POLITECNICO DI MILANO
DEPARTMENT OF INFORMATION, ELECTRONICS AND BIOENGINEERING
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

# ON THE EXPLOITATION OF UNCERTAINTY TO IMPROVE BELLMAN UPDATES AND EXPLORATION IN REINFORCEMENT LEARNING

Doctoral Dissertation of:
**Carlo D'Eramo**

Supervisor:
**Prof. Marcello Restelli**

Tutor:
**Prof. Andrea Bonarini**

The Chair of the Doctoral Program:
**Prof. Andrea Bonarini**

2018 – XXXI cycle

# Abstract

The recent exponential growth of Reinforcement Learning (RL) research has been made possible by the comparable significant improvement in computational power. Indeed, the advent of powerful and relatively affordable hardware, in particular Graphics Processing Units (GPUs), allowed researchers to extend the study of RL methodologies to high-dimensional problems that were unpractical before, opening the line of research that is now commonly known under the name of Deep Reinforcement Learning (DRL). However, the groundbreaking results that DRL is achieving are obtained at the cost of a huge amount of samples needed for learning, along with very large learning times usually in the order of days. One of the reasons why this is happening, besides the outstanding significance of the results that fundamentally poses the problems of the efficiency of these methodologies in the background, relies on the fact that often experiments are run in simulations in which the sample efficiency problem is not such an issue as in real applications. Nevertheless, the issue of sample efficiency always constituted a matter of concern also in classic RL research where several works have been proposed to address the problem. It is historically well-known that this issue arises from the need of the agent to explore the environment it is moving in to improve its knowledge about it, and to exploit simultaneously the actions it considers to be the best to maximize its return, creating a trade-off known in RL as exploration-exploitation dilemma. The addressing of this trade-off is central and constitutes a measure of effectiveness of any algorithm available in literature.

The purpose of this thesis is to study the previously described problems proposing novel methodologies that explicitly consider the concept of *uncertainty* to speed up learning and improve its stability. Indeed, since a relevant goal of an RL agent is to reduce uncertainty about the environment in which it is moving, taking uncertainty explicitly into account can be intuitively an effective way of acting. This solution is not new in RL research, but there is still a lot of work that can be done in this direction and this thesis takes inspiration from the available literature on the subject extending it with novel significant improvements on the state of the art. In particular, the works included in this thesis can be grouped into two parts: one where uncertainty is used to improve the behavior of the Bellman equation and the other where it is used to improve exploration. The works belonging to the former group aim to address some of

the problems of action-value estimation in the context of value-based RL, in particular in the estimate of the maximum operator involved in the famous $Q$-Learning algorithm and more generally in the estimate of the components of the Bellman equation. On the other hand, the works belonging to the latter group study different methodologies to improve exploration by studying the use of Thompson Sampling in RL or by introducing a variant of the Bellman equation that incorporates an optimistic estimate of the action-value function to improve exploration according to the principle of Optimism in the Face of Uncertainty (OFU).

All the works presented in this thesis are described, theoretically studied, and eventually, empirically evaluated on several RL problems. The obtained results highlight the benefits that the explicit exploitation of uncertainty in RL algorithms can provide; indeed, we show how in a large set of problems that have been chosen in order to highlight particular aspects we were interested in, e.g. exploration capabilities, our methods prove to be more stable and faster to learn than others available in the literature.

# Contents

**Contents**

# List of Figures

# List of Algorithms

# Glossary

# Part I

# Starting Point

CHAPTER $1$

# Introduction

Everyone experiences the process of making decisions during his life. As a matter of fact, drastically the life of an individual can be synthesized in its *perception* of the world and its *interaction* with it. The concepts of perception and interaction may seem quite straightforward to understand: for a human the perception of the world comes from its senses and the interaction comes from its possibility to change its surroundings. On the contrary, these concepts are actually absolutely hard to define and aroused, over the centuries, a strong debate between scientists, biologists, and even philosophers.

