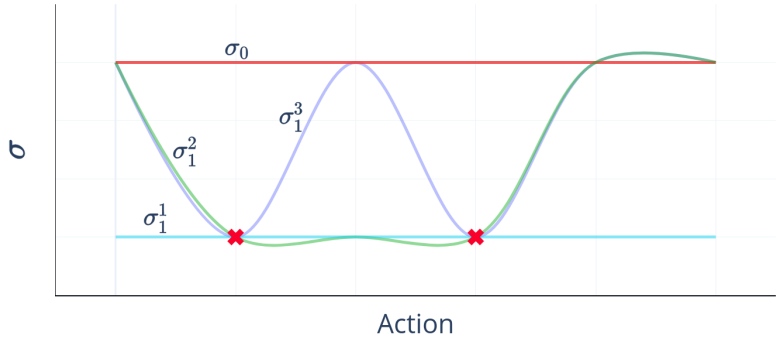


Figure 5.1: Example of uncertainty estimates. σ_0 shows the initial constant high value.

policy. We call this version *Optimistic Estimator-WAC* (OE-WAC), which represents an on-policy algorithm. Alternatively, we can use a *greedy* policy that optimizes the expected value of the Q-posteriors (the mean critic $\mu_{\boldsymbol{\omega}}(s, a)$ in the Gaussian case). In this case, the target policy minimizes:

$$J_T(\boldsymbol{\theta}_T) = \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ a_t \sim \pi_{\boldsymbol{\theta}^T}(s_t)}} \left[\log \pi_{\boldsymbol{\theta}^T}(s_t, a_t) - \mu_{\boldsymbol{\omega}}(s_t, a_t) \right]. \tag{5.6}$$

We call this version *Mean Estimator-WAC* (ME-WAC). The best version to use between the two is task-dependent. Generally, OE-WAC is more suitable for environments that require large exploration, whereas ME-WAC is more suitable for simpler environments where OE-WAC might over-explore and might suffer from some instability.

We underline that our distributional critic maintains uncertainty about the Q^* and not about the Q-function of the current policy $Q^{\pi_{\boldsymbol{\theta}}}$. Indeed, the method starts with an initial high-uncertainty estimate and updates it as we collect samples from the environment. In the tabular case, it can be proven that these upper bounds on the value of Q^* are valid with high probability for every timestep t, under some conditions on the learning rate α_t (Metelli et al., 2019). When extending the method to DeepRL, these guarantees are no longer valid since the uncertainty estimates are outputs of general function approximators, and local updates are no longer possible.

5.4 Regularized Uncertainty Estimation

Our Q-posteriors are initialized to high uncertainty at the beginning of the learning process. Since they represent epistemic uncertainty, their variance will shrink as we observe more samples. This is apparent in Equation (5.4), where the targets $\sigma_{\boldsymbol{\omega}}$ are multiplied with γ. In tabular settings, the updates are localized, i.e., they affect a single state-action pair, without interfering with the others. However, when function

approximators are involved, generalizing uncertainty in an uncontrolled way might cause non-visited areas of the state-action space to take low uncertainty values, which might be undesired.[1] Consider the example in Figure 5.1 showing the uncertainty estimate as a function of the action, in a fixed state s. Starting from an initial high constant estimate of σ_0, at the beginning of the learning process, we will observe samples like the red crosses in the figure, i.e., with lower uncertainty since it gets shrunk with γ. Among all the possible fitting lines, we would prefer an estimate like σ_1^3, which keeps high uncertainty in unseen regions, and would like to avoid failures like σ_1^1. This requires controlling the "smoothness" properties of the approximator. To avoid the additional computational burden, we propose a simple scheme based on *synthetic* samples. Specifically, we periodically save the weights of the uncertainty network σ_{old} and use it as the target for state-action pairs drawn *uniformly* from the state-action space. More formally, our distributional critic minimizes:

$$J'_C(\boldsymbol{\omega}_{\{1,2\}}) = J_C(\boldsymbol{\omega}_{\{1,2\}}) + \qquad \lambda \mathop{\mathbb{E}}_{s,a\sim\mathcal{U}(\mathcal{S}\times\mathcal{A})} \left[\left(\sigma_{\boldsymbol{\omega}_{\{1,2\}}}(s,a) - \sigma_{\text{old}}(s,a) \right)^2 \right], \tag{5.7}$$

where $J_C(\boldsymbol{\omega}_{\{1,2\}})$ is defined in Equation (5.4) and $\lambda \geq 0$ defines the relative weight of the regularization. Furthermore, in practice we add a second parameter, $\rho \in [0,1]$ which represents the fraction of *fake* samples (w.r.t. the samples used for $J_C(\boldsymbol{\omega}_{\{1,2\}})$) drawn for regularization. Specifically, if we estimate $J_C(\boldsymbol{\omega}_{\{1,2\}})$ using N samples from replay buffer $\mathcal{D}$, we will estimate the expectation in Equation (5.7) with $M = \rho N$ samples from $\mathcal{U}(\mathcal{S} \times \mathcal{A})$. Algorithm 7 reports the pseudocode of WAC, embedding the regularized uncertainty estimation.

To investigate the effectiveness of the regularized uncertainty loss, on an illustrative example, we trained two different agents, in a one-dimensional Linear Quadratic Regulator (LQG, Dorato et al., 2000). This task has a one-dimensional state and action spaces, which allows us to visualize the uncertainty estimates. Figure 5.2 shows the resulting uncertainty estimates. On the left, we show the empirical state-action visitation distribution. The agent starts in one of the borders of the state space and has to reach the center in a few steps while calibrating the actions. This is apparent in the histogram, with the highest densities in the borders and the center. We consider it desirable to obtain uncertainty estimates that mirror these state-action densities, as the epistemic uncertainty is inversely proportional to the state-action visitation. While in both cases, the state-action densities are similar, the uncertainty estimates are completely different. In Figure 5.2b, we see that without regularization, the critic completely fails to represent the uncertainty. In Figure 5.2c, we can see that the regularized uncertainty critic almost

[1] This generalization phenomenon happens for the mean too, but, as visible in Equation (5.4), is particularly critical for the variance that gets updated with the next-state-action variance scaled by $\gamma < 1$.

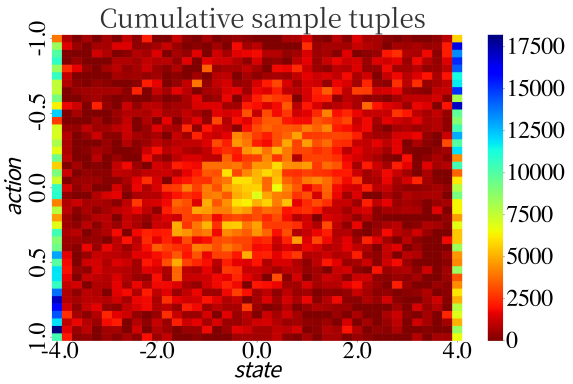
perfectly matches the state-action densities. In Section 6 and Appendix B, we show a more thorough investigation of the effect of the regularized uncertainty loss.

While WAC is a direct extension of Wasserstein TD-learning to the actor-critic architecture, it does come with some modifications (mostly to regularize the learning process). Indeed, the entropy setting we employ in the critic and actor fit is not a part of the base framework. Wasserstein TD-learning, in its pure form, would employ deterministic policies. In practice, we observed that this, coupled with optimistic exploration, caused instability in the learning process. Adding entropy regularization improves stability while not decreasing significantly the exploration capabilities of the method. Moreover, the regularization term in Equation (5.7) is added due to the use of general function approximators for generalizing uncertainty and was not found in the main Wasserstein TD framework. We use the regularization term as a way to control the "smoothness" of the function approximator. The synthetic samples do not necessarily need to be "valid" state-action pairs. They need to be in the input space of the approximator. Indeed, in our implementation, we simply uniformly sample in $[-1, 1]^{nS+nA}$ (where nS and nA are the dimensions of the state and action space, each normalized in $[-1, 1]$).

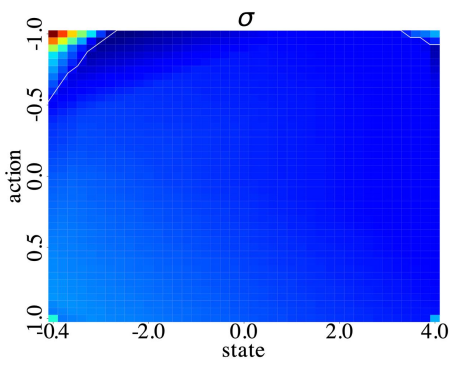
5.5 Related Literature

There exists a large body of literature studying efficient exploration techniques in RL. In the tabular settings, *provably efficient* methods have been devised, both in the model-based (Jaksch et al., 2010b; Ian et al., 2013) and model-free (Strehl et al., 2006; Jin et al., 2018) settings. These methods cannot be easily extended to the Deep RL setting, or when extensions are proposed, they lose their theoretical guarantees. In this section, we focus on tractable exploration methods proposed for Deep RL for continuous action spaces. Two main exploration frameworks exist: *uncertainty-based* methods and *intrinsic motivation* methods (Houthooft et al., 2016; Zhang et al., 2021; Mutti et al., 2021). For space reasons, we will focus our discussion on uncertainty-based methods.

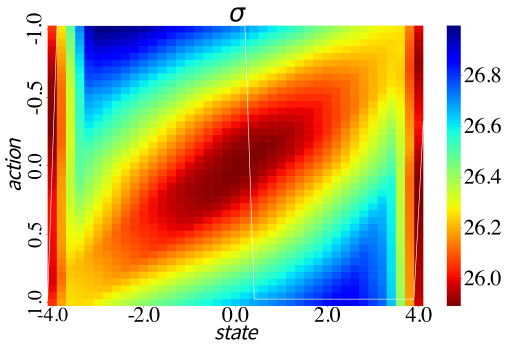
Classical value-based methods (including ACs) maintain a point estimate of the value functions for each state (or state-action pairs). Exploration policies, like ϵ-greedy or Boltzmann (Sutton and Barto, 2018), add noise around the greedy action derived from these point estimates. These methods are not efficient, mainly because the exploration is not *directed* towards unvisited regions of the state space. The entropy regularization of SAC is a form of undirected exploration too, as the policies are trained to sacrifice some returns to preserve stochastic behavior. In recent years, several methods that move away from point estimates have been proposed. Ensemble methods (Chen et al., 2021; Wang et al., 2021) implicitly model the epistemic uncertainty of the Q-value estimates



(a) State-action density



(b) Without regularization



(c) With regularization

Figure 5.2: Comparison on the uncertainty estimates after training with and without uncertainty regularization in the LQG illustrative example (action on the y axis and state on the x axis).

by maintaining multiple Q-function approximators. OAC (Ciosek et al., 2019) explicitly models the uncertainty on the value estimates by computing the variance of two critics, and it uses it to compute an exploration policy that optimizes an upper bound of the Q-values. This uncertainty estimate stems from the disagreement between the two Q-networks with different initialization. Indeed, the networks are also trained with the same samples and the same target Q-values, so any disagreement is purely due to the random initialization only. Recently, SUNRISE (Lee et al., 2021) proposes a framework to unify ensemble methods for epistemic uncertainty estimation and shows considerable performance improvements in discrete and continuous action spaces. Distributional RL, on the other hand, models the *aleatoric* uncertainty, as its goal is to estimate the whole return distribution. First proposed for problems with a discrete action space (Bellemare et al., 2017b; Dabney et al., 2018b; Mavrin et al., 2019), it has been successfully extended also to the AC setting in TOP (Moskovitz et al., 2021). TOP models both aleatoric and epistemic uncertainty and adapts the level of optimism/pessimism by means of a MAB approach. While TOP deals with uncertainty propagation, it mixes epistemic and aleatoric uncertainty while estimating the return distribution.

Tractable model-free methods based on intrinsic motivation have been proposed in recent years. Methods based on pseudo-counts (Bellemare et al., 2016; Ostrovski et al., 2017) assign exploration bonuses according to the novelty of the state-action pairs visited. While they have been applied with good results to deep architectures, they generally rely on (often pre-trained) density models, which are not straightforward to maintain. Other methods apply exploration bonuses based on the state-action visitation density of the policy. MADE (Zhang et al., 2021) adds an exploration bonus, based on the deviation of the state visitation density of the new policy from the last observed policies. While it has been applied to continuous state-action spaces, it comes with a considerable computational cost to estimate the state densities and also requires pre-training of density models.

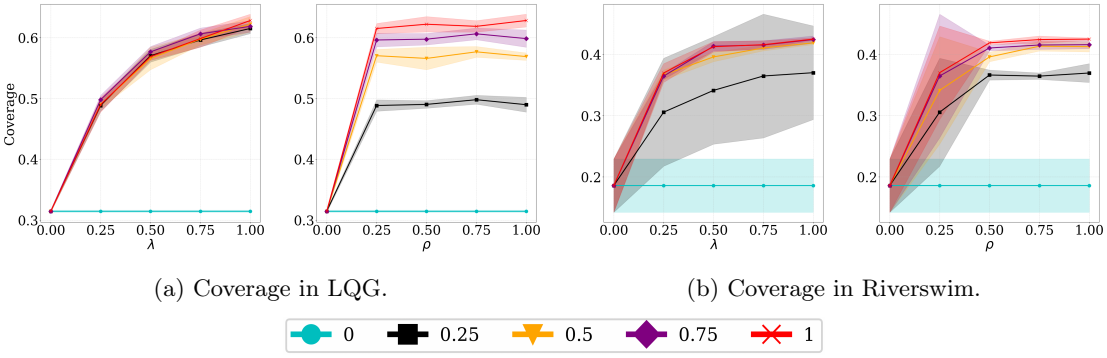
State entropy maximization (Mutti et al., 2021; Seo et al., 2021; Yarats et al., 2021) has also been applied as an incentive to explore the whole state-action space, including hard-to-reach regions. These methods generally scale better to continuous domains, as they do not explicitly need to estimate the state occupancy but only the entropy of this distribution. Numerous methods have also been proposed, with bonuses based on the information-gain (Houthooft et al., 2016; Achiam and Sastry, 2017; Pathak et al., 2019), but come with considerable computational costs to estimate these bonuses.

To the best of our knowledge, WAC is the first method able to propagate epistemic uncertainty in continuous action spaces, without the need for estimating the full return distributions.

5.6 Experiments

In this section, we present the empirical evaluation of WAC in various continuous control domains. We start from simple 1D-navigation, where we can better visualize the effects of the Q-posteriors in the learning and exploration process. In Appendix B, we show an evaluation on several standard MujoCo tasks, which show that this suite of environments does not pose significant exploration challenges. Hence, we focus our evaluation of WAC on multiple versions of the 2D navigation tasks used in (Moro et al., 2022), where we can control the "level of exploration" needed to solve the task. We consider 2 versions of the environment, one with an "informative" reward function based on the distance from a goal state (which generates local maximums of the return) and evaluate whether the different algorithms can escape these local maximums. Moreover, we consider also sparse rewards, which avoid local maximums but make exploration harder. We compare WAC to SAC and OAC since for $\delta = 0$, WAC becomes exactly SAC, and OAC also extends SAC with optimistic exploration by using an "heuristic" estimation of the uncertainty. Details on the implementation, environments, and hyperparameter tuning are presented in Appendix B.3. Our results can be reproduced using the source code in `https://github.com/amarildolikmeta/wac_explore`.

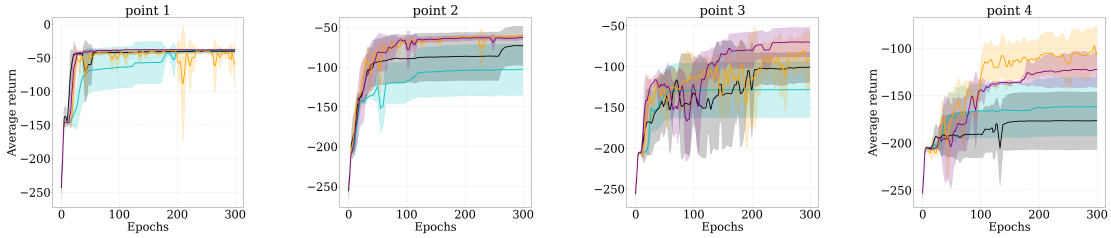
1D Navigation To measure the effect of uncertainty estimation on exploration, we keep track of the cumulative *coverage* of the state-action space, i.e., the portion of the total volume visited with relative frequency larger than $\epsilon > 0$. We consider a one-dimensional LQG, an environment with no particular exploration challenges, and a more challenging continuous-action version of the Riverswim (Strehl and Littman, 2008), where long sequences of rewardless actions are needed to reach high reward states. A full description of the environments is reported in Appendix A. Figure 5.3 shows the results



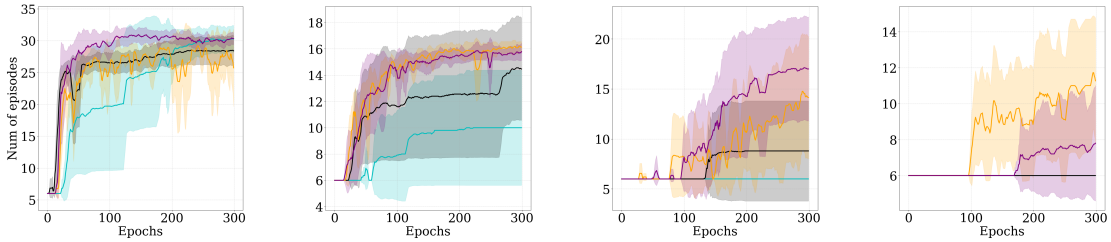
(a) Coverage in LQG. (b) Coverage in Riverswim.

Figure 5.3: Coverage in LQG and Riverswim as function of λ and ρ; average of 5 seeds, 95% c.i..

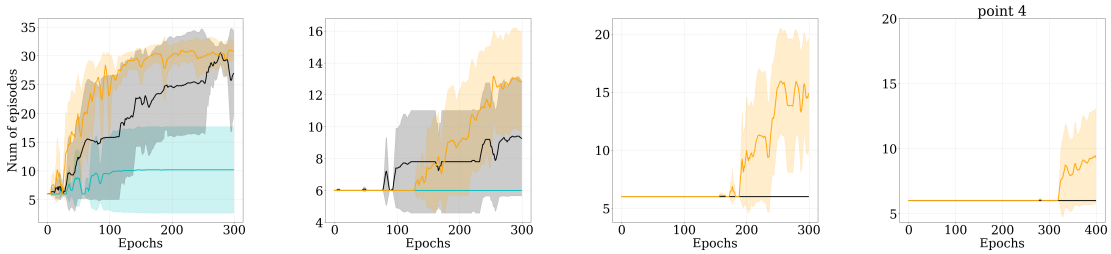
of these experiments. For each environment, we train WAC, varying the parameters λ



(a) Average return in 4 2D navigation tasks.



(b) Number of episodes completed in 3000 steps in 4 2D navigation tasks.



(c) Number of episodes completed in 3000 steps in 4 2D navigation tasks with sparse reward function.

Figure 5.4: Experimental results in 4 2D navigation tasks starting from the easiest (left) to the hardest (right); average of 5 seeds, 95% c.i..

and ρ of the regularized uncertainty loss in Equation (5.7). For each value, we report the coverage averaged over all training epochs. Thus, higher coverage values suggest higher exploration on average. meaning that even when agents reach full coverage at the end of the training, our plots will report lower values as it is averaged over the epochs. Nonetheless, higher values suggest higher exploration. The scale of ρ is measured as a percentage of the batch size used for the Q-value critic gradient estimation.

Firstly, we observe that the coverage is monotonically increasing with both λ and ρ. As expected, low values of ρ cause higher variance, as fewer samples are employed to estimate the uncertainty regularization. This can be seen in all the curves of the leftmost plot, as well as in the third plot, where the black curve corresponding to $\rho = 0.25$ suffers from high variance. In Appendix B, we perform a similar study for OAC and we observe that the coverage is not so easily controllable. We attribute this to the ensemble-based nature of the uncertainty estimation of OAC. While this disagreement between the two critics does give some useful signals to explore due to the variance over different initializations of the critics, it does seem to be enough to allow directed exploration towards unseen regions of the state-action space.

5.6.1 2D Navigation

To assess whether a principled uncertainty estimation and propagation translate into lower sample complexity, we perform an empirical evaluation in a set of MujoCo (Todorov et al., 2012) tasks, where the amount of exploration needed to solve the task can be controlled. We start from the 2D navigation task used in (Moro et al., 2022), where the agent has to reach a goal state in a 2D world, by avoiding obstacles. The reward is the negative Euclidean distance from the goal state. While this is a *dense reward*, the obstacle presence generates local optima which the agent needs to overcome by exploring efficiently. We progressively make the task more challenging by constructing more difficult configurations with the addition of walls and obstacles. In the following, we briefly describe the chosen evaluation tasks.

This environment models a sphere moving inside a two-dimensional maze. The goal of the agent is to get close enough to a goal state on the right side of the maze while avoiding obstacles. Once the agent gets close enough to the goal (Euclidean distance < 2) the episode ends. The state space includes the agent position in the 2-dimensional space, as well as the velocities. The action space is also 2-dimensional, controlling the actuators in both directions. We use 2 reward functions for this task. In the dense reward version of the environment, the reward is the negative Euclidean distance of the sphere from the center of the goal. This represents a dense reward signal, which makes optimization

easier but also introduces local maximums due to the presence of obstacles. In the sparse version, the reward is always -1, so the optimal policy is to reach the goal as quickly as possible so that the episode ends. We devise 4 environment configurations, with different levels of difficulty.

The first environment (Figure 5.5a) has a U-shaped wall in the middle. The agent has to overcome it by either running into it and then moving back to outflank it, so it can escape the local maximum, or preferably, it has to go around it without touching it. The U-shaped of the wall makes the task more difficult since the agent is unlikely to be able to escape the local maximum once it has hit the wall by simply playing random actions.

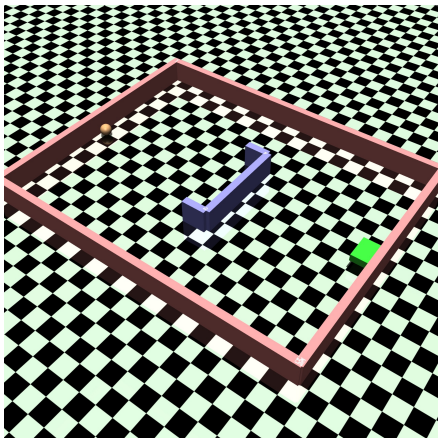
The second environment (Figure 5.5b) has 2 simpler obstacles to overcome compared to the first environment, yet the agent now has to learn how to overcome two obstacles, which overall makes the second task more difficult than the first one.

To overcome the obstacle of the third environment (Figure 5.5c) the agent has to perform a rather complete exploration of the space since the gateway to the second room is quite narrow and it is easy to be stuck in a local maximum at the border due to the reward function based on the Euclidean Distance.