## 1.1 Perception and interaction

We assume that, by definition, an individual perceives the surrounding and acts on it in order to achieve the *goals* expressed by its will. In other words, all the actions performed by an individual are made to satisfy its will to obtain something from the world it lives in. This task is naturally carried out by humans, but implies some challenging problems that are hard, or unfeasible, to solve. One of these derives from the intrinsic *uncertainty* of the perception we have of the world around us. Indeed, the perception of the world consists in the interpretation of the information provided by the senses, but the process of information retrieval by the senses and the mental processes to understand them, inevitably introduce a certain level of noise that distorts the true original information. On the other hand, interaction with the world deals with the individual's willingness to perform actions to change the surrounding, but this apparently simple operation involves complex biological mechanisms to coordinate the body according to the will and difficulties in the perception of the consequences of the interaction. Moreover, the concept of goal can be unclear and the individual may result in performing actions without being sure of what it wants. It is arguable that discussing about the
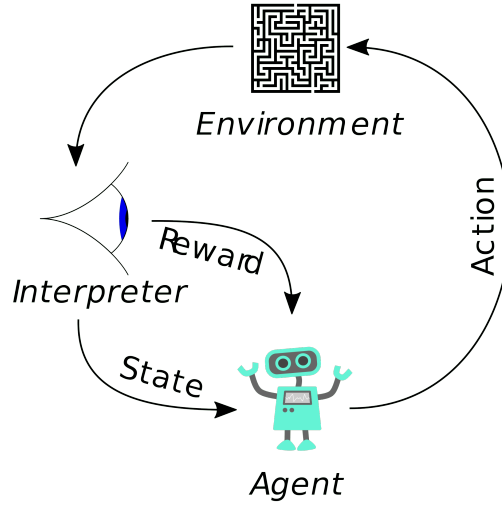
**Figure 1.1:** *The scheme of a Reinforcement Learning model.*

concept of true information and the concept of will requires strong theoretical considerations since they are both hard to define concepts. For many centuries, scientists and philosophers debated about these topics, in particular trying to solve complex problems like the real nature of perceivable things and the concept of free will. However, to make the discussion of these concepts suitable for our purposes throughout this thesis, we lighten their definition to the one provided by common sense.

## 1.2  Learn how to act with Reinforcement Learning

Reinforcement Learning (RL) [82] is a subfield of Machine Learning (ML) that aims to realize autonomous *agents* able to learn how to act in a certain *environment* in order to maximize an objective function; this is achieved by providing the agent with the perception of its *state* in the environment and making it learn the appropriate *action* to be performed, where an action can be seen as an atomic operation that brings the agent from one state to another. The objective function represents a measure of how well the agent is accomplishing its task in the environment and is usually formalized by a discounted sum of *rewards* obtained after each action (Figure 1.1). The sum is discounted to give more importance to the most recent rewards w.r.t. the ones further in the future. The reward function, i.e. the function returning the reward after each action, is not a concrete part of the environment, but it is often designed by a human who decides whether a certain behavior must be reinforced by (returning a positive reward) or inhibited (returning a negative reward).

### 1.2.1  Uncertainty in Reinforcement Learning

The major challenge of RL is represented by uncertainty. In fact, initially, the agent is not provided with the knowledge of how the environment will react to its actions, thus it does not know whether an action would be good or not to maximize its objective function. In other words, before trying an action, it does not know if that action will get a positive or a negative reward, and it does not know if that action will let it go to

4

the expected state or not. Thus, the former problem can be seen as uncertainty in the reward function and the latter as uncertainty in the transition (i.e. model) function. In some cases, also the uncertainty in the perception of the current state of the agent is considered, making the problem more complex.

The uncertainty issue results in the need of the agent to try actions in order to improve its knowledge of the environment. This process delays the collection of high rewards, but helps the agent to reduce its uncertainty. However, since the objective function is a sum of discounted rewards where later rewards worth less than recent ones, the agent also needs to learn fast in order to learn to perform the most rewarding actions as soon as possible. The need to *explore* to reduce uncertainty and the need to *exploit* the actions believed to be good introduces an important problem known as *exploration-exploitation dilemma*.

### 1.2.2 Balancing exploration and exploitation

The exploration-exploitation dilemma has been broadly studied in the Multi-Armed Bandit (MAB) field, a particular case of the RL problem with a single state [49]. In this problem, the goal is to find the sequence of optimal actions minimizing the choice of suboptimal ones, i.e. the sequence of actions that allows to maximize the return. The simplistic setting of the MAB problem allows to theoretically study the balancing of exploratory and exploitative actions, for instance, to derive upper confidence bounds on the *regret*, i.e. a measure of the return lost in performing non-optimal actions [1,18,94], and several algorithms to address this problem have been proposed such as UCB1 [5] and Thompson Sampling [89].

The RL setting complicates the MAB problem because of the presence of multiple states and of the fact that the sequence of visited states strictly depends on the chosen actions. This makes the exploration-exploitation dilemma less tractable in terms of computational and complexity feasibility. Indeed, the quality of the actions must now be evaluated for each state, contrary to the MAB case where the presence of a single state simplifies the problem. This issue is what makes RL so challenging and has been addressed for decades in the literature.

## 1.3 My research

The strong connection between uncertainty and the exploration-exploitation dilemma is highlighted by the previous considerations and it is intuitive how the effectiveness of an RL algorithm depends on its ability to reduce the uncertainty of the agent in a computational and data-efficient way. The RL literature contains many algorithms and methodologies proposed to make the agent learn a good policy by aiming at efficiency; however, despite addressing the reduction of uncertainty through experience, only a few of them explicitly model the uncertainty to learn.

### 1.3.1 What is my research about

During my Ph.D., we studied ways to develop algorithms that exploit uncertainty since the explicit consideration of it has been shown to be often useful for improving the performance and efficiency of learning. One of the most common techniques to explore is known as $\varepsilon$-greedy and consists in performing, at each state, a random action with

probability $\varepsilon$ and the action considered the best with probability $1 - \varepsilon$. This exploratory policy does not consider the uncertainty of the agent and simply randomly collects samples with the drawback of requiring a huge amount of experience to learn effective policies. This is shown especially in recent works in the field of Deep Reinforcement Learning (DRL) [55, 93, 96] which studies the application of Deep Learning (DL) [50] models and methodologies to exploit their strong ability to generalize with the purpose to solve complex problems that were unfeasible before. The research on DRL brought to the realization of groundbreaking works in which the authors were able to reach the state of the art in extremely complex games such as Go [75, 77] and chess [76]. However, the extraordinariness of these results is comparable to the amount of experience required by these algorithms to work. For instance, in [55] the experiments are performed using $50$ millions of samples corresponding to three days of computation and several weeks of human play; obviously this huge demand of time and computational power could make the execution of the algorithm impractical in some cases.

In addition to the exploration problem, we studied the problem of *estimating maximum action-values* in value-based RL. This latter problem is historically known in literature and is a critical issue in many value-based algorithms that may seriously affect the performance, as shown in recent works [79]. In particular, a well-known problem consists in the overestimation of the maximum action-values leading to poor empirical results in certain type of problems. To solve this, some recent works [90, 92, 99] proposed a new estimator that provides an underestimation instead of an overestimation, which is helpful in problems where the overestimation leads to bad results.

### 1.3.2 What we have done

We worked on the problems of exploration and maximum action-value estimation with the purpose to work out some new effective techniques. In particular, for the former problem, we developed two new exploration strategies based on the concept of Optimism in the Face of Uncertainty, a principle that exploits uncertainty to visit mostly unknown states in an environment. On the other hand, for the latter problem we worked on the proposal of a new maximum action-values estimator that computes an estimate of the maximum action-values basing on uncertainty. This estimate can both be positively and negatively biased, a property that helps to obtain good results in heterogeneous problems.

My Ph.D. research led to the publication of four conference papers, most of which focused on the previously described topic. We also developed, together with a colleague of mine, an RL Python library called *Mushroom* which had the initial purpose of facilitating my research, but which has become larger and larger allowing now to do RL research for general purposes. Details about Mushroom are reported in Appendix A. Other works presented here are still ongoing and/or under review.