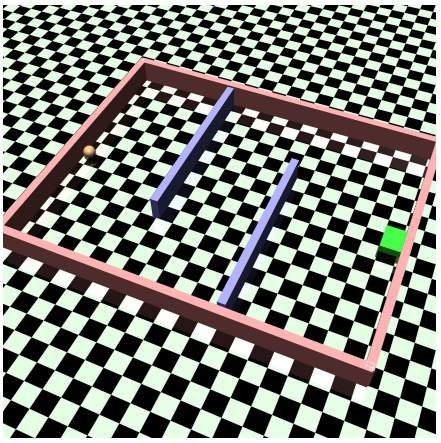
The fourth and last environment, shown in Figure 5.5d, puts together the challenges of the previous two. It has 2 obstacles to overcome, and it also requires a rather complete exploration of the space to advance to the goal.

We name the tasks as "Point x" with $x \in \{1, 2, 3, 4\}$, where a higher x means a more difficult exploration challenge. We compare the performance of WAC, in both versions defined in Section 5.3, with SAC and OAC. In each task, we track the cumulative return, as well as the number of episodes completed in a fixed number of steps (higher is better). We use the implementation of SAC and OAC used in (Ciosek et al., 2019) and extend the repository with our implementation to guarantee comparable results. The same network architectures are used for all algorithms. For the common hyperparameters, we only tune SAC and use the same values for WAC and OAC by additionally tuning the algorithm-specific parameters (δ and β for OAC and λ and ρ for WAC). Details on the hyperparameter tuning are in Appendix B.3.

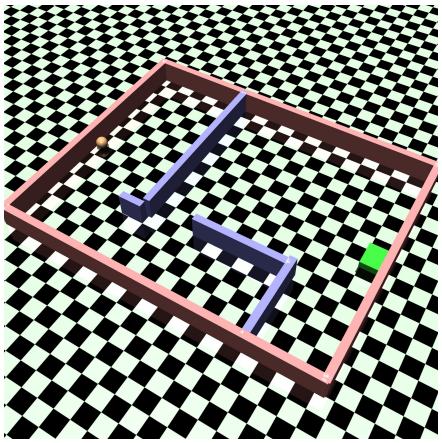
In Figure 5.4a, we present the average return as a function of the training epochs, whereas in Figure 5.4b we present the number of episodes completed in 3000 steps of interaction. Starting from left to right, we increase the difficulty of the task. We can see that for the easiest task, all algorithms are able to find the optimal policy of quickly avoiding an obstacle in the middle to reach the goal state, even though SAC learns slower compared to the others. Being a simple exploration task, ME-WAC performs better and is more stable than OE-WAC. OAC is also able to quickly solve the task. While the



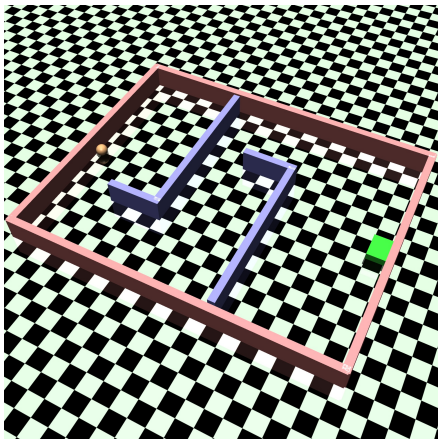
(a) Point 1



(b) Point 2



(c) Point 3



(d) Point 4

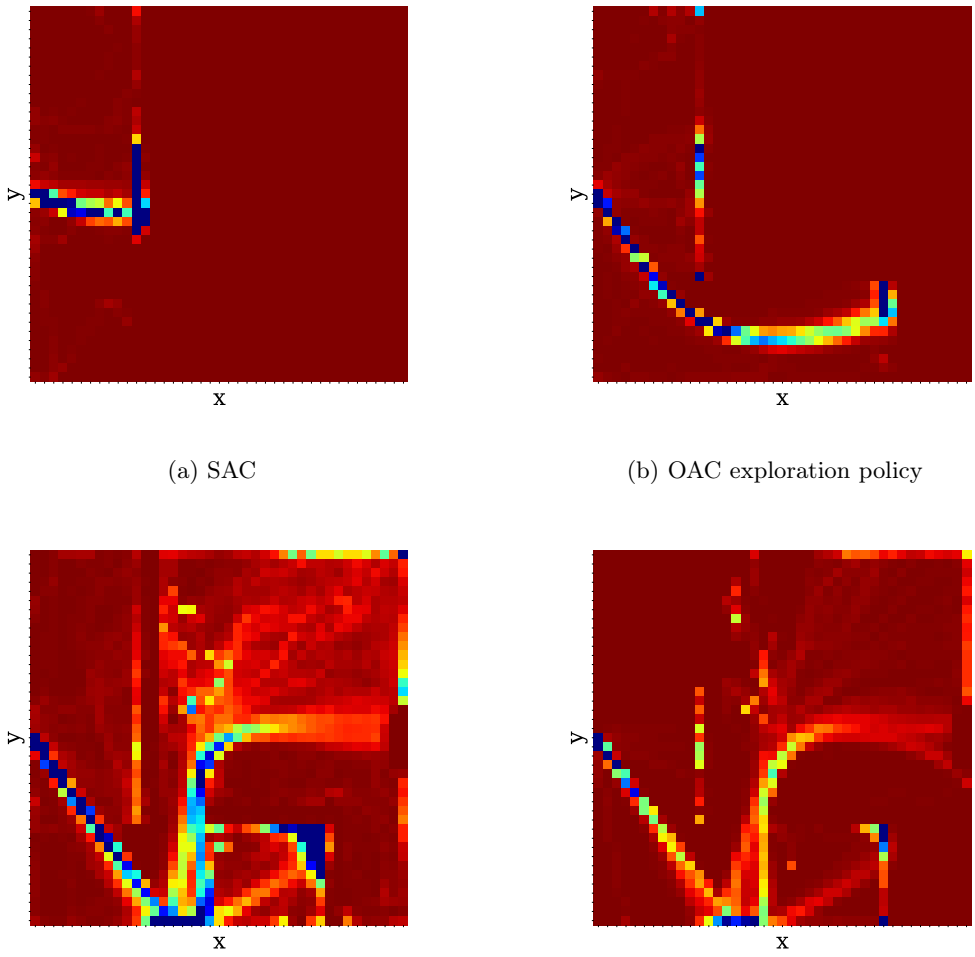
difference in return is negligible, the number of completed episodes shows an advantage for ME-WAC, which completes more episodes faster.

We underline that even though the task does not require particular exploration, WAC does not over-explore, but rather solves with a speed comparable with the other baselines. The clear advantages of WAC in terms of exploration can be seen starting from the second task, where the exploration requirements are increased. Both versions of WAC learn faster and with less variance compared with both SAC and OAC. The difference is even more apparent in the number of episodes completed, where SAC and WAC have disjoint confidence intervals. In the third task, SAC completely fails in learning to reach the goal, while OAC succeeds in some of the seeds only, showing a high variance. WAC, on the other hand, outperforms them both in terms of return and completed episodes. ME-WAC performs better, even though the task requires a good amount of exploration. Compared to ME-WAC, OE-WAC over-explores, and shows a slower learning curve. The last task is solved by the WAC agents only. SAC and OAC never reach the goal state. WAC outperforms them, in both versions, with statistical significance. We also see the need for larger exploration, apparent from the difference in performance between OE-WAC and ME-WAC.

Finally, Figure 5.4c presents the number of episodes completed in a *sparse reward* version of the same tasks. In this scenario, we do not show the return as it is proportional to the number of completed episodes. We only trained OE-WAC agents in these tasks, as they present a substantial exploration challenge. The advantage of WAC is extremely evident in these tasks. SAC and OAC are only able to solve the simplest task. In a sparse reward setting, SAC and OAC will only explore randomly so they fully rely on the chance of reaching the goal state with random actions. OAC explores more compared to SAC, but since the exploration does not depend on the state-action visitations, but only on the disagreement between the critics, sparse reward tasks are a great challenge. WAC, instead, will still explore, even when facing sparse rewards since the uncertainty will gradually decline in visited regions, so the upper bounds will favor reaching unvisited ones. Indeed, OE-WAC outperforms both baselines in all the tasks with sparse rewards.

5.6.2 Exploration Heatmaps

To better appreciate the differences in exploration of the considered algorithms, we show some additional heatmaps which represent the visited states (we have ignored velocities so we could visualize the location of the agent) over 300 epochs of all algorithms we have tested throughout the paper. Figure 5.6 shows the heatmaps from runs on the Point 3 environment with dense rewards. In Figure 5.6a we see that SAC cannot get past the



(a) SAC

(b) OAC exploration policy

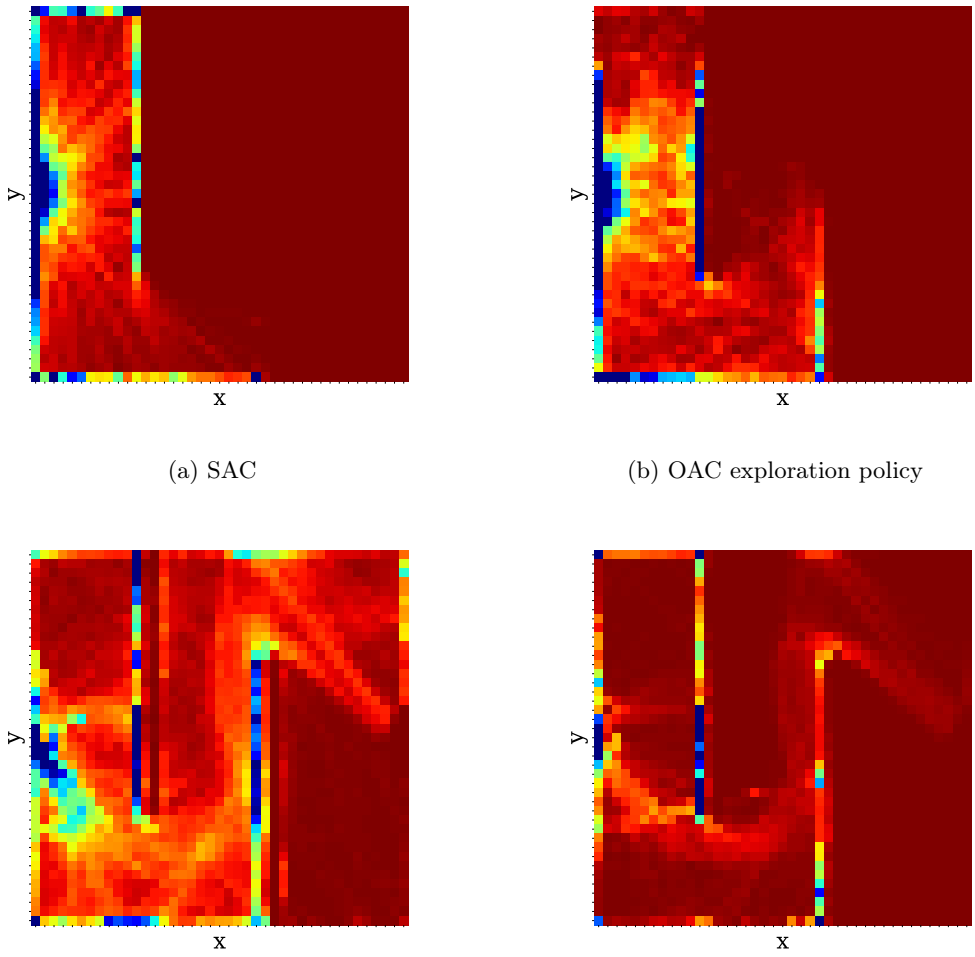
(c) WAC exploration policy

(d) WAC evaluation policy

Figure 5.6: Cumulative visited states in 300 epochs in Point 3 environment (Dense Reward)

first wall and does not explore the space around the local maximum enough to reach other maxima. In Figure 5.6b we see that OAC finds a better maximum but still a local one. WAC does find the same local maximum as OAC, in fact we can see in Figure 5.6c it visits it many times, yet once the uncertainty estimate is low enough it is able to keep exploring and ultimately reach the goal. We have also reported in Figure 5.6d the same heatmap created by the target policy, which follows the critic of the mean instead of the upper bound.

Figure 5.7 shows the cumulative visited states in the Point 2 environment with sparse reward. We can observe that SAC, having no uncertainty estimate and no informative rewards, mostly explores around the starting states with very simple policies that follow straight lines. OAC is able to reach areas of the maze that are further away from the



(a) SAC

(b) OAC exploration policy

(c) WAC exploration policy

(d) WAC target policy

Figure 5.7: Cumulative visited states in 300 epochs in Point 2 environment (Sparse Reward)

starting point, but it still can't reach the goal. Finally, WAC manages to reach the goal of the maze and explores almost every area of it.

5.6.3 Standard MujoCo Experiments

The environments we employed for evaluation are characterized by relevant exploration challenges. This choice, in our view, is motivated by the fact that we propose an approach to effectively address exploration in continuous state-action environments. Nevertheless, for completion, we present here some additional evaluations in the environments of HalfCheetah-v2, Hopper-v2, and Walker2d-v2.

The results are shown in Figure 5.8. We do not perform tuning on the WAC exploration parameters, whereas OAC and SAC are run with the (tuned) parameters

reported in the OAC paper. We are outperformed by OAC in HalfCheetah-v2, outperform both SAC and OAC in Hopper-v2 and perform the same in Walker2d-v2. This supports our claim that standard MujoCo tasks are not suitable benchmarks for exploration.

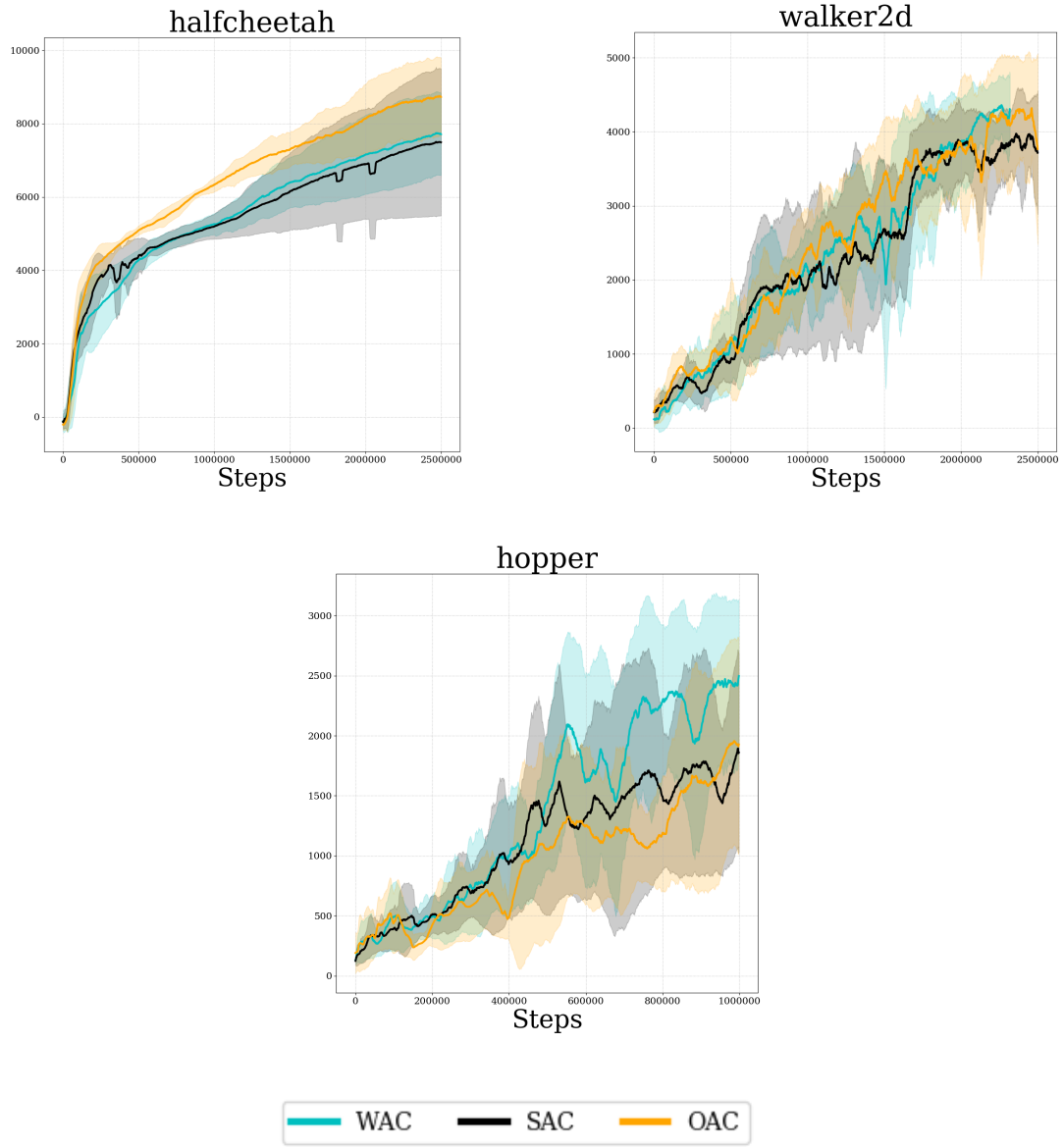


Figure 5.8: Empirical evaluation in 3 standard MujoCo tasks. Average of 4 seeds, 95 % c.i..

5.7 Conclusions

In this chapter, we extended the Wasserstein TD-Learning (WTD) framework to be able to handle continuous action spaces. We presented a novel Actor-Critic algorithm based on the WTD with the goal of performing deep and directed exploration. We presented WAC, which extends the recently proposed WQL to the continuous-actions case. Furthermore, we addressed a problem of uncertainty estimation that arises when using function approximation, related to the generalization of the uncertainty estimates. We proposed a simple yet effective regularization method based on synthetic samples that allowed us to better generalize the uncertainty across the state-action space. Finally, we performed a thorough empirical evaluation to investigate the advantages of performing principled uncertainty estimation and propagation in continuous-action domains. We observed that the uncertainty estimates of WAC can effectively steer exploration towards promising regions of the state-action space, even under sparse rewards, especially when comparing it with heuristic uncertainty estimation based on ensemble methods.

In the next chapter, we will extend the WTD framework to the MCTS setting, similar to the extension we provided in this chapter for the actor-critic setting. In Chapter 3 we observed how AlphaZero struggles when sparse or misleading rewards are optimized and illustrated this issue in the goal-directed setting. While we proposed AlphaZeroHER to handle goal-directed tasks, in the general RL settings the exploration problem still persists. The exploration problem in AlphaZero is mainly related to the tree-search phase, where to avoid exploring the full (huge) search tree, AlphaZero biases the search to regions where the current policy assigns more probability mass. It is clear that this bias of the search phase makes the algorithm exploit more the current policy. Moreover, the PUCT tree policy employed considers each node in the tree as an independent bandit problem, ignoring the temporal dependency between the nodes. By exploiting the WTD framework, we aim to address both these issues. First, the WTD framework allows us to directly generalize uncertainty estimates, and use them to bias the tree search instead of the current policy. This way we perform directed exploration during the tree-search instead of exploitation. Secondly, the WTD framework allows us to propagate the uncertainty estimates through the tree nodes, propagating the uncertainty of deeper nodes up until the root of the tree.

CHAPTER 6

Epistemic Uncertainty in MCTS

6.1 Introduction

In the previous chapters, we have highlighted the remarkable advances the combination of Monte Carlo Tree Search (MCTS) and Reinforcement Learning (RL) has brought during recent years. The ability of MCTS to provide local solutions to sequential decision-making problems, coupled with the ability of RL agents to store and reuse solutions has proved a viable approach to solve many large decision-making tasks. We recall again how the combination of MCTS and RL has demonstrated success in challenging tasks, such as board games, robotics, and complex strategy games such as Go, where DeepMind's AlphaZero utilized an MCTS-based approach coupled with deep neural networks to achieve superhuman performance (Silver et al., 2017a). In all of these domains, MCTS proved effective in taking the partial solutions represented by the parametrized value function and policy and improving them via MCTS, outputting a better policy, and providing, in this way, the policy improvement step. After this policy improvement via MCTS, AlphaZero performs a simple policy evaluation step by estimating the new policy performance via the Monte Carlo (MC) estimator introduced in Chapter 2. In this way, we can interpret the AlphaZero algorithm as an iterative process of policy improvement

and policy evaluation, just as many of the algorithms we have discussed in this thesis. While the policy evaluation step is straightforward in this case, limiting itself to the standard unbiased MC estimator, the policy improvement step is crucial for achieving good performance in practice.

Despite these successes, challenges persist in the integration of MCTS and RL. One significant challenge is the computational cost associated with extensive tree exploration, particularly in domains with large state and action spaces. Scalability remains an issue, limiting the application of MCTS-RL in real-time, resource-constrained environments. Additionally, striking the right balance between exploration and exploitation is an ongoing challenge, especially when the complexity of the environment increases. In scenarios where there is no opponent to compete against, such as navigating complex environments or solving puzzles, MCTS faces challenges related to the need for prolonged and deep exploration. Single-agent tasks often involve vast state spaces and intricate decision landscapes, making it challenging for MCTS to efficiently discover optimal solutions. While the computational cost and scalability issues remain a barrier to the application of these methods, we will only focus in this chapter on improving the exploration capabilities of MCTS.

In Chapter 3, we highlighted how AlphaZero still struggles with exploration-heavy tasks, like the goal-directed tasks we investigated. While we proposed a simple solution via the application of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), this is only achievable when we have access to the reward function and when this reward function accepts a goal-dependent definition. When the decision-making task cannot be mapped in a goal-dependent setting, and when the reward function is sparse or even dense but misleading, the policy improvement step of AlphaZero will struggle to improve the current policy, as demonstrated empirically in this thesis.

The struggle of MCTS in single-agent tasks highlights the algorithm's dependence on competitive interactions for effective exploration. While it excels in exploiting adversarial relationships to improve decision-making, its performance diminishes when the goal is to explore an environment without a clear adversary. Developing effective strategies for balancing exploration and exploitation in the absence of external competition remains a key research area to enhance MCTS's applicability in diverse sequential decision-making contexts.

In the course of this thesis, we discussed how efficient and deep exploration is not a problem only for MCTS, but for RL agents too. Indeed, the ability to effectively decide when to exploit the current information collected or to explore with the hope of finding better paths is crucial for developing intelligent and efficient agents. In its basic form, UCT (and its variants like PUCT used in AlphaZero), simplifies the exploration of

the search tree by considering it an independent sequence of bandit problems. While this greatly simplifies the algorithm and its theoretical study, it comes with significant effects on its performance in hard exploration problems, where UCT has been shown to display arbitrarily slow convergence (Munos, 2014). Many attempts have been made to improve the tree-exploration of MCTS by incorporating techniques developed in the RL literature (Browne et al., 2012). These methods go from simple extensions of the bandit exploration methods to incorporate and propagate information about the variance of the updates in UCB-V (Lieck et al., 2017), to more sophisticated methods that employ Bayesian frameworks to estimate the complete return distribution instead of just its expectation (Bai et al., 2018). Indeed, given the ability of MCTS to locally discretize the state-action space representing it with the search tree, most of the RL exploration techniques developed for the tabular setting can be adapted to be incorporated in MCTS. For a more extended discussion on the combination of MCTS and RL, we refer the reader to (Vodopivec et al., 2017).

While combining MCTS with effective exploration techniques has been extensively studied in the literature, the combination with function approximators such as neural networks presents additional challenges. In this chapter, we will explore the combination of MCTS with the uncertainty estimation techniques presented in the previous chapters. The crucial characteristic of AlphaZero is the ability to bias the search tree exploration by employing the parametrized policy and value function. Extending this to the use of uncertainty estimates to provide better exploration is not straightforward. Wasserstein TD-Learning (WTD), on the other hand, provides intrinsically a framework to generalize uncertainty estimates, which gave promising results even in high dimensional continuous state-action spaces. This ability of WTD to generalize uncertainty estimates presents an opportunity to combine it with MCTS in an AlphaZero fashion. In this way, we avoid using the current policy alone for biasing the search, and can instead use estimates of how much the agent is unsure of its performance.

In the next sections, we start with a brief discussion of the literature on the combination of Reinforcement Learning and planning, with a focus on efficient exploration. We continue with the presentation of our proposed algorithm, Wasserstein Azero, which employs the (epistemic) distributional value functions presented in Chapter 3 and Chapter 4 to derive upper bounds on the value functions and discuss how MCTS can be employed to refine these upper bounds. We conclude the chapter with an experimental campaign and a final discussion.

6.2 Related Literature

In this section, we provide a brief overview of the literature tackling the combination of Reinforcement Learning techniques with online planning. As discussed earlier, the basic idea of MCTS is to iteratively build a search tree of the future by exploiting a forward model of the environment with the ultimate goal of identifying and evaluating the optimal action in the current state, i.e., the root node of the tree. The key aspect of MCTS that led to improved performance in a large number of environments and tasks was the ability to build a highly *assymetric* search tree by focusing on more *promising* paths of the search tree. It is clear at this point that the definition of *promising* greatly affects the performance of the tree search, and generally, the best method to employ is highly problem-dependent. Different choices of the *selection policy* and *backprogation* in the 4 phases presented in Chapter 2, greatly affect the resulting search-trees and final performance.

The first and most popular algorithm building incrementally an asymmetric search tree in this form, UCT (Kocsis and Szepesvári, 2006) employs independent instances of UCB1 (Auer et al., 2002) in each node of the tree, ignoring the sequentiality and non-stationarity of the return processes observed in the tree. Moreover, UCT, in its basic form, employs simple MC estimates during backpropagation. While this has proven to be enough in many planning tasks and endows the algorithm with asymptotical convergence guarantees, in general, it can make the algorithm converge arbitrarily slow (Munos, 2014). A possible direction to improve the search performance of basic UCT is to employ more sophisticated bandit algorithms while traversing the tree. Indeed, a large body of research explores this direction (Browne et al., 2012). UCT-V(Lieck et al., 2017) combines MCTS with the UCB-V algorithm, exploiting variance information of the returns observed in the nodes of the search tree. The authors show that the algorithm provides the same convergence guarantees of UCT while significantly outperforming it empirically in the considered tasks. UCT-V also moved away from standard Monte Carlo backups, by providing Dynamic Programming (DP) updates also for the variance information in the tree nodes. The combination of DP with MCTS has been also studied in (Vodopivec et al., 2017), where the authors unify the literature and terminology of MCTS and RL, and provide a general algorithm, SARSA-UCT(λ), that combines the TD(λ) algorithm, with MCTS, providing the same convergence guarantees, coupled with improved performance in select tasks. More sophisticated backup operators have also been employed in the literature, including the generalized mean backup operators in (Dam et al., 2019), where the authors preserve the convergence guarantees of UCT while providing value estimates that are more robust to overestimation and underestimation.

Bayesian RL has also been successfully combined with MCTS in recent years. In (Tesauro et al., 2012) the authors extend UCT by modeling the value estimates as a Gaussian distribution, similar to WTD, to represent the uncertainty of the estimated values. Similarly to WTD, they propose a fast uncertainty propagation method in the nodes of the tree. In (Bai et al., 2018), the authors take a distributional perspective by extending Bayesian Q-Learning (Dearden et al., 1998) to the MCTS setting, proposing DNG-MCTS. DNG-MCTS represents the return distribution in each node as a Dirichlet-NormalGamma distribution, which allows for fast Bayesian updates with each new sample. Unfortunately, this does not allow showing the convergence of the algorithm, as it ignores the dependency of the samples during these updates, but nonetheless, it shows improved performance in the considered environments. Finally, in (Mern et al., 2021) the authors propose the use of Gaussian processes to model the action-value function in the search tree and select actions that maximize an upper bound of the action-value distribution.

6.3 Wasserstein AlphaZero

In this section, we present the Wassertein AlphaZero (WAZ) algorithm, which employs the Wassertein TD-Learning framework in the context of online planning. Effectively exploring the search tree during MCTS is crucial for improving the sample complexity of MCTS algorithms in environments where deep exploration is required. AlphaZero tackles this problem by biasing the search toward actions favored by the current policy.

More formally, in each node $\mathcal{N}$ representing state $s_{\mathcal{N}}$ of the search tree $\mathcal{T}$, when visiting the node for the n-th time, AlphaZero employs a modified PUCT (Rosin, 2011) selection policy as follows:

$$\begin{aligned} a_n &= \arg\max_{i=1..K} \overline{Q}(s_{\mathcal{N}}, a_i, T_i(n-1)) + B(a_i) \\ &= \overline{Q}(s_{\mathcal{N}}, a_i, T_i(n-1)) + C\pi_\theta(s_{\mathcal{N}}, a_i)\sqrt{\frac{n}{1+T_i(n-1)}}, \end{aligned}$$

where a is the selected action, $B(a_i)$ is the exploration bonus, $\overline{Q}(s_{\mathcal{N}}, a_i, T_i(n-1))$ is the current estimate of the action-value function of the i-th action, $T_i(n-1)$ is the number of visits of the i-th action in node $\mathcal{N}$ and C is an exploration constant. The original PUCT selection policy does not perform the scaling with π_θ in the exploration bonus $B(a_i)$. This means that the tree-search is unbiased, starting with equal confidence intervals across the state-action space. AlphaZero, on the other hand, scales the exploration bonuses directly with the probability that the current policy π_θ assigns to each action. This will favor actions that are already favored by the policy, adding more *exploitation* during the tree-building phase. In this work, we argue that this could be detrimental in certain scenarios where exploration is crucial. If the parametrized policy π_θ becomes deterministic during learning, it is straightforward to show that only one action will be explored, and if this action is not the optimal one, the tree search will not be able to discover that said action is suboptimal. Indeed, in the original AlphaZero paper, the authors argue for additional exploration in the root state s_0 by adding some noise to the policy $\pi_\theta(s_0)$. In the following, we present a more principled approach for biasing the tree search, by exploiting the distributional critics of the WTD framework.

6.3.1 Wassertein Monte Carlo Tree Search

The Wasserstein TD-learning framework, presented in Chapter 4 and Chapter 5 allowed us to propagate uncertainty estimates across the state-action space and generalize them through the use of parametrized function approximators, such as neural networks. In this section, we discuss how to use these parametrized distributional critics during tree-search.

In Wasserstein AlphaZero, as in Wasserstein DQN and Wasserstein Actor-Critic, we

maintain a parametrized distributional critic $\mathcal{Q}_{\boldsymbol{\omega}}$, representing the distribution of possible action-value functions for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, i.e., $Q(s, a) \sim \mathcal{Q}_{\boldsymbol{\omega}}(s, a)$. The variance of $\mathcal{Q}_{\boldsymbol{\omega}}(s, a)$ represents the uncertainty around the best current estimate $\mathbb{E}[\mathcal{Q}_{\boldsymbol{\omega}}(s, a)]$. As in the previous chapters, we have the choice of the distribution class to employ. We will focus in this chapter too, on the Gaussian case, i.e., $Q(s, a) \sim \mathcal{N}(\mu_{\boldsymbol{\omega}}(s, a), \sigma_{\boldsymbol{\omega}}(s, a))$. All the details are easily extensible to other distribution classes, such as the particle distributions discussed in Chapter 4. Compared to the MCTS procedure employed in AlphaZero, Wasserstein Monte Carlo Tree Search differs mainly in the selection and backpropagation phase.

Wasserstein Tree Selection

The goal of our MCTS procedure is to take in input the Q-distributions represented by our distributional critic, which, as in Chapter 4, can be used to derive upper bounds on the state-action value, and refine them through the search to recover tighter upper bounds.

For the selection phase, we now have available at each node of the tree, the whole Q-posterior $\mathcal{Q}_{\boldsymbol{\omega}}(s, a)$ for each action. We can, therefore, apply all the policies described in Section 4.3.4. It is clear the best choice of policy is problem-dependent. We are interested in refining the upper bounds:

$$U^{\delta}_{\boldsymbol{\omega}}(s, a) = \mu_{\boldsymbol{\omega}}(s, a) + \sigma_{\boldsymbol{\omega}}(s, a)\Phi^{-1}(\delta),$$

where $\delta \in (0, 1)$ is the confidence level of the upper bound and Φ^{-1} is the quantile function of the standard normal distribution. Hence we will only focus our attention on the optimistic policies. More formally, when each node in the tree is created, representing state s of the environment, we query the distributional critic to get an initial estimate of the Q-posterior, getting an initial estimate of the mean and standard deviation for each action:

$$\mu_0 = \mu_{\boldsymbol{\omega}}(s, a), \quad \sigma_0 = \sigma_{\boldsymbol{\omega}}(s, a)) \quad \forall a \in \mathcal{A}.$$

Consequently, during the selection phase, when passing through node $\mathcal{N}$ for the n-th time, we apply the optimistic selection policy:

$$\begin{aligned} a_n &= \arg\max_{i=1..K} U^{\delta}_n(s_{\mathcal{N}}, a_i) \\ &= \mu_{n-1}(s_{\mathcal{N}}, a_i) + \sigma_{n-1}(s_{\mathcal{N}}, a_i)\Phi^{-1}(\delta), \end{aligned} \tag{6.1}$$

where $\mu_{n-1}(s_{\mathcal{N}}, a_i)$ is the current estimate of the mean of the Q-posterior for the i-th action after $n-1$ passes and $\sigma_{n-1}(s_{\mathcal{N}}, a_i)$ is the current estimate for the standard deviation of the Q-posterior for the same action after $n-1$ passes. Note that, same

as AlphaZero, we only query the network for an initial estimate μ_0 and σ_0. Afterward, we store these values in each action node of the tree and update them during the backpropagation phase of the search. Other policies, like the mean policy that selects according to the mean of the Q-posteriors $\mu_{n-1}(s, a)$ or the posterior sampling policy, that samples actions according to the probability of being optimal are available but we restrict our focus on the optimistic policy only.

Wasserstein Tree Backpropagation

In this section, we describe the backup operators employed in Wasserstein MCTS. Following the classical MCTS scheme, after traversing the tree with the optimistic selection policy and reaching a leaf node, we expand this node getting an initial estimate of the Q-posterior for each action from the distributional critic $\mathcal{Q}_{\boldsymbol{\omega}}$ resulting in the initial estimates μ_0 and σ_0 for each action $a \in \mathcal{A}$. Similar to AlphaZero, we avoid an expensive (in terms of samples from the model) evaluation of the last node, by performing a bootstrap with these initial estimates. This section is analogous to Section 4.3.3, where we need to define the V-posteriors of the next states, starting from the Q-posteriors of the newly added node.

We will present three choices for the backup operator to employ. We are interested in performing these updates after each pass of the tree during the search. Specifically, by traversing the tree with the selection policy described earlier, we visit a series of $m-1$ nodes, $\mathcal{N}_i$ for $i = 1..m$. This defines a path of length m along the tree $\tau = (s_0, a_0, r_0..., s_{k-1}, a_{m-1}, r_{m-1}, s_m)$ where s_m is the state of the newly expanded node for which we also obtain the initial Q-posterior estimates μ_{0,a_i}, σ_{0,a_i} for each action.

We start by extending the Optimistic Estimator (OE) introduced in Section 4.3.3 to MCTS. We recall that this operator applies the optimistic policy in Equation 6.1 to define the V-posteriors starting from the Q-posteriors of each action. More formally, given the $(t+1)$-th node $\mathcal{N}_{t+1}$ along path τ, and current Q-posteriors $\mathcal{Q}(\mathcal{N}_{t+1}, a_i)$ in the node, we define the V-posterior of $\mathcal{N}_{t+1}$ according to Proposition 4.3.1 as:

$$\begin{aligned} a &= \arg\max_{i=1..K} U^{\delta}(\mathcal{N}_{t+1}, a_i), \\ \mathcal{V}(\mathcal{N}_{t+1}) &= \mathcal{Q}(\mathcal{N}_{t+1}, a). \end{aligned} \tag{6.2}$$

We can now propagate in the Q-posterior of the t-th node of the path, the V-posterior,

performing the following update:

$$\begin{aligned}\mu_n(\mathcal{N}_t, a_t) &\leftarrow (1-\alpha)\mu_{n-1}(\mathcal{N}_t, a_t) + \alpha\left[r_t + \gamma\mu(\mathcal{N}_{t+1})\right], \\ \sigma_n(\mathcal{N}_t, a_t) &\leftarrow (1-\alpha)\sigma_{n-1}(\mathcal{N}_t, a_t) + \alpha\gamma\sigma(\mathcal{N}_{t+1}), \\ \alpha &= \frac{1}{T(n-1)},\end{aligned} \tag{6.3}$$

where a_t is the action taken at node $\mathcal{N}_t$ in path τ, r_t is the reward observed after taking that action, $\mu(\mathcal{N}_{t+1})$ and $\sigma(\mathcal{N}_{t+1})$ are the mean and standard deviation of the V-posterior defined in Equation 6.2 and $T(n-1)$ is the number of times action a_t was visited in node $\mathcal{N}_t$ up to this time. This backup operator is then applied recursively along the path τ starting from the newly added leaf node. We recall again here, that this operator shrinks the uncertainty of the posteriors after each update with the discount factor γ. We denote this as the OE backup operator.

It is straightforward to observe that the OE operator, being optimistic, might cause the uncertainty to shrink too slowly, resulting in overexploration. This is because, in every node visited along path τ, the operator applies the optimistic policy to define the next state V-posterior, meaning that the posterior of the newly added node, might not be propagated up to the root, but only until the corresponding action along the path is the one with the highest upper bound. For this reason, and also to have a clearer comparison with the base algorithm AlphaZero, we employ a $TD(k)$ backup, where k depends on the depth of the newly added leaf node. More formally, given a path of length m along the tree $\tau = (s_0, a_0, r_0 ..., s_{k-1}, a_{m-1}, r_{m-1}, s_m)$, and initial Q-posterior estimates μ_{0,a_i}, σ_{0,a_i} for each action in the newly added node $\mathcal{N}_m$, we define the value of said node as:

$$\begin{aligned}a_m &= \arg\max_{i=1..K} U_0^{\delta}(s_{\mathcal{N}_m}, a_i), \\ \mathcal{V}(\mathcal{N}_m) &= \mathcal{Q}(\mathcal{N}_m, a_m),\end{aligned} \tag{6.4}$$

where $\mathcal{Q}(\mathcal{N}_m, a_m)$ denotes the Q-posterior of action a_m in node $\mathcal{N}_m$ and we apply again the optimistic policy to estimate the value of the new state $s_{\mathcal{N}_m}$. $\mathcal{V}(\mathcal{N}_m)$ is then propagated along path τ until reaching the roof, performing a $TD(k)$ update with variable k. Specifically, at the t-th node $\mathcal{N}_t$ of path τ, where we have taken action a_t in this path, which has been visited $T(n-1)$ times so far, the update takes the form:

$$\begin{aligned}\mu_n(\mathcal{N}_t, a_t) &\leftarrow (1-\alpha)\mu_{n-1}(\mathcal{N}_t, a_t) + \alpha\left[\sum_{j=t}^{j<m} \gamma^{j-t} r_{t+j} + \mu_0(\mathcal{N}_m)\right], \\ \sigma_n(\mathcal{N}_t, a_i) &\leftarrow (1-\alpha)\sigma_{n-1}(\mathcal{N}_t, a_i) + \alpha\gamma^{m-t}\sigma_0(\mathcal{N}_m), \\ \alpha &= \frac{1}{T(n-1)},\end{aligned} \tag{6.5}$$

where $\mu_0(\mathcal{N}_m)$ and $\sigma_0(\mathcal{N}_m)$ are the mean and standard deviation of the V-posterior defined in Equation 6.4. This backup operator is the closest to the PUCT update scheme employed in AlphaZero so we include it for a better comparison with AlphaZero. Its main characteristic is that it tends to reduce the uncertainty estimates faster during the search phase, being that the update decreases the uncertainty of the V-posterior of the leaf node exponentially w.r.t. the depth of the leaf node before incorporating it in the nodes visited during the backup and this backup is performed recursively until reaching the root node. We denote this as the TD backup operator.

Finally, we present a third backup operator that combines the characteristics of the OE and TD backup operators, providing an uncertainty that shrinks at a slower rate than the TD operator, while still incorporating the newly collected sample in all updates that reach the root node. For this operator, we move away from the optimistic policy when defining the V-posterior, using instead the counts of every action in the node to define the next state policy. More formally, given the $t+1$-th node $\mathcal{N}_{t+1}$ along path τ, and current Q-posteriors $\mathcal{Q}(\mathcal{N}_{t+1}, a_i)$ in the node, we define the V-posterior of $\mathcal{N}_{t+1}$ as:

$$\begin{aligned} \mathcal{V}(\mathcal{N}_{t+1}) &= \mathop{\mathbb{E}}_{a\sim\overline{\pi}(\cdot)}\left[\mathcal{Q}(\mathcal{N}_{t+1}, a)\right], \\ \overline{\pi}(a) &= \frac{T(n-1)(a)}{\sum_{a'\in\mathcal{A}} T(n-1)(a')}, \end{aligned} \tag{6.6}$$

where $T(n-1)(a)$ is the number of times we have tried action a in node $\mathcal{N}_{t+1}$. After defining the V-posterior of node $\mathcal{N}_{t+1}$ according to Equation 6.6 we perform the backup up the tree similar to Equation 6.3 where the only change is in the definition of the V-posterior of the next state. In this case, the V-posterior of the newly added node will always impact the updates that reach the root node of the tree, albeit with a smaller weight according to the policy derived from the action counts in each node $\overline{\pi}$. We note that the policy $\overline{\pi}$ still tends to favor actions with higher upper bounds, since the selection policy employed during the tree traversal is optimistic, and it is the selection policy that generates the action counts. We denote this operator as the Wasserstein Operator (WO).

In (Dam et al., 2023), the authors propose a similar combination of Wasserstein TD-Learning with Monte Carlo Tree Search for tackling highly stochastic environments and POMDPs. The Wasserstein MCTS algorithm is similar to ours with the main difference being in the Wasserstein barycenter scheme used to propagate the uncertainty estimates. In this work, the authors employ a combination of $\mathbb{L}^1$-Wasserstein barycenters and α-divergences while updating the Q-posteriors, drawing a connection with the generalized mean estimators for MCTS introduced in (Dam et al., 2019). Moreover, they provide a proof of polynomial convergence to the optimal policy at the root of the tree when employing a posterior sampling strategy for sampling from the posteriors. Finally,

they employ a thorough empirical evaluation in several simulated domains showing the empirical benefits of the combination of WTD with MCTS. In this work, the authors focus on pure MCTS, without the use of function approximators to bias the tree search. The main difference of the approach described in this setting is the use of function approximators to initialize the Q-posteriors in the nodes. In the next section, we discuss how to train the parametrized Q-posteriors after the W-MCTS phase.

6.3.2 Training the Wasserstein Critic

In the previous section, we described how Wasserstein MCTS can be used to refine the Q-posteriors given by our distributional critics $\mathcal{Q}_{\boldsymbol{\omega}}$ when a forward model of the environment is available. In the current section, we will present the main loop of the Wasserstein MCTS algorithm.

Similar to AlphaZero, at each time step t, starting from the current environment state s_t, we perform M search iterations using the general MCTS scheme of selecting a leaf node to expand by traversing the tree with the optimistic policy, expanding and evaluating the new node with the distributional critic $\mathcal{Q}_{\boldsymbol{\omega}}$, and propagating the V-posteriors to the root node. Depending on the choice of backup operator, we recover three versions of the algorithm, W-MCTS-OE, W-MCTS-TD, and W-MCTS-WO, for the OE, TD, and WO backup operators, respectively.

The MCTS procedure provides us with new target posteriors for each of the actions available in the current state s_t, $\overline{\mathcal{Q}}(s_t, a), \forall a \in \mathcal{A}$. At this point, we store the Q-posterior targets and select an action to play at the current state. We can play an action optimistically, maximizing the upper bounds of the posteriors $\overline{\mathcal{Q}}$, or for a more conservative play we can still sample actions according to the counts at the root node like AlphaZero, following Equation 2.26. We execute the selected action a_t and observe the next state s_{t+1}. After each interaction with the true environment, we store in the replay buffer $\mathbb{D}$ one tuple of the form $(s_t, a, \overline{\mu}(s_t, a), \overline{\sigma}(s_t, a)$ for each available action $a \in \mathcal{A}$, where $\overline{\mu}$ and $\overline{\sigma}$ denote the mean and standard deviation of the Q-posteriors derived via MCTS. Consequently, we repeat the same procedure in the next state until the end of the episode.

To maintain the comparison with AlphaZero, we repeat this for a number of episodes and then train the parametrized value functions. Specifically, we employ a Mean Squared Error (MSE) loss for both the mean and uncertainty networks. More formally, Wasserstein AlphaZero minimizes the following loss:

$$\mathop{\mathbb{E}}_{(s,a,\mu,\sigma)\sim\mathbb{D}} \left[(\mu_{\boldsymbol{\omega}}(s,a) - \mu)^2 + (\sigma_{\boldsymbol{\omega}}(s,a) - \sigma)^2 \right]. \tag{6.7}$$

Like in Chapter 5, we employ separate approximators to estimate the mean and standard deviation of the posteriors. Moreover, we also regularize the uncertainty network according to Equation 5.7. The pseudocode of Wasserstein AlphaZero is presented in Algorithm 8.