### 1.3.3 Outline of the thesis

The whole document is divided in four parts, with this introduction included in the first one:

- the **introductory** part includes this chapter and Chapter 2 which introduces the main concepts of RL starting from the fundamental theory and then giving a

description of several methodologies related to this thesis in order to provide a general, but useful, overview of what is necessary to understand the following chapters;

- the **Bellman update** part includes the description of three publications I made on ways to exploit the uncertainty in the context of value-based RL and more specifically in the famous algorithm of $Q$-Learning. In particular, Chapter 3 describes a novel way of addressing the problem of overestimating the Maximum Expected Value in $Q$-Learning and Chapter 4 describes a novel way of dealing with uncertainty in the estimation of the Bellman operator components;

- the **Uncertainty-Driven Exploration** part includes two works about the exploitation of uncertainty to drive exploration. Chapter 5 describes a set of algorithms for estimating uncertainty of action-values to allow the use of an exploration policy inspired by Thompson Sampling. Chapter 6 introduces a variant of the Bellman operator that incorporates an optimistic update of the action-value estimate in order to favor exploration according to the principle of Optimism in the Face of Uncertainty;

- the **conclusive** and last part concludes the thesis by resuming the previous chapters and giving my considerations about the research I have done and on what I think will be interesting to pursue in the following years, by me or someone else!

# Preliminaries

RL is intuitively describable as the process of learning from interaction with the environment. This hasty explanation offers a very high-level definition of RL, then a more formal way to model the problem is required to analyze it properly. To this end, this chapter provides a description of the mathematical framework required to model RL. It also describes a selection of algorithms related to the work done in this thesis in order to provide enough knowledge about the literature I dealt with during my Ph.D. years.

## 2.1 Agent and environment

The interaction of an agent inside an environment can be seen as the execution of actions to change the environment and the observation of the consequences of its actions (Figure 2.1). The temporal progress of the interaction is modeled in a set of discrete time steps $t \in [0, 1, 2, \dots]$ where the agent sees a representation $s$ of the environment
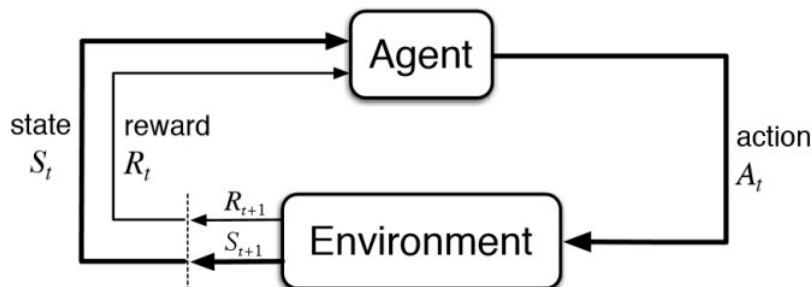


**Figure 2.1:** *Graphical representation of a Markov Decision Process.*

state, executes an action $a$ and observes the new representation of the state of the environment $s'$. The problems about observation and interaction discussed in Chapter 1 are simplified by an explicit selection of data to be observed from the environment and of the executable actions. In this way, only the relevant aspects of the sensory data acquired from the environment by the agent are used. The total number of time steps $t_i$ is called *horizon H* and determines a first taxonomy of problems:

- finite time horizon: $t_i, \forall i \in [0, 1, 2, \ldots, H)$;

- infinite time horizon: $t_i, \forall i \in [0, 1, 2, \ldots, \infty)$.

Some problems may terminate before reaching the horizon, which happens when the agent reaches special situations called *absorbing* states. These states are usually desired or catastrophic states when the agent's interaction with the environment is no longer useful or impossible. The set of steps between the start of the interaction and the end is called *episode*.

The agent's interaction with the environment is performed in order to reach a goal for which the agent has been designed. The way to give the knowledge of the goal to the agent is to provide it with a measure of the quality of its behavior. This measure is called *reward $R(s, a, s')$* and is a function that usually returns a real scalar value $r$ given the action $a$ performed by the agent in state $s$ and bringing to state $s'$. The agent's goal of the agent is to maximize a measure related to the collected rewards. In an infinite time horizon problem it can be:

- cumulative reward:

$$J = \sum_{t=0}^{\infty} r_t; \tag{2.1}$$

- average reward:

$$J = \lim_{n \to \infty} \frac{\sum_{t=0}^{n} r_t}{n}; \tag{2.2}$$

- discounted cumulative reward:

$$J = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{2.3}$$

The measure in Equation (2.3) uses a real scalar $\gamma \in [0, 1)$, called *discount factor*, which has the purpose of giving different importance to the rewards w.r.t. the time steps in which they have been collected. If $\gamma = 1$ the equation reduces to Equation (2.1), whereas the smaller it becomes the less the agent cares about rewards collected far in time.