The main difference in the training procedure employed by Wasserstein AlphaZero, compared to base AlphaZero, is the value function targets. In AlphaZero, the authors employ Monte Carlo estimates of the return of the played policy whereas we employ Temporal Difference updates, where we use the output of our critic inside the search tree

Algorithm 8 Wasserstein AlphaZero

Initialize memory buffer $\mathbb{D}$
Initialize distributional critic networks $\mu_{\boldsymbol{\omega}}, \sigma_{\boldsymbol{\omega}}$
for epoch $= 1, \cdots, N$ **do**
  for episode $= 1, \cdots, M$ **do**
    Sample initial state $s_0 \sim \mu$
    while not done **do**
      $\{\overline{\mu}(s_t, a_i)\}_{a_i \in \mathcal{A}}, \{\overline{\sigma}(s_t, a_i)\}_{a_i \in \mathcal{A}}, a_t \leftarrow$ W-MCTS$(s_t, \mu_{\boldsymbol{\omega}}, \sigma_{\boldsymbol{\omega}})$
      $s_{t+1}, r_t,$ done $\leftarrow$ applyAction(a_t)
      $\mathbb{D} \leftarrow \mathbb{D} \bigcup \{(s_t, a_i, \overline{\mu}(s_t, a_i), \overline{\sigma}(s_t, a_i)\}_{a_i \mathcal{A}}$
      $s_t \leftarrow s_{t+1}$
    end while
    update $\mu_{\boldsymbol{\omega}}, \sigma_{\boldsymbol{\omega}}$ according to Equation 6.7
  end for
end for

to generate the value function targets. In the next section, we perform an experimental campaign to evaluate whether this learning procedure provides performance benefits empirically.

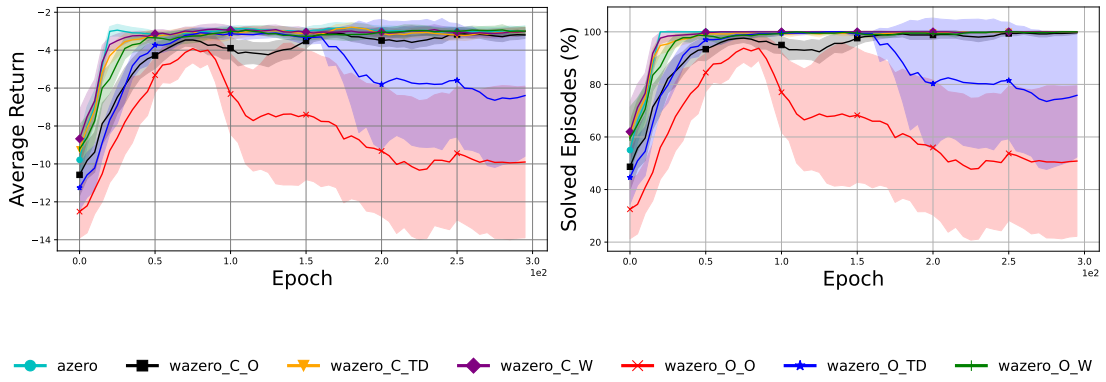


Figure 6.1: Performance of AlphaZero and Wasserstein AlphaZero in a bitflip environment with length 10 using 30 search iterations. Average over 5 runs, 95% c.i.

6.4 Experimental Results

In this section, we perform an experimental campaign of the Wasserstein AlphaZero algorithm, with the goal of observing if the benefits of the WTD framework introduced in Chapter 4 can also be extended to the MCTS setting. For this reason, we perform an evaluation in the simulated domains introduced in Chapter 3, where we saw AlphaZero struggling to solve most of the hard exploration tasks. We test 6 versions of Wasserstein AlphaZero, representing all the combinations of behavioral policy in the true environment and backup operator during MCTS with the forward model. For the behavioral policy, we consider two options, a policy based on the counts at the root of the tree like AlphaZero (suffix C) or based on the optimistic policy derived from the posteriors at the root (suffix O). For the backup operator, we consider all three operators introduced in the previous section, Optimistic operator (suffix O), TD operator (suffix TD) or Wasserstein operator (suffix W).

6.4.1 Bitflip Environment

We start with an evaluation in the Bitflip simulated environment introduced in Chapter 3. The goal of this experiment is to evaluate the exploration capabilities of Wasserstein AlphaZero in a simple simulated environment where we can easily control the difficulty of the learning task, with the goal of observing the algorithm's performance in both easy and hard environments. We reuse the same hyperparameters used in the experiments of Chapter 3 for both AlphaZero and Wasserstein AlphaZero. For the Wasserstein AlphaZero we have 1 extra hyperparameter, the initialization of the Q-posteriors which we do not tune due to computational constraints. In all the experiments, we used an initial Q-posterior with a mean of 0 and variance of 1.

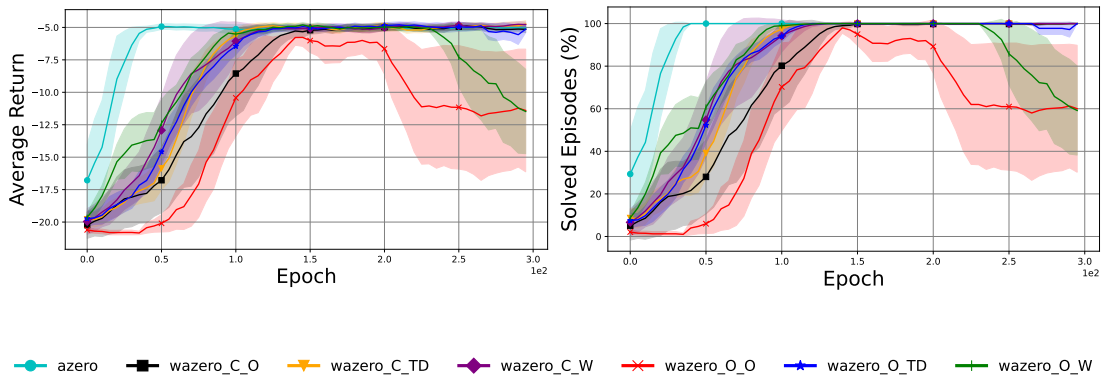


Figure 6.2: Performance of AlphaZero and Wasserstein AlphaZero in a bitflip environment with length 14 using 30 search iterations. Average over 5 runs, 95% c.i.

We start by considering easy cases of bit strings of lengths 10 and 14 where we observed that AlphaZero could learn the tasks easily. In Figure 6.1 we show the results of AlphaZero and Wasserstein AlphaZero in a bitflipt environment of length 10. We present the agent's return and percentage of solved episodes as a function of training epochs. Each training epoch consists of the collection of 50 episodes of the middle loop in line 4 in Algorithm 8.

We observe that as expected, AlphaZero solves the environment easily. When it comes to Wasserstein AlphaZero, we observe that most configurations of the algorithm version have no problem with solving the environment, albeit expectedly slower than AlphaZero. We attribute this to overexploration since the 10-bit environment does not need much exploration. The important takeaway from this environment is the fact that the versions of AlphaZero employing an explorative behavioral policy exhibit much instability in training. Indeed wazero_O_TD and wazero_O_O while they start by learning quickly in all runs, show instability when some of the runs start overexploring decreasing average performance and increasing the variance in the second part of learning. In Figure 6.2, we present the results of the same experiment in a bitflip environment of length 14. In this environment, the overexploration issue of Wasserstein AlphaZero is even more evident, with most of the configurations recovering and maintaining the optimal policy but substantially slower compared to base AlphaZero. Moreover, when using optimistic behavioral policies, we confirm the learning instabilities observed in the previous environment.

Next, we lengthen the bit sequence to a value of 18, when we first started observing exploration issues in AlphaZero in Chapter 3. In Figure 6.3, we present the results of this experiment. In the case of AlphaZero, we confirm the previous results showing that the algorithm struggles to solve this environment, with most of the learning runs not

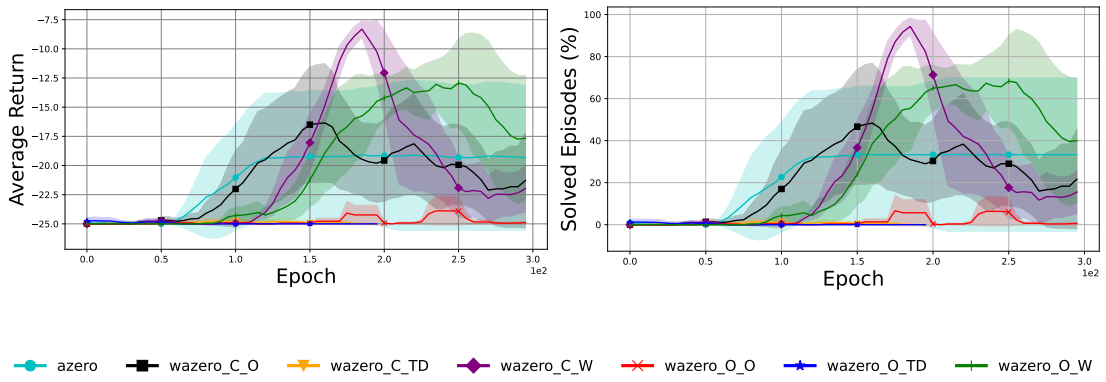


Figure 6.3: Performance of AlphaZero and Wasserstein AlphaZero in a bitflip environment with length 18 using 30 search iterations. Average over 5 runs, 95% c.i.

solving the environment and converging to a suboptimal policy resulting in a low average performance with high variance. Wasserstein AlphaZero on the other hand, confirms the high instability of its training observed in the previous environments. While in half of the versions of the algorithm, we manage to outperform AlphaZero, recovering policies with better average performance and with less variance between runs, the algorithm struggles to maintain these good policies and overexplores. This overexploration is observed in all versions of the algorithm, both in the case of conservative and explorative behavioral policies. When extending the bitflip environment further to longer bit sequences, none of the algorithms managed to solve the tasks so we omit the presentation of these results.

To conclude this section, we observed that in the bitflip environments of varying lengths, the Wasserstein AlphaZero algorithm was prone to overexploration. While in simple environments, when used with a conservative exploration policy derived from the counts at the root, this overexploration did not bring instabilities, in harder environments the algorithm showed substantial instabilities. Nevertheless, Wasserstein AlphaZero does indeed show more directed exploration capabilities compared to base AlphaZero, outperforming the latter often, but struggling to maintain high performance. In the next section, we investigate the 2D navigation environment introduced in Chapter 3 to observe if these properties of the algorithm extend to other environments.

6.4.2 2D Navigation

In the previous section, we observed how the Wasserstein AlphaZero algorithm displayed instabilities in learning attributable to overexploration in both easy environments and harder explorative tasks. In this section, we investigate if this behavior is observable also in navigation tasks like the point environment used in Chapter 3 and Chapter 5. We

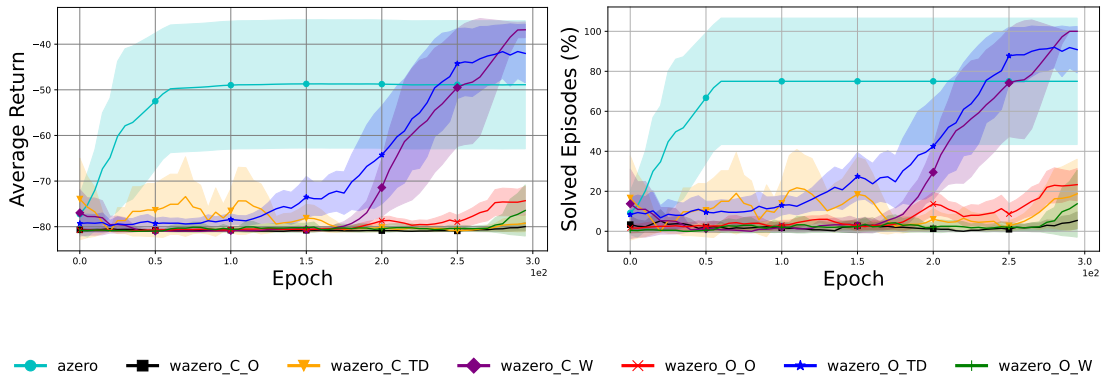


Figure 6.4: Performance of AlphaZero and Wasserstein AlphaZero in the easier navigation environment using 70 search iterations. Average over 5 runs, 95% c.i.

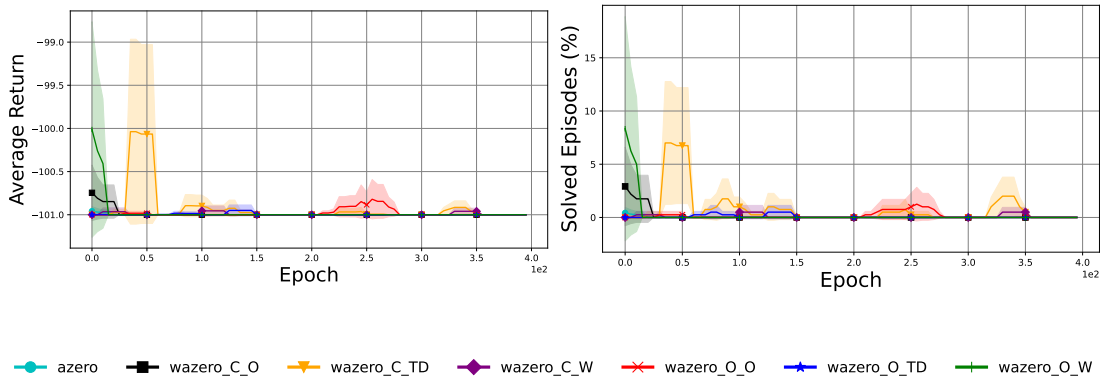


Figure 6.5: Performance of AlphaZero and Wasserstein AlphaZero in the harder navigation environment using 70 search iterations. Average over 5 runs, 95% c.i.

recall that these tasks consist in navigating in a 2D space with the goal of reaching a fixed goal state, avoiding one or more obstacles in the environment. We employ a sparse reward function, i.e., the agent receives a reward of -1 in each timestep until it reaches the goal. This way the optimal behavior is reaching the goal in the smallest number of steps. We reuse the same hyperparameters reported in Chapter 3 for both AlphaZero and Wasserstein AlphaZero, without tuning the initialization of the Q-posteriors for the latter due to computational constraints.

In Figure 6.4, we show the results of comparing AlphaZero and Wasserstein AlphaZero in the easiest configuration of the 2D navigation task we introduced in Chapter 3. We observe how AlphaZero again shows a more exploitative behavior, quickly learning an optimal behavior in 3 of the training runs and recovering suboptimal behaviors in the remaining 2 runs. Again after an initial learning period, AlphaZero settles on a policy for the remaining duration of learning. On the other hand, Wasserstein AlphaZero confirms

its overexplorative and unstable behavior, struggling to learn in most configurations but reaching the optimal behavior in all the runs for the wazero_C_W and wazero_O_TD configurations. This confirms our previous observations of learning instability and overexploration. We repeat the same experiment, in a harder version of the same navigation environment, by adding a harder obstacle on the the way to the goal and present the results in Figure 6.5. Like in Chapter 3, AlphaZero never observes the goal in this harder configuration. While Wasserstein AlphaZero manages to explore more and observe the goal during learning and obtain positive solving rates, it does not manage to maintain the learned policies demonstrating the same learning instabilities observed in the other environments.

To conclude, while the combination of Wasserstein TD with AlphaZero resulted in agents that explore the environment much more, this increased exploration resulted in high learning instabilities making it harder for the resulting agents to maintain a high performance. The TD learning procedure employed by Wasserstein AlphaZero proved to be too unstable to recover good, stable policies in the investigated environments indicating the need for a different learning algorithm for fitting the critic.

6.5 Conclusions and Discussion

In this chapter, we introduced Wasserstein AlphaZero, an attempt to combine the directed exploration of Wasserstein TD-Learning with AlphaZero to tackle hard exploration problems where a forward model of the environment is available. Wasserstein TD-learning provided us with the means to generalize, in a theoretically grounded way, epistemic uncertainty over the value function estimates and to exploit them during the Monte Carlo Tree Search. While in (Dam et al., 2023) the authors prove that a similar combination of MCTS and Wasserstein TD converges polynomially to the optimal policy in the pure MCTS case, when combining the algorithm with function approximators in an AlphaZero fashion, we observed high learning instabilities due to overexploration. Wasserstein AlphaZero, while able to explore the environment better compared to base AlphaZero, struggled to maintain the learned policies resulting in generally lower final performance.

An interesting future direction for research is the investigation of the effect of the critic learning procedure on the performance. While in AlphaZero, training the value network with Monte Carlo estimates of the return proved sufficient to recover a good performance, in Wasserstein AlphaZero we employ TD targets recovered from MCTS instead. Off-policy learning, paired with TD targets and function approximators has been shown in the literature to be hard to stabilize (van Hasselt et al., 2018) so a better investigation of the learning procedure might allow us to maintain the exploratory behavior of Wasserstein AlphaZero, while providing more stable learning.

CHAPTER 7

Conclusions and Future Works

In this thesis, we tackled the problem of improving the sample efficiency of Reinforcement Learning algorithms by exploiting theoretically grounded estimates of epistemic uncertainty over the value function. We introduced Wassertein TD-Learning, a novel framework for representing and propagating the uncertainty across the state-action space and incorporated it in several model-free and model-based algorithms allowing us to introduce provably sample-efficient algorithms in the tabular setting and extend them to the Deep Reinforcement Learning setting to enable tackling more complex tasks. We extend the framework to handle both continuous state spaces and continuous action spaces, as well as investigated some practical issues that arise when generalizing uncertainty estimates over the value function. Finally, we showed some promising results when expanding the method to the online planning setting, introducing Wasserstein AlphaZero, an extension of AlphaZero. While the resulting algorithm showed more explorative behavior, discovering hard to reach policies in hard exploration tasks, it struggled to maintain the performance due to high learning instabilities.

Despite the promising theoretical and practical results achieved during this research project, several interesting open questions and future directions of research remain open. In the following, we go over some important open questions.

Is the Wasserstein TD-Learning scalable to large models? We were able to extend the framework to several Deep Reinforcement Learning scenarios, ranging from Atari games in Chapter 4, continuous-action control in Chapter 5, and Monte Carlo Tree Search with value function approximators in Chapter 6 with promising results when handling deep exploration tasks. In the settings investigated in this thesis, we limited ourselves to shallow neural networks with a small number of hidden layers and a limited number of trainable parameters. It remains an open question whether the uncertainty estimates will remain informative in cases where the approximators used have parameters in the range of Millions or Billions. We hope the community builds on top of our work to scale our approach further.

How to handle over-exploration? In several of the investigated domains, while our proposed algorithms were able to explore much more than the considered baselines and base algorithms, they were often subject to over-exploration. This over-exploration often resulted in instabilities in practice, especially when combined with Monte Carlo Tree Search in Chapter 6. The hyperparameters defining the exploration budget also proved hard to tune in this case. It remains an open problem how to deal with this issue properly. A mechanism to tune the speed with which the uncertainty estimates decrease or increase online could be an interesting direction of research.

Can we perform efficient posterior sampling in continuous domains? In Chapter 5 We extended WTD to handle the case of continuous actions by devising an actor-critic algorithm that maintains an optimistic policy to collect samples as well as to define the value targets for the distributional critic. While this showed promising results for the considered hard exploration tasks, the question of optimizing a posterior sampling policy, i.e., a policy that maximizes the probability of the actions being optimal under the current Q-posteriors remains an open problem mainly due to the difficulty of defining a differentiable posterior sampling objective for the actor. An approximate scheme of optimizing the posterior sampling objective would allow moving away from optimistic exploration also in the continuous action case.

How to boost exploration when the exploration budget depletes? Wasserstein TD-learning initializes the Q-posteriors with an initial estimate of uncertainty which in the base case is uniform across the state-action space. This initialization defines a sort of exploration budget, which, while it is propagated and moved around across the state-action space, it depletes over time as more samples are needed. While in the tabular case, the initialization can be chosen so that it guarantees, in high probability, convergence

to the optimal policy with a sublinear convergence rate, when moving to DeepRL, it becomes a hyperparameter of the method. When initialized too large, the Q-posteriors will result in overexploration but eventually will reach a high performance, as we observed in our empirical studies. When the initialization of uncertainty is too low, WTD lacks a mechanism to realize this and boost the amount of exploration. An interesting direction would be to investigate how we can increase the uncertainty estimates when we observe value targets that are too unlikely under the current Q-posteriors, which would indicate an underestimation of the uncertainty.

Bibliography

Mohammed Amin Abdullah, Hang Ren, Haitham Bou Ammar, Vladimir Milenkovic, Rui Luo, Mingtian Zhang, and Jun Wang. Wasserstein robust reinforcement learning. *arXiv preprint arXiv:1907.13196*, 2019.

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *CoRR*, abs/1703.01732, 2017. URL `http://arxiv.org/abs/1703.01732`.

Martial Agueh and Guillaume Carlier. Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf`.

Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 49–56, 2007.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2–3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL `https://doi.org/10.1023/A:1013689704352`.

Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/file/e4a6222cdb5b34375400904f03d8e6a5-Paper.pdf.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 263–272, 2017.

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

Aijun Bai, Feng Wu, and Xiaoping Chen. Posterior sampling for monte carlo planning under uncertainty. *Applied Intelligence*, 48(12):4998–5018, Dec 2018. ISSN 1573-7497. doi: 10.1007/s10489-018-1248-5. URL https://doi.org/10.1007/s10489-018-1248-5.

Peter L Bartlett and Ambuj Tewari. Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 35–42. AUAI Press, 2009.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1479–1487, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 2017a.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017b. URL https://proceedings.mlr.press/v70/bellemare17a.html.

Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN 9780486428093.

Donald A Berry and Bert Fristedt. Bandit problems: sequential allocation of experiments (monographs on statistics and applied probability). *London: Chapman and Hall*, 5: 71–87, 1985.

Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Gianmarco Reho, Nico Montali, Marcello Restelli, and Cristiana Corno. Foreign exchange trading: a risk-averse batch reinforcement learning approach. In *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450375849. doi: 10.1145/3383455.3422571. URL `https://doi.org/10.1145/3383455.3422571`.

Ronen I. Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3: 213–231, 2002.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A:1007379606734. URL `https://doi.org/10.1023/A:1007379606734`.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=AY8zfZm0tDd`.

Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/a34bacf839b923770b2c360eefa26748-Paper.pdf`.

Adrien Couetoux. *Monte Carlo tree search for continuous and stochastic sequential decision making problems*. PhD thesis, Université Paris Sud, 2013.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors,

Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 1104–1113. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018b. ISBN 978-1-57735-800-8.

Tuan Dam, Pascal Klink, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Generalized mean estimation in monte-carlo tree search. *CoRR*, abs/1911.00384, 2019. URL `http://arxiv.org/abs/1911.00384`.

Tuan Dam, Pascal Stenger, Lukas Schneider, Joni Pajarinen, Carlo D'Eramo, and Odalric-Ambrym Maillard. Monte-carlo tree search with uncertainty propagation via optimal transport, 2023.

Christoph Dann and Emma Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2818–2826, 2015.

Richard Dearden, Nir Friedman, and Stuart J. Russell. Bayesian q-learning. In Jack Mostow and Chuck Rich, editors, *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 761–768. AAAI Press / The MIT Press, 1998.

Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

Carlo D'Eramo, Marcello Restelli, and Alessandro Nuara. Estimating maximum expected value through gaussian approximation. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1032–1040. JMLR.org, 2016.

Carlo D'Eramo, Alessandro Nuara, Matteo Pirotta, and Marcello Restelli. Estimating the maximum expected value in continuous reinforcement learning problems. In Satinder P.

Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1840–1846. AAAI Press, 2017.

Vikas Dhiman, Shurjo Banerjee, Jeffrey Mark Siskind, and Jason J. Corso. Floyd-warshall reinforcement learning learning from past experiences to reach new goals. *CoRR*, abs/1809.09318, 2018. URL `http://arxiv.org/abs/1809.09318`.

Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear quadratic control: an introduction*. Krieger Publishing Co., Inc., 2000.

Nicolai Dorka, Tim Welschehold, and Wolfram Burgard. Dynamic update-to-data ratio: Minimizing world model overfitting, 2023.

DC Dowson and BV Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego—an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:259–270, 2010.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017a. URL `http://proceedings.mlr.press/v70/finn17a.html`.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017b. URL `https://proceedings.mlr.press/v70/finn17a.html`.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *CoRR*, abs/1806.02817, 2018. URL `http://arxiv.org/abs/1806.02817`.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Emma Brunskill. Regret minimization in mdps with options without prior knowledge. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/90599c8fdd2f6e7a03ad173e2f535751-Paper.pdf`.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Ronald Ortner. Efficient bias-span-constrained exploration-exploitation in reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1573–1581. PMLR, 2018.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/fujimoto18a.html`.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8 (5-6):359–483, 2015.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *CoRR*, abs/1811.07819, 2018. URL `http://arxiv.org/abs/1811.07819`.

Marek Grzes and Daniel Kudenko. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *2009 International Conference on Machine Learning and Applications*, pages 337–344, 2009. doi: 10.1109/ICMLA.2009.33.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/haarnoja18b.html`.

Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL `https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf`.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1117–1125, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

Osband Ian, Van Roy Benjamin, and Russo Daniel. (more) efficient reinforcement learning via posterior sampling. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3003–3011, Red Hook, NY, USA, 2013. Curran Associates Inc.

Nozomu Ikehata and Takeshi Ito. Monte-carlo tree search in ms. pac-man. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 39–46, 2011. doi: 10.1109/CIG.2011.6031987.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.*, 6(6):1185–1201, November 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.6.1185. URL `http://dx.doi.org/10.1162/neco.1994.6.6.1185`.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010a.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.*, 11:1563–1600, aug 2010b. ISSN 1532-4435.

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are pac-learnable. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1704–1713. JMLR. org, 2017.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is q-learning provably efficient? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4868–4878. Curran Associates, Inc., 2018.

Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. Sub-goal trees - a framework for goal-based reinforcement learning. *CoRR*, abs/2002.12361, 2020. URL `https://arxiv.org/abs/2002.12361`.

Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46056-5.

Sven Koenig and Reid G Simmons. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical report, Carnegie Mellon University Pittsburgh PA School of Computer Science, 1992.

Tor Lattimore. *Bandit algorithms / Tor Lattimore (deepMind) and Csaba Szepesvari (University of Alberta).* Cambridge University Press, Cambridge, United Kingdom ;, 2020. ISBN 9781108486828.

Tor Lattimore and Marcus Hutter. Near-optimal PAC bounds for discounted mdps. *Theor. Comput. Sci.*, 558:125–143, 2014.

Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. Open loop execution of tree-search algorithms. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2362–2368. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/327. URL `https://doi.org/10.24963/ijcai.2018/327`.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6131–6141. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/lee21g.html`.

D Leibfried, E Knill, Christian Ospelkaus, and David J Wineland. Transport quantum logic gates for trapped ions. *Physical Review A*, 76(3):032324, 2007.

Robert Lieck, Vien Ngo, and Marc Toussaint. Exploiting variance information in monte-carlo tree search. June 2017. ICAPS workshop: Heuristics and Search for Domain-independent Planning (HSDIP) ; Conference date: 20-06-2017.

Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems*, 131:103568, 2020. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2020.103568. URL `https://www.sciencedirect.com/science/article/pii/S0921889020304085`.

Amarildo Likmeta, Matteo Sacco, Alberto Maria Metelli, and Marcello Restelli. Wasserstein actor-critic: directed exploration via optimism for continuous-actions control. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i7.26056. URL https://doi.org/10.1609/aaai.v37i7.26056.

Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

Puze Liu, Davide Tateo, Haitham Bou Ammar, and Jan Peters. Robot reinforcement learning on the constraint manifold. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1357–1366. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/liu22c.html.

Yishay Mansour and Satinder P. Singh. On the complexity of policy iteration. *CoRR*, abs/1301.6718, 2013. URL http://arxiv.org/abs/1301.6718.

Dmitri Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics*, 19(2):023035, 2017.

Mausam and Andrey Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1): 1–210, 2012. doi: 10.2200/S00426ED1V01Y201206AIM017. URL https://doi.org/10.2200/S00426ED1V01Y201206AIM017.

Borislav Mavrin, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4424–4434. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/mavrin19a.html.

Stephen McAleer, Forest Agostinelli, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with approximate policy iteration. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Hyfn2jCcKm.

John Mern, Anil Yildiz, Zachary Sunberg, Tapan Mukerji, and Mykel J. Kochenderfer. Bayesian optimized monte carlo planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11880–11887, May 2021. doi: 10.1609/aaai.v35i13.17411. URL https://ojs.aaai.org/index.php/AAAI/article/view/17411.

Alberto Maria Metelli, Amarildo Likmeta, and Marcello Restelli. Propagating uncertainty in reinforcement learning via wasserstein barycenters. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/f83630579d055dc5843ae693e7cdafe0-Paper.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Lorenzo Moro, Matteo G. A. Paris, Marcello Restelli, and Enrico Prati. Quantum compiling by deep reinforcement learning. *Communications Physics*, 4(1):178, Aug 2021. ISSN 2399-3650. doi: 10.1038/s42005-021-00684-3. URL https://doi.org/10.1038/s42005-021-00684-3.

Lorenzo Moro, Amarildo Likmeta, Enrico Prati, and Marcello Restelli. Goal-directed planning via hindsight experience replay. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=6NePxZwfae.

Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, and Michael Jordan. Tactical optimism and pessimism for deep reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12849–12863. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/6abcc8f24321d1eb8c95855eab78ee95-Paper.pdf.

Remi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014. doi: 10.1561/2200000038. URL https://doi.org/10.1561%2F2200000038.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. Task-agnostic exploration via

policy gradient of a non-parametric state entropy estimate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9028–9036, 2021.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/c8cc6e90ccbff44c9cee23611711cdc4-Paper.pdf`.

Michael A Nielsen. A simple formula for the average gate fidelity of a quantum dynamical operation. *Physics Letters A*, 303(4):249–252, 2002.

Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Vlad Mnih. The uncertainty Bellman equation and exploration. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3839–3848, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3003–3011, 2013.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4026–4034, 2016a.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2377–2386. JMLR.org, 2016b.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31:*

Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada., pages 8626–8638, 2018.

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2721–2730. JMLR.org, 2017.

Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Anna Choromanska, Krzysztof Choromanski, and Michael I. Jordan. Behavior-guided reinforcement learning. *arXiv preprint arXiv:1906.04349*, 2019.

Giambattista Parascandolo, Lars Buesing, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica B. Hamrick, Nicolas Heess, Alexander Neitz, and Theophane Weber. Divide-and-conquer monte carlo tree search for goal-directed planning. *CoRR*, abs/2004.11410, 2020. URL `https://arxiv.org/abs/2004.11410`.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. *CoRR*, abs/1906.04161, 2019. URL `http://arxiv.org/abs/1906.04161`.

Jason Pazis, Ronald Parr, and Jonathan P. How. Improving PAC exploration using the median of means. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3891–3899, 2016.

Diego Piccinotti, Amarildo Likmeta, Nicolo Brunello, and Marcello Restelli. Online planning for f1 race strategy indentification. 2021.

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. *CoRR*, abs/1609.09025, 2016. URL `http://arxiv.org/abs/1609.09025`.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, Mar 2011. ISSN 1573-7470. doi: 10.1007/s10472-011-9258-6. URL `https://doi.org/10.1007/s10472-011-9258-6`.

Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

Mark Rowland, Marc G. Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. In Amos J. Storkey and Fernando Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 29–37. PMLR, 2018.

G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

Nicholas J Russell, Levon Chakhmakhchyan, Jeremy L O'Brien, and Anthony Laing. Direct dialling of haar random unitary matrices. *New Journal of Physics*, 19(3):033007, 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/schaul15.html`.

Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987. URL `http://www.idsia.ch/~juergen/diploma.html`.

Jürgen Schmidhuber. Optimal ordered problem solver. *CoRR*, cs.AI/0207097, 2002. URL `https://arxiv.org/abs/cs/0207097`.

Jürgen Schmidhuber. *Gödel Machines: Fully Self-referential Optimal Universal Self-improvers*, pages 199–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-68677-4. doi: 10.1007/978-3-540-68677-4_7. URL `https://doi.org/10.1007/978-3-540-68677-4_7`.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, pages 9443–9454. PMLR, 2021.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017a. URL `http://arxiv.org/abs/1712.01815`.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017b. ISSN 1476-4687. doi: 10.1038/nature24270. URL `https://doi.org/10.1038/nature24270`.

Satinder P. Singh, Tommi S. Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008. doi: 10.1016/j.jcss.2007.08.009.

Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 881–888, 2006.

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Cs. Szepesvári. The asymptotic convergence-rate of q-learning. In *Proceedings of the 10th International Conference on Neural Information Processing Systems*, NIPS'97, page 1064–1070, Cambridge, MA, USA, 1997. MIT Press.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Istvan Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1031–1038, 2010.

Gerald Tesauro, V. T. Rajan, and Richard B. Segal. Bayesian inference in monte-carlo tree search. *CoRR*, abs/1203.3519, 2012. URL `http://arxiv.org/abs/1203.3519`.

Ambuj Tewari and Peter L Bartlett. Optimistic linear programming gives logarithmic regret for irreducible mdps. In *Advances in Neural Information Processing Systems*, pages 1505–1512, 2008.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Hado Van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 2094–2100. AAAI Press, 2016.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648, 2018. URL `http://arxiv.org/abs/1812.02648`.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *Adaptive Dynamic Programming and*

Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on, pages 177–184. IEEE, 2009.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Edoardo Vittori, Amarildo Likmeta, and Marcello Restelli. Monte carlo tree search for trading and hedging. In *Proceedings of the Second ACM International Conference on AI in Finance*, ICAIF '21, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391481. doi: 10.1145/3490354.3494402. URL https://doi.org/10.1145/3490354.3494402.

Tom Vodopivec, Spyridon Samothrakis, and Branko Šter. On monte carlo tree search and reinforcement learning. *J. Artif. Int. Res.*, 60(1):881–936, sep 2017. ISSN 1076-9757.

Hang Wang, Sen Lin, and Junshan Zhang. Adaptive ensemble q-learning: Minimizing estimation bias via error feedback. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24778–24790. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/cfa45151ccad6bf11ea146ed563f2119-Paper.pdf.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989a.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989b.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pages 11920–11931. PMLR, 2021.

Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph E. Gonzalez, and Stuart Russell. MADE: Exploration via maximizing deviation from explored regions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=DTVfEJIL3DB.

Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13344–13362, 2023. doi: 10.1109/TPAMI.2023.3292075.

APPENDIX $\mathcal{A}$

Technical Appendix

Index of the Appendix

In the following, we briefly recap the contents of the Appendix.

– Appendix A.1 provides additional details about the properties of the Wasserstein metric and the Wasserstein barycenters.

– Appendix A.2 reports all proofs and derivations.

A.1 Details on Wasserstein distance

In this appendix, we provide some additional details on Wasserstein distance and some properties of our approximate posterior distribution models. It can be proved Villani (2008) that the functions W_p are metrics on the sets $\mathscr{P}_p(\mathcal{X})$. Moreover, the following monotonicity property holds: $W_p(\mu,\nu) \leq W_q(\mu,\nu)$ if $p \leq q$. We will assume that all the involved probability measures μ admit cumulative distribution function (c.d.f.) F_μ and probability density function (p.d.f.) f_μ w.r.t. the Lebesgue measure. When $\mathcal{X} = \mathbb{R}$ and $d(x,y) = |x-y|$ is the Euclidean distance, the Wasserstein distance can be rephrased in

terms of the quantile functions:

$$W_p(\mu,\nu) = \left(\int_0^1 \left| F_\mu^{-1}(t) - F_\nu^{-1}(t) \right|^p \mathrm{d}t \right)^{1/p}, \tag{A.1}$$

where F^{-1} is the quantile function, i.e., $F_\mu^{-1}(t) = \inf\{x \in \mathbb{R} : t \leq F_\mu(x)\}$. The L^2-Wasserstein distance admits a closed form for both the Gaussian and particle models. For two Gaussian distributions, considering just the univariate case, $\mu = \mathcal{N}(m_1, \sigma_1^2)$ and $\nu = \mathcal{N}(m_2, \sigma_2^2)$, we have Dowson and Landau (1982):

$$W_2(\mu,\nu)^2 = (m_1 - m_2)^2 + (\sigma_1 - \sigma_2)^2. \tag{A.2}$$

For the particle models, with the same weighting w_j we have the following result.

Proposition A.1.1. *Let μ and ν be two mixture of M Dirac deltas with the same weighting w_j for $j = 1, 2, ..., M$ having $f_\mu(x) = \sum_{j=1}^M w_j \delta(x - x_j)$ and $f_\nu(x) = \sum_{j=1}^M w_j \delta(x - y_j)$ as p.d.f.s with $x_1 \leq x_2 \leq ... \leq x_M$ and $y_1 \leq y_2 \leq ... \leq y_M$. Then, the L^2-Wasserstein distance between μ and ν is given by:*

$$W_2(\mu,\nu)^2 = \sum_{j=1}^M w_j (x_j - y_j)^2. \tag{A.3}$$

Proof. We use Equation (A.1), thus we need to compute the quantile function of a mixture of M Dirac deltas. Let us introduce the intervals $I_j = [t_{j-1}, t_j)$, $t_0 = 0$, $t_j = \sum_{k=1}^j w_k$ and $|I_j| = t_j - t_{j-1}$ for $j = 1, 2, ...N$. It is clear that:

$$F_\mu^{-1}(t) = \sum_{j=1}^M x_j \mathbb{1}_{I_j}(t), \quad F_\mu^{-1}(t) = \sum_{j=1}^M y_j \mathbb{1}_{I_j}(t),$$

where $\mathbb{1}_\mathcal{X}$ is the indicator function of the set $\mathcal{X}$. Thus we can compute the Wasserstein distance by employing Equation (A.1):

$$\begin{aligned}
W_2(\mu,\nu)^2 &= \int_0^1 \left| F_\mu^{-1}(t) - F_\nu^{-1}(t) \right|^2 \mathrm{d}t \\
&= \sum_{j=1}^M \int_{I_j} \left| F_\mu^{-1}(t) - F_\nu^{-1}(t) \right|^2 \mathrm{d}t \\
&= \sum_{j=1}^M \int_{I_j} (x_j - y_j)^2 \mathrm{d}t \\
&= \sum_{j=1}^M (x_j - y_j)^2 \int_{I_j} \mathrm{d}t \\
&= \sum_{j=1}^M w_j (x_j - y_j)^2,
\end{aligned}$$

where we observed that $\int_{I_j} \mathrm{d}t = t_j - t_{j-1} = w_j$. □

A.1.1 Approximation of an arbitrary distribution with a mixture of Deltas

In this section, we show that we are able to approximate an arbitrary distribution with a mixture of Deltas, provided that we consider a sufficiently large number of components.

Proposition A.1.3. *Let μ be an arbitrary probability measure over the interval $[a, b] \subset \mathbb{R}$ admitting F_μ as c.d.f. and let ν be a mixture of M Dirac deltas $\widehat{x}_1 \leq \widehat{x}_2 \leq ... \leq \widehat{x}_M$ weighted by w_j for $j = 1, 2, ..., M$ fixed. Then, the L^2-Wasserstein $W_2(\mu, \nu)$ has a unique minimizer:*

$$\widehat{x}_j = \frac{1}{|I_j|} \int_{I_j} F_\mu^{-1}(t) \mathrm{d}t, \quad j = 1, 2, \ldots, M,$$

where $I_j = [t_{j-1}, t_j)$, $t_0 = 0$, $t_j = \sum_{k=1}^{j} w_k$ and $|I_j| = t_j - t_{j-1}$ for $j = 1, 2, ...N$. In this case, the L^2-Wasserstain distance can be bounded as:

$$W_2(\mu, \nu)^2 \leq \frac{(b-a)^2}{4} \max_{j=1,2,\ldots,n} |I_j|.$$

Proof. Let us first compute the quantile function of ν, i.e., F_ν^{-1}:

$$F_\nu^{-1}(t) = \sum_{j=1}^{M} x_j \mathbb{1}_{I_j}(t). \tag{A.4}$$

Using Equation (A.1), the L^2-Wasserstein distance can be written as:

$$\begin{aligned} W_2(\mu, \nu)^2 &= \int_0^1 \left(F_\mu^{-1}(t) - F_\nu^{-1}(t)\right)^2 \mathrm{d}t \\ &= \sum_{j=1}^{M} \int_{I_j} \left(F_\mu^{-1}(t) - x_j\right)^2 \mathrm{d}t. \end{aligned}$$

The objective is clearly convex, thus, we take the derivative w.r.t. x_j and we get:

$$\frac{\partial W_2^2}{\partial x_j} = -2 \int_{I_j} \left(F_\mu^{-1}(t) - x_j\right) \mathrm{d}t = 0, \tag{A.5}$$

from which the first result follows, by observing that $\int_{I_j} \mathrm{d}t = |I_j|$. Let us now observe, since F_μ^{-1} is monotonically increasing, that for every j:

$$\begin{aligned} \int_{I_j} \left(F_\mu^{-1}(t) - \widehat{x}_j\right)^2 \mathrm{d}t &\leq \int_{I_j} \left(F_\mu^{-1}(t) - \frac{F_\mu^{-1}(t_j) + F_\mu^{-1}(t_{j-1})}{2}\right)^2 \mathrm{d}t & \text{(A.6)} \\ &\leq \frac{1}{4} \int_{I_j} \left[(F_\mu^{-1}(t) - F_\mu^{-1}(t_j)) + (F_\mu^{-1}(t) - F_\mu^{-1}(t_{j-1}))\right]^2 \mathrm{d}t \\ &\leq \frac{1}{4} \int_{I_j} \left[(F_\mu^{-1}(t) - F_\mu^{-1}(t_{j-1})) - (F_\mu^{-1}(t_j) - F_\mu^{-1}(x))\right]^2 \mathrm{d}t & \text{(A.7)} \\ &\leq \frac{1}{4} \int_{I_j} \left(F_\mu^{-1}(t_j) - F_\mu^{-1}(t_{j-1})\right)^2 \mathrm{d}t \\ &\leq \frac{1}{4} |I_j| \left(F_\mu^{-1}(t_j) - F_\mu^{-1}(t_{j-1})\right)^2, \end{aligned}$$

where (A.6) follows from the fact that $\widehat{x}_j$ is the minimizer and (A.7) derives from observing that $F_\mu^{-1}(t) - F_\mu^{-1}(t_{j-1}) \leq 0$ by definition of $F_\mu^{-1}(t_{j-1})$ whereas $F_\mu^{-1}(t) - F_\mu^{-1}(t_{j-1}) \geq 0$. Let us rename $\Delta_j = F_\mu^{-1}(t_j) - F_\mu^{-1}(t_{j-1})$, the error can be bounded overall as:

$$\begin{aligned} W_2(\mu,\nu)^2 &\leq \frac{1}{4}\sum_{j=1}^{M} |I_j|\Delta_j^2 \\ &\leq \frac{1}{4}\max_{j=1,2,\ldots,M} |I_j| \sum_{j=1}^{n} \Delta_j^2 \\ &\leq \frac{1}{4}\max_{j=1,2,\ldots,M} |I_j| \left(\sum_{j=1}^{M} \Delta_j\right)^2 \\ &\leq \frac{1}{4}\max_{j=1,2,\ldots,M} |I_j| \,(b-a)^2\,, \end{aligned}$$

where we used Cauchy–Schwarz inequality in the last passage, observing that $\Delta_j \geq 0$ from the monotonicity of F_μ^{-1}. □

When we consider a uniform particle model, i.e., $w_j = 1/M$, the result reduces to:

$$W_2(\mu,\nu)^2 \leq \frac{(b-a)^2}{4M}. \tag{A.8}$$

The result tells us that when $M \to \infty$ the error vanishes as expected. Up to now, we considered the error introduced by representing a given distribution with a mixture of deltas. It is interesting to investigate the properties of the approximating distribution ν.

Lemma A.1.1. *Let ν be the best mixture of deltas L^2-Wasserstein approximation of an arbitrary distribution μ, as defined in Proposition A.2.11. If μ admits expectation, then it holds that:*

$$\mathop{\mathbb{E}}_{X\sim\mu}[X] = \mathop{\mathbb{E}}_{X\sim\nu}[X]. \tag{A.9}$$

Proof. We are going to assume that μ admits a p.d.f. f_μ and we indicate with f_ν the p.d.f. of ν. We first observe that by making the substitution $x = F_\mu^{-1}(t)$, we have the identity:

$$\int_{I_j} F_\mu^{-1}(t)\mathrm{d}t = \int_{F_\mu^{-1}(t_{j-1})}^{F_\mu^{-1}(t_j)} x f_\mu(x)\mathrm{d}x. \tag{A.10}$$

Therefore:

$$\begin{aligned}
\mathop{\mathbb{E}}_{X\sim\nu}[X] &= \int_a^b x f_\nu(x)\mathrm{d}x \\
&= \sum_{j=1}^{M} w_j \widehat{x}_j \\
&= \sum_{j=1}^{M} w_j \frac{1}{|I_j|}\int_{I_j} F_\mu^{-1}(t)\mathrm{d}t \\
&= \sum_{j=1}^{M} \int_{I_j} F_\mu^{-1}(t)\mathrm{d}t \\
&= \sum_{j=1}^{M} \int_{F_\mu^{-1}(t_{j-1})}^{F_\mu^{-1}(t_j)} x f(x)\mathrm{d}x \\
&= \int_a^b x f_\mu(x)\mathrm{d}x = \mathop{\mathbb{E}}_{X\sim\mu}[X].
\end{aligned}$$

□

Thus, our approximation preserves the mean, but unfortunately, this does not hold for the higher order moments.

A.1.2 Combining Posteriors via Wasserstein Barycenters

In this section, we introduce the notion of Fréchet mean, we show how it reduces to the Wasserstein barycenter when selecting the Wasserstein distance as a metric.