## 2.2 Markov Decision Processes

The mathematical framework for studying the interaction of the agent with the environment is provided by the theory behind MDPs. A MDP is defined as a 6-tuple where $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma, \mu \rangle$:

- $\mathcal{S}$ is the set of states where the agent can be in the environment;
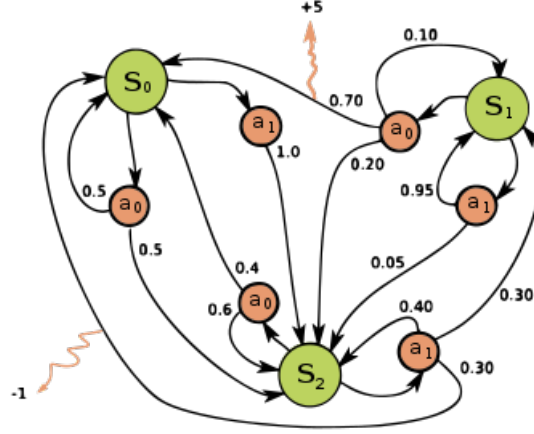
**Figure 2.2:** *An example of a Markov Decision Process (MDP) graph representing states $S_i$, actions $a_i$, transition probabilities (numbers over each edge) and rewards (pink arrows).*

- $\mathcal{A}$ is the set of actions that the agent can execute in the environment;

- $\mathcal{R}$ is the *reward function*;

- $\mathcal{T}$ is the *transition function* consisting in the probability of reaching a state $s'$ executing action $a$ in state $s$;

- $\gamma$ is the discount factor;

- $\mu$ is the probability for each state of being the initial one: $\mu(s) = P(s_0 = s)$.

An MDP is called *finite*, or *discrete*, if the set of states $\mathcal{S}$ and the set of actions $\mathcal{A}$ are finite; it is called *infinite*, or *continuous*, when the set of states $\mathcal{S}$ is infinite and/or the set of actions $\mathcal{A}$ is infinite. Two important properties of MDPs are:

- **stationarity:** the reward function $\mathcal{R}$ and the transition function $\mathcal{T}$ do not change over time;

- **Markovian assumption:** the transition and reward functions depend only on the current time step and not on the previous ones;

- **ergodicity:** an MDP is called *ergodic* when the agent can reach all the states of the MDP from every state.

These three properties are taken as assumptions by most of the literature about MDPs and by the works presented in this thesis too.

### 2.2.1 Value functions

Recalling that the agent's goal is to maximize the cumulative (discounted) reward $J$ obtained during an episode, an MDP is considered *solved* when the agent learns the actions to be performed in each state that maximizes this measure. The function that defines the probability of executing an action $a$ in a state $s$ is called *stochastic policy*: $\pi(a|s) = P(a|s)$; on the other hand, a *deterministic policy* $\pi(s)$ is the action $a$ taken

by the policy in state $s$. An *optimal* policy $\pi^*$ is the one that, when followed, allows the agent to solve the MDP. Considering the stochasticity in the transition function $\mathcal{T}$ and in the policy $\pi$, the expected value of the cumulative discounted reward obtainable following $\pi$ is called the *state-value function*

$$V_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0} \gamma^k r_{t+k} \right], \forall s \in \mathcal{S}. \tag{2.4}$$

Then, an optimal policy can be defined also as the one which maximizes the value function of each state

$$V^*(s) = \max_\pi V_\pi(s), \forall s \in \mathcal{S}. \tag{2.5}$$

Together with the state-value function, the *action-value function* is defined as

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0} \gamma^k r_{t+k} \right], \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.6}$$

Subsequently, the optimal policy maximizes the action value function in each state-action tuple

$$Q^*(s, a) = \max_\pi Q_\pi(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.7}$$

## 2.3 Solving an MDP

Value functions are the main concept used by several algorithms to address the problem of solving MDPs. In the following, a description of algorithms exploiting value functions to solve MDPs is provided, from the easiest to the hardest case.