Definition A.1.2. *Let $(\mathcal{X}, d)$ be a complete separable metric (Polish) space and $\mu \in \mathscr{P}_2(\mathcal{X})$ a probability measure over $\mathcal{X}$. The Fréchet mean is defined as:*

$$\overline{x} = \arg\inf_{x\in\mathcal{X}} \mathop{\mathbb{E}}_{Y\sim\mu}\left[d(x,Y)^2\right]. \tag{A.11}$$

The Fréchet mean generalizes the notion of mean by minimizing the expected value of a metric. In the particular case in which $\mathcal{X} = \mathbb{R}$, and $d(x,y) = |x-y|$ is the Euclidean distance, the Fréchet mean is the expectation of X under μ, i.e., $\overline{x} = \mathbb{E}_{X\sim\mu}[X]$.[1]

In our scenario, we have that $\mathcal{X} = \mathscr{Q}$ is a set of probability measures (the Q-posteriors) and we select as metric $d = W_2$, i.e., the L^2-Wasserstein distance. Therefore, Equation (A.11) defines the *Wasserstein barycenter* Agueh and Carlier (2011). Typically, the notion of barycenter is defined in terms of a finite set of distributions Agueh and Carlier (2011). However, we can also consider infinite (possibly continuous) sets of distributions. For our posterior distribution models, the Wasserstein barycenter is unique and we have a closed form expression.

[1] Different choices of exponent to which $d(x,Y)$ is raised generate different indexes of central tendency, like median for exponent 1 and mode for the exponent going to 0 in the limit.

A.1.3 Closed-Forms for Wasserstein Barycenters of Gaussians and Particle models

We show in the following that the standard close forms for finite sets, when using Gaussian and particle models, naturally extends for continuous spaces.

Proposition A.1.7. *Let $(\mathcal{T}, \mathscr{F})$ be a measurable space and let $\mu \in \mathscr{P}(\mathcal{T})$ be a probability measure over $\mathcal{T}$. Let $\{\nu(t)\}_{t\in\mathcal{T}}$ be a family of probability measures. Then,*

$$\inf_{\nu\in\mathscr{N}} \mathop{\mathbb{E}}_{T\sim\mu}\left[W_2\left(\nu, \nu(T)\right)^2\right]$$

has a unique solution both for Gaussian and uniform particle models. In particular, for the Gaussian model, the parameters of the L^2-Wasserstein barycenter are:

$$\overline{m} = \mathop{\mathbb{E}}_{T\sim\mu}\left[\mu(T)\right], \quad \overline{\sigma} = \mathop{\mathbb{E}}_{T\sim\mu}\left[\sigma(T)\right],$$

and for equally weighted particle models:

$$\overline{x}_j = \mathop{\mathbb{E}}_{T\sim\mu}\left[x_j(T)\right], \quad j = 1, 2, ..., M.$$

Proof. All it takes is to write down the objective function and compute its minimizer. Let us start with the Gaussian model. The L^2-Wasserstein distance between two Gaussians is given in Equation (A.2), therefore our objective becomes:

$$\mathcal{L}(\mu, \sigma) = \mathop{\mathbb{E}}_{T\sim\mu}\left[W_2\left(\nu, \nu(T)\right)^2\right] = \mathop{\mathbb{E}}_{T\sim\mu}\left[(\overline{m} - m(T))^2 + (\overline{\sigma} - \sigma(T))^2\right],$$

where $\nu(T) = \mathcal{N}(m(T), \sigma^2(T))$ and $\overline{\nu} = \mathcal{N}(\overline{m}, \overline{\sigma}^2)$. The objective is clearly convex, therefore we just need to take the derivatives w.r.t. μ and σ:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\overline{\mu}} &= 2 \mathop{\mathbb{E}}_{T\sim\mu}\left[\overline{m} - m(T)\right] = 0 \implies \overline{m} = \mathop{\mathbb{E}}_{T\sim\mu}\left[m(T)\right], \\
\frac{\partial\mathcal{L}}{\partial\overline{\sigma}} &= 2 \mathop{\mathbb{E}}_{T\sim\mu}\left[\overline{\sigma} - \sigma(T)\right] = 0 \implies \overline{\sigma} = \mathop{\mathbb{E}}_{T\sim\mu}\left[\sigma(T)\right],
\end{aligned}$$

from which the result follows. Similarly, for the equally weighted particle models, using Proposition A.1.1, we have:

$$\begin{aligned}
\mathcal{L}(x_1, x_2, ..., x_M) &= \mathop{\mathbb{E}}_{T\sim\mu}\left[W_2\left(\nu, \nu(T)\right)^2\right] \\
&= \mathop{\mathbb{E}}_{T\sim\mu}\left[\sum_{j=1}^{M} w_j(\overline{x}_j - x_j(T))^2\right] \\
&= \sum_{j=1}^{M} w_j \mathop{\mathbb{E}}_{T\sim\mu}\left[(\overline{x}_j - x_j(T))^2\right].
\end{aligned}$$

We take the derivatives again and we obtain for every $j = 1, 2, ..., M$:

$$\frac{\partial \mathcal{L}}{\partial \overline{x}_j} = 2w_j \mathop{\mathbb{E}}_{T\sim\mu} [\overline{x}_j - x_j(T)] = 0 \implies \overline{x}_j = \mathop{\mathbb{E}}_{T\sim\mu} [x_j(T)] .$$

□

In other words, the L^2-Wasserstein barycenter of a set of Gaussians is a Gaussian distribution having as mean the expectations of the mean and standard deviation the expectations of the standard deviations. Similarly, the L^2-Wasserstein barycenter of a set of uniform mixtures of deltas is a uniform mixture of deltas where each particle is located at the expectation of the locations.

A.2 Proofs and Derivations

Proposition 4.3.1. *If $\mathscr{Q}$ is the set of deterministic distributions over $\mathbb{R}$, then the WTD update rule (Equation (4.13)) has a unique solution that corresponds to the TD update rule.*

Proof. It is a simple application of Proposition A.1.6 for the particle model setting $M = 1$, $\mu = (1 - \alpha_t, \alpha_t)^T$, $\nu_1 = Q_t(S_t, A_t)$ and $\nu_2 = R_{t+1} + \gamma V_t(S_{t+1})$. □

A.2.1 Provable Efficiency

In this section, we provide a series of results about the provable efficiency of a slightly modified version of WQL. We restrict our attention to the Gaussian model and we consider the optimistic estimator (OE) for the maximum and optimistic exploration (OX). We will consider w.l.o.g. the following assumption.

Assumption A.2.1. *The reward function is deterministic and positive.*

As a consequence we have that $0 \leq Q^\pi(s, a) \leq Q_{\max}$. Several parts of the proofs we are going to present are inspired to Jin et al. (2018). We now illustrate how to modify WQL in order to have the desired theoretical guarantees.

Modified Gaussian WQL

First of all, we need to particularize the MWTD for the Gaussian case:

$$\begin{aligned}
\widetilde{m}_{t+1}(s, a) &= (1 - \alpha_t)\widetilde{m}_t(s, a) + \alpha_t \left(R_{t+1} + \gamma m_t(S_{t+1})\right), \\
\widetilde{\sigma}_{t+1}(s, a) &= (1 - \alpha_t)\widetilde{\sigma}_t(s, a) + \alpha_t \gamma \sigma_t(S_{t+1}), \\
m_{t+1}(s, a) &= \widetilde{m}_{t+1}(s, a) + \beta_t m_b, \\
\sigma_{t+1}(s, a) &= \widetilde{\sigma}_{t+1}(s, a) + \beta_t \sigma_b,
\end{aligned}$$

with $m_b = 0$ by definition. We also define the auxiliary quantities that account for the accumulated effect of the learning rate:

$$\alpha_t^0 = \prod_{i=0}^{t}(1-\alpha_i), \qquad \alpha_t^i = \alpha_i \prod_{j=i+1}^{t}(1-\alpha_j)$$

It is clear that, by definition, for any $t = 0, 1, ...$ we have that $\alpha_0^t + \sum_{i=1}^{t} \alpha_i^t = 1$. Using these quantities we can rewrite the update rules for the mean and the standard deviation:

$$m_t(s,a) = \alpha_{n_t(s,a)} m_0 + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left[R_{t_i+1} + \gamma m_{t_i}(S_{t_i+1})\right],$$
$$\sigma_t(s,a) = \alpha_{n_t(s,a)} \sigma_0 + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \sigma_{t_i}(S_{t_i+1}) + \beta_t \sigma_b,$$

where $n_t(s,a)$ is the number of times pair (s,a) was visited up to time t, t_i is the time at which pair (s,a) was visited for the i-th time, $m_{t_i}(S_{t_i+1}) = m_{t_i}(S_{t_i+1}, \overline{a})$ and $\sigma_{t_i}(S_{t_i+1}, \overline{a})$, where $\overline{a} = \arg\max_{a\in\mathcal{A}} u^\delta_{t_i}(S_{t_i+1}, a)$ is the action that maximizes the upper bound of the Q-function, defined as:

$$\overline{u}_t^\delta(s,a) = m_t(s,a) + z_{1-\delta}\sigma_t(s,a), \quad u_t^\delta(s,a) = \min\left\{\overline{u}_t^\delta(s,a), Q_{\max}\right\} \tag{A.12}$$

Notice that we can define an update rule for the upper bound $\overline{u}_t^\delta(s,a)$ too:

$$\begin{aligned}
\overline{u}_t^\delta(s,a) &= \alpha_{n_t(s,a)}\left[m_0 + z_{1-\delta}\sigma_0\right] \\
&\quad + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left[R_{t_i+1} + \gamma\left(m_{t_i}(S_{t_i+1}) + z_{1-\delta}\sigma_{t_i}(S_{t_i+1})\right)\right] + \beta_t\sigma_b \\
&= \alpha_{n_t(s,a)}\left[m_0 + z_{1-\delta}\sigma_0\right] + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left[R_{t_i+1} + \gamma u_t^\delta(S_{t_i+1})\right] + \beta_t\sigma_b
\end{aligned}$$

where $u_t(S_{t_i+1})$ is the upper bound of the V-function defined as:

$$u_t^\delta(s) = \max_{a\in\mathcal{A}}\left\{u_t^\delta(s,a)\right\}. \tag{A.13}$$

Learning Rate

In this section, we introduce the learning rates α_t and β_t we will use to prove our theoretical results and we will present some properties we are going to exploit for the subsequent proofs:

$$\alpha_t = \frac{a}{b+t}, \qquad \beta_t = \frac{c}{\sqrt{d+t}}. \tag{A.14}$$

with $0 < a \leq b+1$, $b \geq 1$, $0 < c^2 \leq d$, $d \geq 1$ and $d \leq b$, whose values will be specified later. This choice of learning rates allows us to prove the following properties.

Lemma A.2.1. *If $a > 1$ and $b > 1$, the following relations hold for any $t = 1, 2, ...$ and $i = 1, 2, ...$:*

1. $\sum_{t=1}^{+\infty} \alpha_t^i \leq \frac{a}{a-1}$;

2. $\sum_{t=1}^{+\infty} \alpha_t^0 = \frac{b}{a-1} - 1$;

3. $\sum_{i=1}^{t} (\alpha_t^i)^2 \leq \frac{a}{b+t}$.

Proof. Let us start with 1. By using the properties of the Gamma function, we have that:

$$\sum_{t=1}^{M} \alpha_t^i = \frac{a}{a-1} \frac{\Gamma(b+i)}{\Gamma(1-a+b+i)} \left[\frac{\Gamma(2-a+b)}{\Gamma(1+b)} - \frac{\Gamma(2-a+b+M)}{\Gamma(1+b+M)} \right].$$

Since $a > 1$ we have that $2 - a + b + M < 1 + b + M$, thus for $M \to +\infty$ the second addendum goes to zero. Moreover, for the same reason $\frac{\Gamma(b+i)}{\Gamma(1-a+b+i)} \leq 1$ and $\frac{\Gamma(2-a+b)}{\Gamma(1+b)} \leq 1$. The result follows immediately. A similar argument can be stated for 2. By using the properties of the Gamma function, we have the following equality:

$$\sum_{t=1}^{M} \alpha_t^0 = \frac{b}{a-1} - 1 - \frac{\Gamma(1+b)\Gamma(2-a+b+M)}{(a-1)\Gamma(1-a+b)\Gamma(1+b+M)}.$$

Since $a > 1$ we have that $2 - a + b + M < 1 + b + M$ and therefore for $M \to +\infty$ the ratio of Gamma functions goes to zero, from which the result follows. Finally, for 3 we employ an argument similar to that of Lemma 4.1 (b) of Jin et al. (2018):

$$\begin{aligned} \alpha_t^i &= \frac{a}{b+i} \left(\frac{b+i+1-a}{b+i+1} \cdot \frac{b+i+2-a}{b+i+2} \cdots \frac{b+t-a}{b+t} \right) \\ &= \frac{a}{b+t} \left(\frac{b+i+1-a}{b+i} \cdot \frac{b+i+2-a}{b+i+1} \cdots \frac{b+t-a}{b+t-1} \right) \\ &\leq \frac{a}{b+t}, \end{aligned}$$

where we exploited the fact that $\frac{b+i+j-a}{b+i+j-1} \leq 1$ for all $j = 1, ...t - i$ being $a > 1$. Now, observing that $\sum_{i=1}^{t} \alpha_t^i \leq 1$, we have:

$$\sum_{i=1}^{t} (\alpha_t^i)^2 \leq \frac{a}{b+t} \sum_{i=1}^{t} \alpha_t^i \leq \frac{a}{b+t}.$$

□

Optimism

We now prove that with a suitable choice of m_0, σ_0 and σ_b we are able to guarantee that $u_t^\delta(s,a)$ is optimistic w.r.t. $Q^*(s,a)$ with high probability. We start proving the following intermediate result.

For any $\delta \in [0,1]$, with probability at least $1-\delta$, we have simultaneously for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \{1,2,...,T\}$:

$$\sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left[u^\delta_{t_i}(S_{t_i+1}) - \mathop{\mathbb{E}}_{S' \sim \mathcal{P}(\cdot|s,a)} \left[u^\delta_{t_i}(S') \right] \right] \leq Q_{\max} \sqrt{\sum_{i=1}^{n_t(s,a)} \left(\alpha^i_{n_t(s,a)} \right)^2 \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}.$$

Proof. Let us provide a formal definition of t_i:

$$t_i = \min \left(\{ t \in \{1,2,...,T\} : t > t_{i-1} \wedge (s_t, a_t) = (s,a) \} \cup \{T+1\} \right), \tag{A.15}$$

where we have assigned the value $T+1$ if (s,a) is experienced less than i times. Consider the filtration $\mathcal{F}_i = \sigma(S_0, A_0, R_1, ..., S_{t_i-1}, A_{t_i-1}, R_{t_i})$ generated by all the random variables realized until time t_i. The random variable $X_{t_i} = \mathbb{1}_{\{t_i \leq T\}} \left[u^\delta_{t_i}(S_{t_i+1}) - \mathbb{E}_{S' \sim \mathcal{P}(\cdot|s,a)} \left[u^\delta_{t_i}(S') \right] \right]$ is a martingale difference sequence (MDS) w.r.t. the filtration $\{\mathcal{F}_i\}_{i=1,2,...}$, as $\mathbb{E}[X_{t_i}|\mathcal{F}_i] = 0$ and $|X_{t_i}| < Q_{\max}$ a.s.. Using Azuma-Hoeffding inequality and a union bound over the time $\{1,2,...,T\}$, the states $\mathcal{S}$ and the actions $\mathcal{A}$ we have that w.p. at least $1-\delta$ the statement holds. □

We are now ready to prove that $u_t^\delta(s,a)$ are optimistic with high probability.

Theorem A.2.1. *(Optimism) Let $m_0 = Q_{\max}$, $\sigma_0 = 0$ and $\sigma_b = \frac{\gamma Q_{\max}}{c z_{1-\delta}} \sqrt{a \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$. Then, for any $\delta \in [0,1]$, with probability at least $1-\delta$, we have simultaneously for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \{1,2,...,T\}$:*

$$u_t^\delta(s,a) \geq Q^*(s,a). \tag{A.16}$$

Proof. The proof is by induction on t. For $t=0$, we have that $u_0^\delta(s,a) \geq Q_{\max} \geq Q^*(s,a)$. Let us assume the statement hold up to time $t-1$, we prove that it holds for t. Recall that $u_t^\delta(s,a) = \min\{\overline{u}_t^\delta(s,a), Q_{\max}\}$. If $u_t^\delta(s,a) = Q_{\max}$ then the statement hold. Otherwise we have $u_t^\delta(s,a) = \overline{u}_t^\delta(s,a)$. Let us write explicitly the expression of the upper bound

$\overline{u}_t^\delta(s,a)$. With probability at least $1-\delta$ we have:

$$\begin{aligned}
\overline{u}_t^\delta(s,a) &= \alpha_{n_t(s,a)}\left[m_0 + z_{1-\delta}\sigma_0\right] + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)}\left[R_{t_i+1} + \gamma u_t^\delta(S_{t_i+1})\right] + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b \\
&= \alpha_{n_t(s,a)}\left[m_0 + z_{1-\delta}\sigma_0\right] + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b \\
&\quad + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{S'\sim\mathcal{P}(\cdot|s,a)}\left[u^\delta_{t_i}(S')\right]\right] && \text{(A.17)} \\
&\quad + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)}\left[u^\delta_{t_i}(S_{t_i+1}) - \mathop{\mathbb{E}}_{S'\sim\mathcal{P}(\cdot|s,a)}\left[u^\delta_{t_i}(S')\right]\right] \\
&\geq \alpha^0_{n_t(s,a)} Q^*(s,a) + \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{S'\sim\mathcal{P}(\cdot|s,a)}\left[V^*(S')\right]\right] && \text{(A.18)} \\
&\quad + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b - \gamma Q_{\max}\sqrt{\sum_{i=1}^{n_t(s,a)}\left(\alpha^i_{n_t(s,a)}\right)^2 \log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} && \text{(A.19)} \\
&\geq Q^*(s,a) + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b - \gamma Q_{\max}\sqrt{\frac{a}{b+n_t(s,a)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}, && \text{(A.20)}
\end{aligned}$$

where line (A.17) derives from the fact that the reward is deterministic ($R_{t_i+1} = r(s,a)$), line (A.18) is an application of the inductive hypothesis being all $t_i < t$ and observing that $u^\delta_{t_i}(S') = \max_{a\in\mathcal{A}}\{u^\delta_{t_i}(S',a)\} \geq \max_{a\in\mathcal{A}}\{Q^*(S',a)\} = V^*(S')$ and we applied Lemma A.2.2 at line (A.19). Line (A.20) follows from the application of Bellman equation and using Lemma A.2.1 to bound the summation in the square root. In order to guarantee that this expression is non-negative for all $t \in \{1,2,...,T\}$ we need to satisfy:

$$\begin{aligned}
\beta_{n_t(s,a)} z_{1-\delta}\sigma_b &\geq \gamma Q_{\max}\sqrt{\frac{a}{b+n_t(s,a)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} \\
&\implies \sigma_b \geq \gamma q_{\max}\frac{\sqrt{d+n_t(s,a)}}{c z_{1-\delta}}\sqrt{\frac{a}{b+n_t(s,a)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}.
\end{aligned}$$

The term depending on t can be bounded recalling that $d \leq b$ and that $n_t(s,a) \leq T$:

$$\sqrt{\frac{d+n_t(s,a)}{b+n_t(s,a)}} \leq \sqrt{\frac{d+T}{b+T}} \leq 1.$$

Therefore, we choose:

$$\sigma_b = \frac{\gamma Q_{\max}}{c z_{1-\delta}}\sqrt{a\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}.$$

□

Main Result

We now provide this central lemma that we will use to state a bound on the sample complexity considering our running algorithm as a non stationary policy $\mathfrak{A}$.

Let $s \in \mathcal{S}$ be any state and let $\Delta_t(s) = V^*(s) - V_{\mathfrak{A}}(s)$ be the instantaneous regret of state s at time t and define $\Psi_t(s) = u_t^\delta(s) - V_{\mathfrak{A}}(s)$. Let $\delta \in [0,1]$, then with probability at least $1-\delta$ the following chain inequalities holds simultaneously for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \{1,2,...,T\}$: $\Delta_t(s) \leq \Psi_t(s)$ and

$$\Psi_t(s) \leq \alpha^0_{n_t(s,a)} Q_{\max} + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \Psi_{t_i}(s_{t_i+1}) + 2\gamma Q_{\max} \sqrt{\frac{a}{d + n_t(s,a)} \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}, \tag{A.21}$$

where $a \in \arg\max_{a' \in \mathcal{A}} \{u_t^\delta(s,a')\}$.

Proof. Consider a state $s \in \mathcal{S}$, we decompose the instantaneous regret at time t, i.e., $\Delta_t(s)$. It is important to notice that s is not necessarily the state visited by our algorithm at time t, i.e., s_t. With probability at least $1-\delta$ we have simultaneously for all t:

$$\begin{aligned}
\Delta_t(s) &= V^*(s) - V_{\mathfrak{A}}(s) \\
&= \max_{a \in \mathcal{A}} \{Q^*(s,a)\} - V_{\mathfrak{A}}(s) \\
&\leq \max_{a' \in \mathcal{A}} \{u_t^\delta(s,a')\} - V_{\mathfrak{A}}(s) && \text{(A.22)} \\
&= u_t^\delta(s) - V_{\mathfrak{A}}(s) = \Psi_t(s) && \text{(A.23)} \\
&\leq u_t^\delta(s,a) - Q_{\mathfrak{A}}(s,a), && \text{(A.24)}
\end{aligned}$$

were $a \in \arg\max_{a' \in \mathcal{A}} u_t^\delta(s,a')$. Line (A.22) follows from the optimism (Theorem A.2.1) and line (A.24) is obtained by observing that $\max_{a' \in \mathcal{A}} \{Q_{\mathfrak{A}}(s,a')\} \geq Q_{\mathfrak{A}}(s,a)$ for any

$a \in \mathcal{A}$. We now apply the Bellman equation on the upper confidence bound:

$$
\begin{aligned}
u_t^\delta(s,a) - Q_{\mathfrak{A}}(s,a) &\leq \overline{u}_t^\delta(s,a) - Q_{\mathfrak{A}}(s,a) && \text{(A.25)}\\
&= \alpha^0_{n_t(s,a)}\left(m_0 - Q_{\mathfrak{A}}(s,a)\right) + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b \\
&\quad + \sum_{i=1}^{n_t(s,a)} \gamma \alpha^i_{n_t(s,a)} \left(u^\delta_{t_i}(s_{t_i+1}) - \mathop{\mathbb{E}}_{S'\sim\mathcal{P}(\cdot|s,a)}\left[V_{\mathfrak{A}}(S')\right]\right) \\
&= \alpha^0_{n_t(s,a)}\left(m_0 - Q_{\mathfrak{A}}(s,a)\right) + \beta_{n_t(s,a)} z_{1-\delta}\sigma_b \\
&\quad + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left(u^\delta_{t_i}(s_{t_i+1}) - V_{\mathfrak{A}}(s_{t_i+1})\right) \\
&\quad + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left(V_{\mathfrak{A}}(s_{t_i+1}) - \mathop{\mathbb{E}}_{S'\sim\mathcal{P}(\cdot|s,a)}\left[V_{\pi_t}(S')\right]\right) \\
&\leq \alpha^0_{n_t(s,a)} Q_{\max} + \gamma Q_{\max}\sqrt{\frac{a}{d+n_t(s,a)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} && \text{(A.26)}\\
&\quad + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \left(u^\delta_{t_i}(S_{t_i+1}) - V_{\mathfrak{A}}(S_{t_i+1})\right) \\
&\quad + \gamma Q_{\max}\sqrt{\sum_{i=1}^{n_t(s,a)} \left(\alpha^i_{n_t(s,a)}\right)^2 \log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} && \text{(A.27)}\\
&\leq \alpha^0_{n_t(s,a)} Q_{\max} + 2\gamma Q_{\max}\sqrt{\frac{a}{d+n_t(s,a)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} + \gamma \sum_{i=1}^{n_t(s,a)} \alpha^i_{n_t(s,a)} \Psi_{t_i}(S_{t_i+1}),
\end{aligned}
$$

where line (A.26) follows from observing that $m_0 - Q_{\mathfrak{A}}(s,a) = Q_{\max} - Q_{\mathfrak{A}}(s,a) \leq q_{\max}$ and by substitution of the value of $\sigma_0^{(2)}(s,a)$ and line (A.27) is obtained by applying Azuma-Hoeffding inequality, like in Lemma A.2.2 (all it takes it to consider $V_{\mathfrak{A}}$ instead of $u^\delta_{t_i}$) and recalling that $d \leq b$. □

Using the previous lemma we are able to state the following theorem that represents the core of our analysis. In this case, we are going to evaluate how well is our algorithm performing (in terms of value function) over the states visited by the algorithm itself. This will allow us to derive immediately a guarantee on the sample complexity of PE-WQL.

Theorem A.2.2. *Let $S_0, S_1, ..., S_T$ be the sequence of states and actions visited by the algorithm. Let $a = \frac{2+\gamma}{2(1-\gamma)}$ and $b = a - 1$. Then, under the same assumptions as Lemma A.2.1, for any $\delta \in [0,1]$, with probability at least $1-\delta$ it holds that:*

$$
\sum_{t=1}^{T} \Delta_t(S_t) \leq \mathcal{O}\left(\frac{Q_{\max}}{(1-\gamma)^{3/2}}\sqrt{|\mathcal{S}||\mathcal{A}|T\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\right). \tag{A.28}
$$

Proof. We are now going do deal with the summation $\sum_{t=1}^{T} \Psi_t(s_{t+1})$:

$$\sum_{t=1}^{T} \Psi_t(s_{t+1}) = Q_{\max} \underbrace{\sum_{t=1}^{T} \alpha^0_{n_t(s_{t+1},a_{t+1})}}_{\text{(i)}} + \gamma Q_{\max} \sqrt{a \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} \underbrace{\sum_{t=1}^{T} \frac{1}{\sqrt{d + n_t(s_{t+1}, a_{t+1})}}}_{\text{(ii)}}$$
$$+ \underbrace{\gamma \sum_{t=1}^{T} \sum_{i=1}^{n_t(s_{t+1},a_{t+1})} \alpha^i_{n_t(s_{t+1},a_{t+1})} \Psi_{t_i(s_{t+1},a_{t+1})}(s_{t_i(s_{t+1},a_{t+1})+1})}_{\text{(iii)}}$$

We make the following observation we are going to use throughout the proof

$$n_t(s_{t+1}, a_{t+1}) = n_{t+1}(s_{t+1}, a_{t+1}) - 1. \tag{A.29}$$

Let us start with (i):

$$\begin{aligned}
\sum_{t=1}^{T} \alpha^0_{n_t(s_{t+1},a_{t+1})} &= \sum_{t=1}^{T} \alpha^0_{n_{t+1}(s_{t+1},a_{t+1})-1} \\
&= \sum_{h=2}^{T+1} \alpha^0_{n_h(s_h,a_h)-1} && \text{(A.30)} \\
&\leq \sum_{h=1}^{T+1} \alpha^0_{n_h(s_h,a_h)-1} \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{i=1}^{n_{T+1}(s,a)} \alpha^0_{i-1} && \text{(A.31)} \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{i=0}^{n_{T+1}(s,a)-1} \alpha^0_i \\
&\leq \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{i=0}^{+\infty} \alpha^0_i = \left(\frac{b}{a-1} + 1\right) |\mathcal{S}||\mathcal{A}|, && \text{(A.32)}
\end{aligned}$$

where we made the change of variable $h = t + 1$ to get line (A.30), we decomposed the summation over the state action pairs obseving that each of them appears $n_{T+1}(s, a)$ times to get line (A.31) and we used Lemma A.2.1 to get line (A.32).

Let us now consider (ii); using Equation (A.29) we get:

$$\begin{aligned}
\sum_{t=1}^{T} \frac{1}{\sqrt{d + n_t(s_{t+1}, a_{t+1})}} &= \sum_{t=1}^{T} \frac{1}{\sqrt{d + n_{t+1}(s_{t+1}, a_{t+1}) - 1}} \\
&= \sum_{h=2}^{T+1} \frac{1}{\sqrt{d + n_h(s_h, a_h) - 1}} \\
&\leq \sum_{h=1}^{T+1} \frac{1}{\sqrt{d + n_h(s_h, a_h) - 1}} \\
&= \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sum_{i=1}^{n_{T+1}(s,a)} \frac{1}{\sqrt{d + i - 1}} && \text{(A.33)} \\
&\leq 2 \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sqrt{n_{T+1}(s, a)} && \text{(A.34)} \\
&\leq 2\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)}, && \text{(A.35)}
\end{aligned}$$

where line (A.33) derives from decomposing the summation over state-action pairs and observing that each state-action pair appears $n_{T+1}(s, a)$ times. Line (A.34) is obtained by bounding the summation with the integral: $\int_1^{n_{T+1}(s,a)+1} \frac{1}{\sqrt{b+x-1}} \mathrm{d}x = 2\sqrt{d + n_{T+1}(s, a)} - 2\sqrt{d} \leq 2\sqrt{n_{T+1}(s, a)}$, where the last inequality derives from the subadditivity of the square root. Finally, line (A.35) is obtained by observing that $\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} n_{T+1}(s, a) = T + 1$ and the expression is maximized by taking $n_{T+1}(s, a) = \frac{T+1}{|\mathcal{S}||\mathcal{A}|}$.

Now we consider the term (iii). First observe that $n_t(s_{t+1}, a_{t+1}) \geq 1$ in order to

appear in the inner summation. Consider the derivation:

$$
\begin{aligned}
&\sum_{t=1}^{T} \sum_{i=1}^{n_t(s_{t+1},a_{t+1})} \alpha^i_{n_t(s_{t+1},a_{t+1})} \Psi_{t_i(s_{t+1},a_{t+1})}(s_{t_i(s_{t+1},a_{t+1})+1}) \\
&\quad = \sum_{t=1}^{T} \sum_{i=1}^{n_{t+1}(s_{t+1},a_{t+1})-1} \alpha^i_{n_{t+1}(s_{t+1},a_{t+1})-1} \Psi_{t_i(s_{t+1},a_{t+1})}(s_{t_i(s_{t+1},a_{t+1})+1}) \\
&\quad = \sum_{h=2}^{T+1} \sum_{i=1}^{n_h(s_h,a_h)-1} \alpha^i_{n_h(s_h,a_h)-1} \Psi_{t_i(s_h,a_h)}(s_{t_i(s_h,a_h)+1}) \\
&\quad \leq \sum_{h=1}^{T+1} \sum_{i=1}^{n_h(s_h,a_h)-1} \alpha^i_{n_h(s_h,a_h)-1} \Psi_{t_i(s_h,a_h)}(s_{t_i(s_h,a_h)+1}) && \text{(A.36)} \\
&\quad = \sum_{t=1}^{T+1} \Psi_t(s_{t+1}) \sum_{i=n_t(s_t,a_t)}^{n_{T+1}(s_t,a_t)} \alpha_{i-1}^{n_t(s_t,a_t)} && \text{(A.37)} \\
&\quad \leq \sum_{t=1}^{T+1} \Psi_t(s_{t+1}) \sum_{i=2}^{+\infty} \alpha_{i-1}^{n_t(s_t,a_t)} && \text{(A.38)} \\
&\quad \leq \sum_{t=1}^{T+1} \Psi_t(s_{t+1}) \sum_{i=1}^{+\infty} \alpha_{i}^{n_t(s_t,a_t)} \\
&\quad \leq \frac{a}{a-1} \sum_{t=1}^{T+1} \Psi_t(s_{t+1}) && \text{(A.39)} \\
&\quad \leq \frac{a}{a-1} \sum_{t=1}^{T} \Psi_t(s_{t+1}) + \frac{a}{a-1} Q_{\max}. && \text{(A.40)}
\end{aligned}
$$

where line (A.37) is obtained by observing that given $\Psi_t(s_{t+1})$ it is going to appear in the summation for all $t \geq t_i(s_t, a_t)$. The first time it will appear multiplied by $\alpha^{n_t(s_t,a_t)}_{n_t(s_t,a_t)-1}$, the second time with $\alpha^{n_t(s_t,a_t)}_{n_t(s_t,a_t)}$ and so on. Line (A.38) derives from observing that $n_t(s_{t+1}, a_{t+1}) \geq 1$, thus $n_{t+1}(s_{t+1}, a_{t+1}) \geq 2$. Line (A.39) is obtained by applying Lemma A.2.1.

Now, we put all together into the bound on the summation:

$$
\begin{aligned}
\sum_{t=1}^{T} \Psi_t(s_{t+1}) \leq Q_{\max} \left(\frac{b}{a-1} + 1 \right) |\mathcal{S}||\mathcal{A}| + 2\gamma q_{\max} \sqrt{a \log \frac{|\mathcal{S}||\mathcal{A}|T}{\delta}} \sqrt{|\mathcal{S}||\mathcal{A}|(T+1)} \\
+ \frac{a\gamma}{a-1} \sum_{t=1}^{T} \Psi_t(s_{t+1}) + \frac{a\gamma}{a-1} Q_{\max}.
\end{aligned}
$$

In order to solve the inequality, we must require $\frac{a\gamma}{a-1} < 1$, i.e., $a > \frac{1}{1-\gamma}$. In such a case,

we obtain:

$$\sum_{t=1}^{T}\Psi_t(s_{t+1}) \leq \left(1-\frac{a\gamma}{1-a}\right)^{-1}\left[Q_{\max}\left(\frac{b}{a-1}+1\right)|\mathcal{S}||\mathcal{A}| + 2\gamma Q_{\max}\sqrt{a\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)} + \frac{a\gamma}{a-1}Q_{\max}\right].$$

Now, we propose a value for a and b that fulfills all the conditions. In particular, among all possible values we select:[2] $a=\frac{2+\gamma}{2(1-\gamma)}$. Then we take the smallest possible value for b, i.e., $b=a-1$. From which we get:

$$\begin{aligned}\sum_{t=1}^{T}\Psi_t(s_{t+1}) &\leq \frac{3}{1-\gamma}\left[2q_{\max}|\mathcal{S}||\mathcal{A}| + 2\gamma Q_{\max}\sqrt{\frac{2+\gamma}{2(1-\gamma)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)} + \frac{2+\gamma}{3}Q_{\max}\right]\\ &\leq \frac{3}{1-\gamma}\left[Q_{\max}(|\mathcal{S}||\mathcal{A}|+1) + 2\gamma Q_{\max}\sqrt{\frac{3}{2(1-\gamma)}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)}\right]\\ &\leq \mathcal{O}\left(\frac{Q_{\max}}{(1-\gamma)^{3/2}}\sqrt{|\mathcal{S}||\mathcal{A}|T\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\right),\end{aligned}$$

where the last passage is obtained by observing that: if $T+1\geq\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)}$ then $\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)}\geq|\mathcal{S}||\mathcal{A}|$; on the contrary $\sum_{t=1}^{T}\Psi_t(s_{t+1})\leq T\leq\sqrt{|\mathcal{S}||\mathcal{A}|(T+1)}-1$. Thus, we can discard the first term. □

Theorem A.2.2 allows us to bound the per-step regret over the trajectories visited by the algorithm.

Corollary A.2.1. *Let $a=\frac{2+\gamma}{2(1-\gamma)}$ and $b=a-1$. Then, under the same assumptions as Lemma A.2.1, for any $\delta\in[0,1]$, with probability at least $1-\delta$ for $T\geq T_0$:*

$$T_0=\mathcal{O}\left(\frac{Q_{\max}^2|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3}\log\frac{Q_{\max}^2|\mathcal{S}|^2|\mathcal{A}|^2}{\delta\epsilon^2(1-\gamma)^3}\right)$$

we have that:

$$\frac{1}{T}\sum_{t=1}^{T}\Delta_t(S_t)\leq\epsilon.$$

Proof. Following the reasoning of Jaksch et al. (2010a), we just need to find a sufficiently large T_0 such that for all $T\geq T_0$ the per step regret is smaller than ϵ:

$$\frac{1}{T}\mathcal{O}\left(\frac{Q_{\max}}{(1-\gamma)^{3/2}}\sqrt{|\mathcal{S}||\mathcal{A}|T\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}}\right)<\epsilon \implies T\geq\mathcal{O}\left(\frac{Q_{\max}^2|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3}\log\frac{|\mathcal{S}||\mathcal{A}|T}{\delta}\right)$$

[2] It can be easily proved that taking the value of a that minimizes $\left(1-\frac{a\gamma}{1-a}\right)^{-1}\sqrt{a}$ just changes the bound by a constant and does not modify the dependence on $(1-\gamma)$.

We rename $\tau = \frac{Q_{\max}^2 |\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3}$. We select $T_0 = \mathcal{O}\left(2\tau \log \frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta}\right)$. Therefore, we have:

$$\begin{aligned}
T_0 &= \mathcal{O}\left(2\tau \log \frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta}\right) \\
&= \mathcal{O}\left(\tau \log \left(\frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta} \frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta}\right)\right) \\
&\geq \mathcal{O}\left(\tau \log \left(2\frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta} \log \frac{\tau|\mathcal{S}||\mathcal{A}|}{\delta}\right)\right) \\
&= \mathcal{O}\left(\tau \log \frac{T_0|\mathcal{S}||\mathcal{A}|}{\delta}\right) \\
&= \mathcal{O}\left(\frac{Q_{\max}^2|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3} \log \frac{|\mathcal{S}||\mathcal{A}|T_0}{\delta}\right),
\end{aligned}$$

where we exploited the inequality $x > 2\log x$ for $x > 0$. □

Finally, we prove that MWQL is PAC-MDP in the average loss setting. We import several ideas from Strehl et al. (2006). First of all, we recall the notion of adjusted loss.

Definition A.2.1 (Definition 5 of Strehl and Littman (2008))**.** *Suppose a learning algorithm $\mathfrak{A}$ is run for one sequence of $T_1 + T_2 - 1$ steps. Consider partial sequence $S_0, R_1, ..., S_{t-1}, R_t, S_t$ visited by $\mathfrak{A}$. For any policy π and integer t such that $t \leq T_1$, let $R_{T_2}^{\pi}(t) = \sum_{t'=t}^{t+T_2-1} \gamma^{t'-t} R_{t'+1} + \gamma^{T_2} v_{\pi}(S_{t+T_2})$ be the* adjusted return*. Let $I_{T_2}^{\pi}(t) = V^{\pi}(S_t) - R_{T_2}^{\pi}(t)$ be the* adjusted instantaneous loss*. Let $L_{T_1,T_2}^{\pi} = \frac{1}{T_1}\sum_{t=1}^{T_1} I_{T_2}^{\pi}(t)$ be the* adjusted average loss*.*

We are now ready to prove the result.

Theorem 4.5.2. *Under the hypothesis of Theorem 4.5.1, MWQL with Gaussian posterior, OE and OX is PAC-MDP in the average loss setting, i.e., for any $\epsilon \geq 0$ and $\delta \in [0, 1]$, after*

$$T = \mathcal{O}\left(\frac{Q_{\max}^2|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3} \log \frac{Q_{\max}^2|\mathcal{S}|^2|\mathcal{A}|^2}{\delta\epsilon^2(1-\gamma)^3}\right)$$

steps we have that the average loss $\mathcal{L}_{\mathfrak{A}} \leq \epsilon$ with probability at least $1 - \delta$.

Proof. A running algorithm can be viewed as a non stationary policy $\mathfrak{A}$. Define $T = T_1 + T_2 - 1$ and define the adjusted average loss of $\mathfrak{A}_t$ w.r.t. itself.

$$L_{T_1,T_2}^{\mathfrak{A}} = \frac{1}{T_1}\sum_{t=1}^{T_1} I_{T_2}^{\mathfrak{A}}(t). \tag{A.41}$$

In Strehl and Littman (2008) it is proven that for any algorithm $\mathfrak{A}$ and for

$$T_1 \geq \max\left\{\frac{1 + 2\log(1/\delta)Q_{\max}^2}{\epsilon^2(1-\gamma)^2}, 2\log(1/\delta)Q_{\max}^2(T_2 - 1)\right\}, \tag{A.42}$$

we have that $L^{\mathfrak{A}}_{T_1,T_2} \leq \epsilon$ with probability at least $1-\delta$. We now consider the adjusted average loss w.r.t. to the optimal policy π^* and we decompose it:

$$
\begin{aligned}
L^{\pi^*}_{T_1,T_2} &= \frac{1}{T_1}\sum_{t=1}^{T_1} I^{\pi^*}_{T_2}(t) && \text{(A.43)}\\
&= \frac{1}{T_1}\sum_{t=1}^{T_1} I^{\pi^*}_{T_2}(t) \pm \frac{1}{T_1}\sum_{t=1}^{T_1} I^{\mathfrak{A}}_{T_2}(t)\\
&\leq \epsilon + \frac{1}{T_1}\sum_{t=1}^{T_1}\left(I^{\pi^*}_{T_2}(t) - I^{\mathfrak{A}}_{T_2}(t)\right) && \text{(A.44)}\\
&= \epsilon + \frac{1}{T_1}\sum_{t=1}^{T_1}\left(V^*(S_t) - R^{\pi^*}_{T_2}(t) - \left(V_{\mathfrak{A}}(S_t) - R^{\mathfrak{A}}_{T_2}(t)\right)\right) && \text{(A.45)}\\
&= \epsilon + \frac{1}{T_1}\sum_{t=1}^{T_1}\left(V^*(S_t) - V_{\mathfrak{A}}(S_t)\right) + \frac{1}{T_1}\sum_{t=1}^{T_1}\gamma^{T_2}\left(V_{\mathfrak{A}}(S_{t+T_2}) - V^*(S_{t+T_2})\right) && \text{(A.46)}\\
&\leq \epsilon + \frac{1}{T_1}\sum_{t=1}^{T_1}\Delta_t(S_t) && \text{(A.47)}\\
&\leq 2\epsilon, && \text{(A.48)}
\end{aligned}
$$

where (A.44) derives from the fact that $L^{\mathfrak{A}}_{T_1,T_2} \leq \epsilon$, (A.45) is from the definition of adjusted instantaneous loss, (A.46) derives from the definition of adjusted return, (A.47) is obtained by observing that $V_{\mathfrak{A}}(S_{t+T_2}) \leq V^*(S_{t+T_2})$ and the definition of $\Delta_t(S_t)$ and (A.48) derives from Corollary A.2.1. Therefore, the inequality hold for T_1 satisfying condition (A.42) and Corollary A.2.1, with probability at least $1-2\delta$. Proposition 3 of Strehl and Littman (2008) proves that for $T_1 \geq \frac{2T_2}{\epsilon}$ and $T_2 \geq \frac{\log(\epsilon(1-\gamma))}{\log\gamma}$ we have that the adjusted loss $L^{\pi^*}_{T_1,T_2}$ is ϵ-close to the average loss $\mathcal{L}_{\mathfrak{A}}$. Therefore we have that $\mathcal{L}_{\mathfrak{A}} \leq 3\epsilon$ with probability at least $1-2\delta$ provided that:

$$
T_1 = \mathcal{O}\left(\frac{Q^2_{\max}|\mathcal{S}||\mathcal{A}|}{9\epsilon^2(1-\gamma)^3}\log\frac{Q^2_{\max}|\mathcal{S}|^2|\mathcal{A}|^2}{2\delta\epsilon^2(1-\gamma)^3}\right) = \mathcal{O}\left(\frac{Q^2_{\max}|\mathcal{S}||\mathcal{A}|}{\epsilon^2(1-\gamma)^3}\log\frac{Q^2_{\max}|\mathcal{S}|^2|\mathcal{A}|^2}{\delta\epsilon^2(1-\gamma)^3}\right). \tag{A.49}
$$

It can be easily proved that among all the conditions T_1 has to satisfy the most restrictive is the one imposed by Corollary A.2.1. □

APPENDIX B

Experimental Appendix

B.1 Experimental Details of Chapter 4

B.1.1 Reproducibility Details

In this section, we provide the hyper-parameters employed in the experiments presented in this work. Table B.1 and Table B.2 provide a list of hyperparameters employed for both AlphaZero and AlphaZeroHER, without being optimized. Moreover, Table B.3 lists the hyperparameters employed in thr DQN+HER experiments. We ran each experiment in a single multi-core machine, with no GPUs.

Table B.1: List of the hyperparameters and their values used in all environments.

Hyperparameter	Value
Optimizer	Adam
c_{uct}	2.0
Discount factor	0.999
Episodes per epoch	50

Table B.2: List of the hyperparameters and their values used in each environment.

Hyperparameter	Environment	Value
Learning rate	BitFlip	0.0005
	2D Navigation	0.001
	2D Maze	0.0005
	Quantum Compiling	0.00005
Batch size	BitFlip	256
	2D Navigation	512
	2D Maze	512
	Quantum Compiling	512
Search Iterations	BitFlip	20
	2D Navigation	70
	2D Maze	120
	Quantum Compiling	20

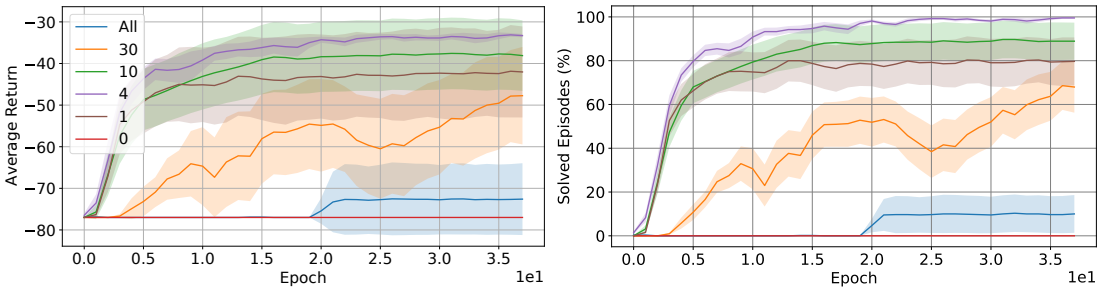


Figure B.1: Varying the number of subgoals in the BitFlip environment. Average of 10 runs, 95% c.i..