### 2.3.1 Dynamic Programming

When the transition function $\mathcal{T}$ and the reward function $\mathcal{R}$ of an MDP are known, the full model of the environment is available. This is not the case in many real world problems where an agent does not know where the action would bring it and what return would obtain, but it is an interesting scenario to start studying the problem of solving an MDP. The theory underlying the resolution of MDPs with the full model available is related to the field of Dynamic Programming (DP) [13, 14]. The main concept in the research on DP to deal with MDP is the optimal Bellman Equation (BE), defined as

$$V^*(s) = \max_a \mathbb{E}[R(s, a, s') + \gamma V^*(s')]$$
$$= \max_a \sum_{s'} p(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \tag{2.8}$$

for state-value function, and

$$Q^*(s, a) = \mathbb{E}[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$
$$= \sum_{s'} p(s'|s, a)[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')] \tag{2.9}$$

---

**Algorithm 1** Iterative Policy Evaluation

---

1: **Inputs:** policy $\pi$ to evaluate, a small threshold $\theta$ determining the accuracy of the estimate
2: **Initialize:** $V(s), \forall s \in \mathcal{S}$ arbitrarily, $V(s') = 0$ for all terminal states $s'$
3: **repeat**
4:    $\Delta \leftarrow 0$
5:    **for all** $s \in \mathcal{S}$ **do**
6:       $v \leftarrow V(s)$
7:       $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$
8:       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
9:    **end for**
10: **until** $\Delta < \theta$

---

for action-value function $\forall (s, s') \in \mathcal{S} \times \mathcal{S}$ and $a \in \mathcal{A}$. The optimal BE serves as a way to derive the optimal policy, but requires the optimal value functions to be known. Usually, the optimal value functions are unknown and in order to learn them several algorithms change the BE in the form of an assignment repeated iteratively.

**Policy Iteration**

The iterative application of the BE when following a policy $\pi$ is called *iterative policy evaluation* (Algorithm 1) since it allows to compute the value functions of states and actions w.r.t. the policy $\pi$:

$$
\begin{aligned}
V_{t+1}(s) &= \mathbb{E}_\pi[R(s, a, s') + \gamma V_t(s'))] \\
&= \sum_a \pi(a|s) \sum_s P(s'|s, a)[R(s, a, s') + \gamma V_t(s')], \forall s \in \mathcal{S}.
\end{aligned}
\tag{2.10}
$$

It can be shown that the iterative application of the BE always converges to a single fixed point $V_\pi$.

Once the value functions have converged, it is interesting to see whether the current policy $\pi$ can be improved to make it closer to the optimal one $\pi^*$ or not. One way to do this consists in considering a state $s$ and an action $a \neq \pi(s)$ and computing

$$
\begin{aligned}
Q_\pi(s, a) &= \mathbb{E}[R(s, a, s') + \gamma V_\pi(s')] \\
&= \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V_\pi(s')], \forall s \in \mathcal{S}, a \in \mathcal{A}.
\end{aligned}
\tag{2.11}
$$

Whenever $Q_\pi(s, a) > Q_\pi(s, \pi(s))$ it is convenient to update the policy such that $\pi(s) = a$. This way policy is updated with the *greedy* policy $\pi(s) = \arg\max_a Q(s, a)$ is followed. This procedure is called *policy improvement*. The process of alternating steps of iterative policy evaluation and policy improvement leads to the estimation of the optimal value functions and is resumed in an algorithm called Policy Iteration (PI) (Algorithm 2).

**Value Iteration**

The alternation of policy evaluation and policy improvement is a drawback of PI that may slow down the learning process. Among other algorithms, the Value Iteration (VI) one (Algorithm 3) addresses this problem by stopping policy evaluation after only one

---

**Algorithm 2** Policy Iteration

---
1: **Initialize:** $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$
2: **repeat**
3:    **Iteration policy evaluation**
4:    **Policy improvement:**
5:    $policy\_stable \leftarrow true$
6:    **for all** $s \in \mathcal{S}$ **do**
7:      $old\_a \leftarrow \pi(s)$
8:      $\pi(s) \leftarrow arg\max_a \sum_{s'} P(s'|s,a)[r + \gamma V(s')]$
9:      If $old\_a \neq \pi(s)$, then $policy\_stable \leftarrow false$
10:   **end for**
11: **until** policy-stable