B.1.2 Varying the number of subgoals sampled

In this section, we study the effect of increasing the number of sampled subgoals in AlphaZeroHER. Figure B.1 and Figure B.2 show the results of varying the number of subgoal in the BitFlip and quantum compiling environments respectively. We can see that in both enviroments the performance increases as we increase the number of subgoals k, until we reach a (problem dependend) threshold after which the performance starts falling until it reaches the lower levels when we use as subgoals, all the available ones (label All). This is in line with the results presented in the original HER paper (Andrychowicz et al., 2017).

Table B.3: List of the hyperparameters and their values used in DQN+HER for each environment.

Hyperparameter	Environment	Value
Target net update frequency	BitFlip	100
	2D Navigation	500
	Quantum Compiling	500
Final exploration epsilon	BitFlip	0.21
	2D Navigation	0.21
	Quantum Compiling	0.15
Train Frequency	BitFlip	4
	2D Navigation	4
	Quantum Compiling	1
Learning Rate	BitFlip	0.00064
	2D Navigation	0.0007
	Quantum Compiling	0.00022
Batch Size	BitFlip	32
	2D Navigation	128
	Quantum Compiling	64
Buffer Size	BitFlip	500000
	2D Navigation	1000000
	Quantum Compiling	1000000

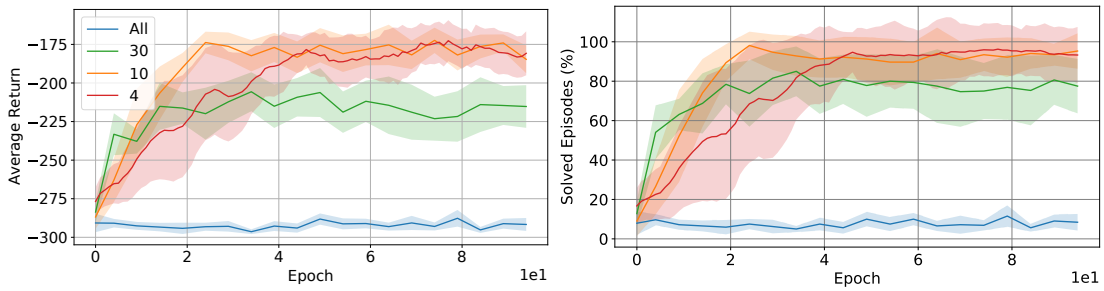


Figure B.2: Varying the number of subgoals in the quantum compiling environment. Average of 10 runs, 95% c.i..

B.2 Experimental Details of Chapter 5

In this section, we provide the experimental setup we adopted and some additional results we did not include in the main paper.

B.2.1 Tabular RL

Experimental Setup

We train each agent for 100 episodes, the length of each episode is domain dependent. During the training periods we collect the rewards and use them to calculate the *online scores*. After each training period, we turn off exploration and evaluate the greedy policies learned by the agent. The length of the evaluation episodes is the same as the training episodes. We use the rewards collected during evaluation to calculate the *offline* scores. We perform this process in each domain, for each algorithm considered and show the mean scores of 10 runs with 95% c.i.. We calculate the undiscounted scores, even though we use a discount factor $\gamma = 0.99$ in each domain during learning.

For our particle algorithms, we initialize the particles equally spaced in an interval $[q_{\min}, q_{\max}]$, for each state action-pair. For the Gaussian model we initialized $\mu_0(s,a) = (q_{\max} + q_{\min})/2$ and $\sigma_0(s,a) = (q_{\max} - q_{\min})/\sqrt{12}$. The range of this interval is problem dependent and we see these hyperparameters as a way to incorporate prior knowledge about the domain. We consider Bootstrapped Q-learning with two policy models, the Bootstrapped policy defined in Osband et al. (2016a) and the posterior sampling policy. We initialize the Q-tables with values drawn from a Gaussian distribution with parameters $\mu = \frac{q_{\min}+q_{\max}}{2}, \sigma = q_{\max} - q_{\min}$. Furthermore, we consider Q-learning algorithm with ϵ-greedy and Boltzmann exploration. In both Q-learning versions, the Q-table is initialized to 0. We compare our results with Delayed Q-learning Strehl et al. (2006), a model-free PAC-MDP algorithm. In each of the problems considered we tuned the m parameter, the number of visits necessary to attempt an update for each state-action pair, to find the one that yields better results. We did not employ the theoretical values being too much conservative.

For all algorithms, we use an *exponentially decaying* learning rate given by:

$$\alpha_t(s,a) = \frac{b}{t(s,a)^a}, \tag{B.1}$$

where $t(s,a)$ is the visit count for state-action pair (s,a), b is the initial value which we set to 1 and a is the *decay exponent*. We cross validated the value of a, which was set to $a = 0.2$, for all our experiments.

For the Q-learning algorithms, we had to chose also the schedules for ϵ and β, for ϵ-greedy and Boltzmann exploration respectively. For ϵ we used an exponentially decaying

schedule as in (B.1) with $b = 1$ and $a = 0.5$ whereas for the Boltzmann policy we used an exponentially decaying β with initial value, $b = 1.5q_{\max}$ and decay exponent, $a = 0.5$.

Here we show the results on more domains: Taxi, Chain and Loop domain from Dearden et al. (1998), River Swim and Six Arms from Strehl and Littman (2008) and Knight Quest from Fruit et al. (2018).

Comparison of WQL with State of the Art Algorithms

In Figure B.3, we show a comparison between the best version of WQL, using the two models to approximate posteriors, and the considered RL algorithms.

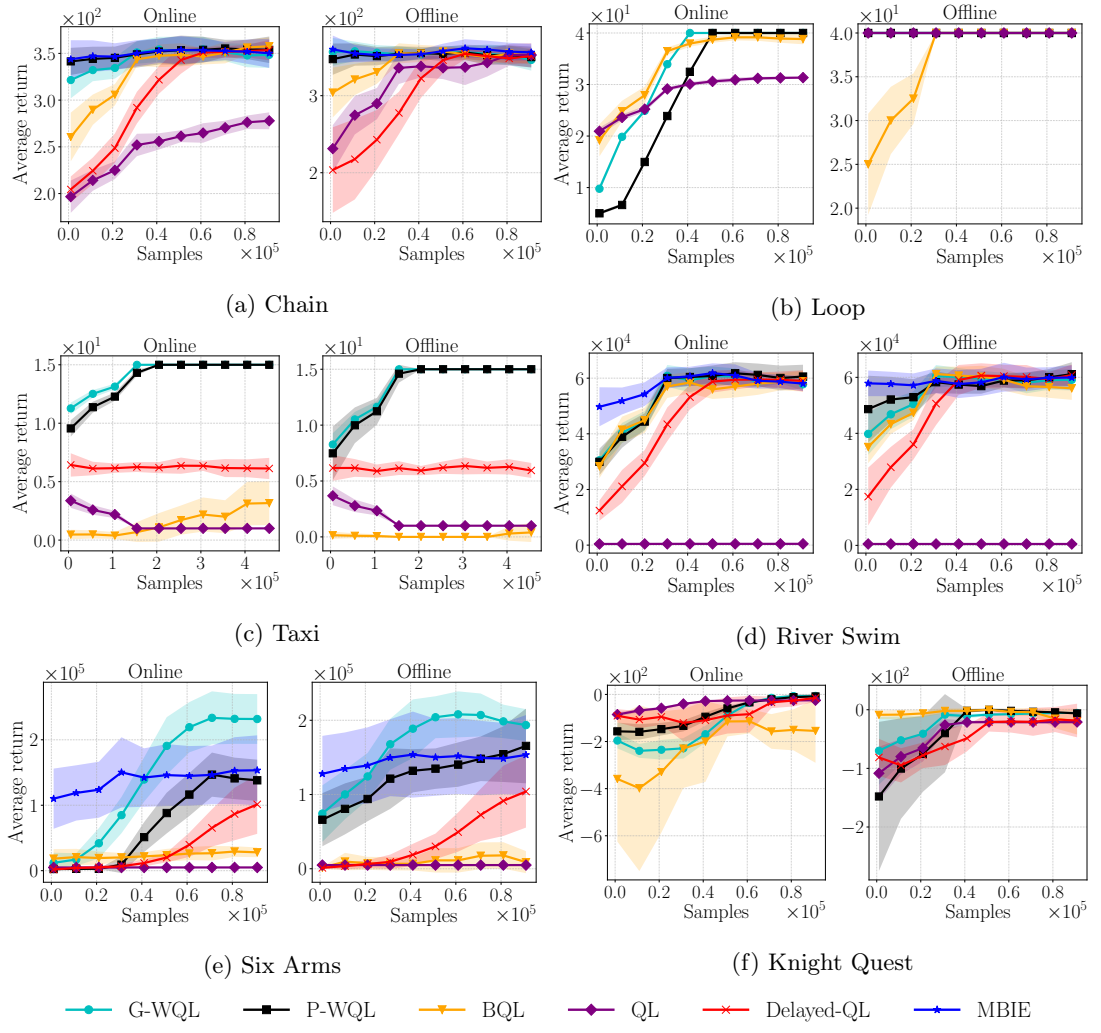


Figure B.3: Comparison of G-WQL, P-WQL, QL, BQL and Delayed QL. 10 runs, 95% c.i.

Results of WQL algorithm

Figure B.4 provides a full empirical analysis of the different flavors of WQL algorithms in the domains we considered.

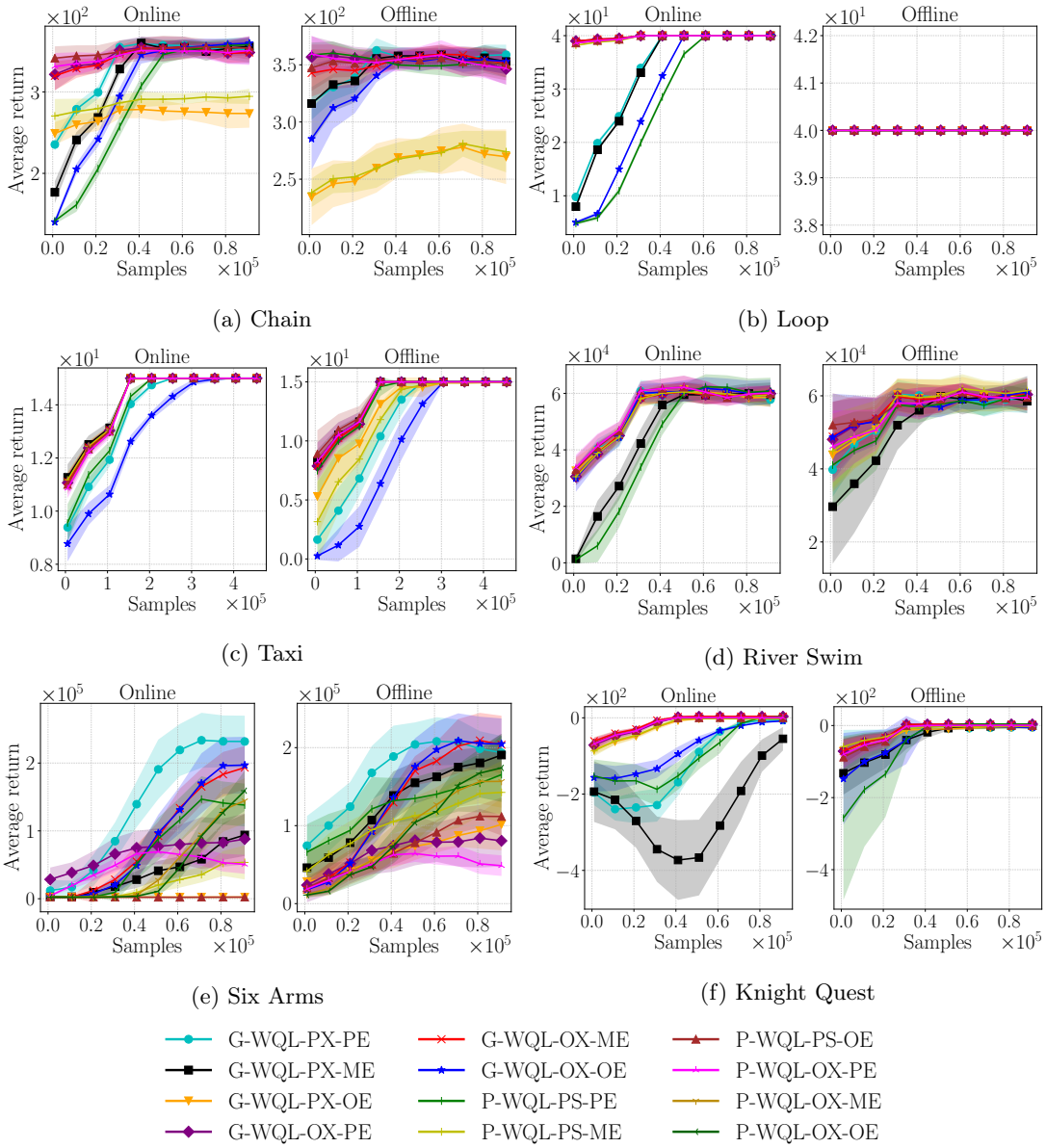


Figure B.4: Results of different variations of WQL algorithm. 10 runs, 95% c.i.

B.3 Experimental Details of Chapter 6

B.3.1 Environments Description

RiverSwim We extend the classical Riverswim domain (Strehl and Littman, 2008) to a continuous setting. In this environment, the agent has to navigate a 1 dimensional state space, ranging from 0 to max_state, by applying a 1 dimensional action, representing the intended movement $a \in [-1 \quad 1]$. The initial state is a uniformly distributed in $[0 \quad 0.5]$. When an action is chosen the agent moves left or right on the state space. The distance of the movement is equal to the absolute value of the action (if the result is outside the state space a clip operations brings it back inside). The direction $d \in \{-1, 0, 1\}$ of the movement is stochastic, according to the following probabilities:

$$P(d = -1|a) = \begin{cases} 1 - 0.9 \cdot (a + 1) & \text{if } a \leq 0 \\ 0.1 & \text{if } a > 0 \end{cases}$$

$$P(d = 0|a) = \begin{cases} 0.9 \cdot (a + 1) & \text{if } a \leq 0 \\ 0.9 - 0.3a & \text{if } a > 0 \end{cases}$$

$$P(d = 1|a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 0.3a & \text{if } a > 0 \end{cases}$$

Given the current state s_t, the action a_t and the direction d_t sampled according the previous probabilities, the next state is $s_{t+1} = \text{clip}(s_t + d_t|a_t|)$, where clip clips the state in the range $[0 \quad max\_state]$ The reward depends on the starting state s and the action sign:

$$r = \begin{cases} 5 \cdot 10^{-4} & \text{if } s \leq 1 \\ 1 & \text{if } s \geq (max\_state - 1) \text{ and } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

The optimal policy is to always perform $a = 1$ which gives the agent the best chance of moving toward high reward states.

In our experiments:

- $max\_state = 25$

LQG We test our agents also on an instance of a Linear Quadratic Gaussian control. Given a state x, an action a, and $v \sim \mathcal{N}(0, 0.5)$ the next state and the cost c $(= -r)$ are defined as:

$$x' = Ax + Ba + v$$
$$c = Qx^2 + Ra^2$$

In our experiments, we use $A = 1, B = 1, Q = 0.9, R = 0.9$. The agents starts from the borders of the state space and, with this configuration, the goal is to reach the origin of the state space while balancing the actions.

B.3.2 Reproducibility Details

In this appendix, we present the hyperparameter tuning employed, as well as the final values used in our experiments. For all the approximators employed, including critics and actors, we use 2 layer MLPs. For what concerns LQG all algorithms could solve it easily independently from the hyper-parameters chosen. The same can be said for RiverSwim, except for SAC which could not solve for any set of hyper-parameters contained in the grid search we performed.

In point environment we performed a grid search on all environments with dense rewards starting with SAC (3 runs with 3 different seeds for each node of the grid). We choose the set of hyper-parameters that could solve the most runs for the two most difficult environments in which at least one run could learn a policy that reached the goal. The best recovered values are reported in Table B.4.

Table B.4: SAC parameters

Parameter	**best value**
networks' number of layers	2
layers' size	256
replay buffer size	10^6
number of train steps per train loop	1000
number of exploration steps per train loop	1000
batch size	256
learning rates	10^{-3}

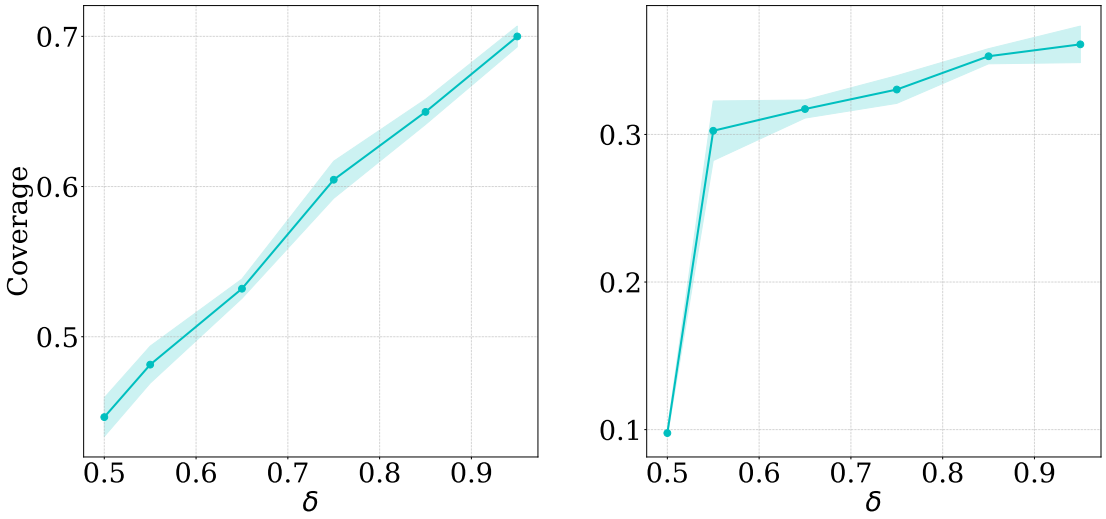
Afterwards, we performed a grid search on OAC and WAC, where we fixed all the hyper-parameters they share with SAC to the best values we found on SAC hyper-parameter tuning and we tuned only on their additional hyper-parameters. Once again, we choose the hyper-parameters sets that allow each algorithm to perform best the 2 most difficult environment it could solve. The final values are reported in Table B.5 and Table B.6.

Table B.5: OAC parameters

Parameter	best value
δ	18
β_{UB}	6.5

Table B.6: WAC parameters

Parameter	best value
δ	0.95
λ	0.6
ρ	0.6



(a) Coverage in LQG.

(b) Coverage in Riverswim.

Figure B.5: Coverage in LQG and Riverswim as function of δ; average of 5 seeds, 95% c.i..

B.4 Additional Experiments

B.4.1 Tuning on δ

In Section 3.5 we have shown how by tuning λ and ρ we can control the amount of exploration. We now show some experiment that illustrate how the hyper parameter δ can also be use to control exploration, since it defines what percentile of the estimated Gaussian distribution we use as upper bound. The results are reported in Figure B.5. We observe that also δ is directly related with the coverage. Indeed by increasing δ, we employ larger upper bounds for the value estimates, and this directly translates to larger coverage of the state-action space.

B.4.2 Coverage in OAC

We performed a similar study to the one presented in Section 3.5 on WAC for OAC, to investigate whether we could control how much the algorithm explores based on the values of the hyper-parameters. However, what we found is that is hard to predict OAC's

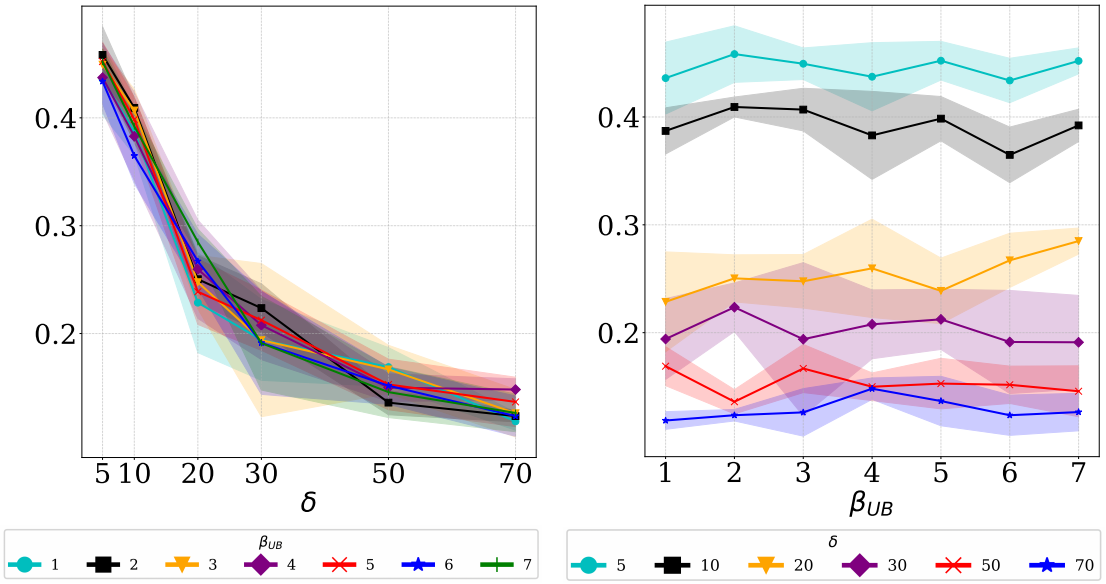


Figure B.6: Coverage in LQG as function of δ and β_{UB}; average of 5 seeds, 95% c.i..

exploration based on its hyper parameters δ and β_{UB}. In OAC, δ controls how much the exploration policy differs from the target policy. From Figure B.6, we can see that δ can even negatively affect exploration, if we allow the exploration policy to differ too much from the target policy. The dependence on β, which controls the definition of the upper bound (similar to our δ in OAC), suggests that the uncertainty estimate of OAC is not directly related to exploration either. We attribute both these results to the heuristic estimation of uncertainty that OAC employees, based only on the disagreement between the two critics. We argue that this uncertainty estimation is not enough to direct exploration meaningfully.