---

**Algorithm 3** Value Iteration

---
1: **Inputs:** a small threshold $\theta$ determining the accuracy of the estimate
2: **Initialize:** $V(s), \forall s \in \mathcal{S}$ arbitrarily, $V(s') = 0$ for all terminal states $s'$
3: **repeat**
4:    $\Delta \rightarrow 0$
5:    **for all** $s \in \mathcal{S}$ **do**
6:      $v \rightarrow V(s)$
7:      $V(s) \rightarrow \max_a P(s'|s,a)[r + \gamma V(s')]$
8:      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
9:    **end for**
10: **until** $\Delta < \theta$

---

update of each state value function. The update is different from the one in policy evaluation since it combines the policy evaluation steps and the policy improvement:

$$V_{t+1}(s) = \max_a \mathbb{E}[R(s, a, s') + \gamma V_t(s_{t+1})]$$
$$= \max_a \sum_s P(s'|s, a)[R(s, a, s') + \gamma V_t(s')], \forall s \in \mathcal{S}. \tag{2.12}$$

Desirably, VI maintains the properties of PI about the convergence to the fixed point corresponding to the optimal value functions.

As stated at the beginning of the section, the previous methods can be applied only when the full MDP model is known. However, in most real cases the full model is not available and the agent has to move into the environment to understand it.

### 2.3.2 Reinforcement Learning

The purpose of RL is to address the problem of solving an MDP in the cases where DP is not helpful, i.e. when the transition function $\mathcal{T}$ and the reward function $\mathcal{R}$ are not known. To achieve this, the agent should acquire the necessary knowledge from the transition samples obtained by moving through the environment. There are two main ways to use transition samples, resulting in two classes of RL algorithms:

- **model-based:** samples are used to learn a model of the environment that is subsequently used to learn the policy;

- **model-free:** samples are used to directly compute the policy, for instance by approximating the value function.

In both classes of methods, the RL problem can be addressed in two ways:

- **value-based:** the algorithm computes an estimate of the action-value function;

- **policy-based:** the algorithm computes an estimate of the value of the policies.

This taxonomy splits the RL literature into two sharply different fields where value-based techniques are mostly used for discrete MDPs whereas policy-based methods are mostly used in continuous MDPs, particularly in Robotics [47].

In all these classes of problems, the policy followed to collect samples plays a crucial role since the performance of the learning algorithm heavily depends on the balance between exploratory actions and exploitative actions, a fundamental issue of RL known as *exploration-exploitation dilemma*. The work done in this thesis focuses mainly on model-free value-based RL in MDP with a finite number of actions. Thus all the following analyses will be dedicated to this subfield.

**Exploration policies**   In a hypothetical setting in which the agent is given the knowledge of the optimal policy $\pi^*$, there is no need to explore new actions and it can simply follow the optimal policy to solve the MDP even without learning anything. Besides this very unlikely case, there are numerous situations where the agent does not know the environment it is moving in and the effect of its actions on the environment, then it needs to explore the environment to acquire the necessary knowledge to learn an optimal policy. The most trivial exploratory policy is called $\varepsilon$-greedy and is computed as

$$\pi(a|s) = \begin{cases} \dfrac{\varepsilon}{|\mathcal{A}|} + 1 - \varepsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} Q(s, a) \\ \dfrac{\varepsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{2.13}$$

Equation (2.13) results in the choice, at each step, between a greedy or random action according to the value of $\varepsilon$: the policy is completely random when $\varepsilon = 1$ and is completely greedy when $\varepsilon = 0$. Despite its simplicity, this policy is used in numerous works in the literature that allow to reach good results with minimum computational demand, but at the cost of a bad sample efficiency.

One of the drawbacks of $\varepsilon$-greedy is that it does not use the information available to the agent. The *Boltzmann* policy, also known as the *Softmax* policy, uses the current estimate of the action-value functions computing the probabilities for each action $a_i$ to be sampled as

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in \mathcal{A}} e^{Q(s,a')/\tau}}, \tag{2.14}$$

where $\tau$ is called *temperature* and controls the exploration ratio such as if $\tau \to 0$ the policy is greedy and if $\tau \to \infty$ the policy is random. This is an attempt to use the knowledge acquired by the agent to drive exploration in a smarter way than $\varepsilon$-greedy. A more recent strategy based on Boltzmann policy exploits the mellowmax (mm) operator to compute the Maximum Entropy mm policy [3]

$$\pi(a|s) = \frac{e^{\beta Q(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\beta Q(s,a')}}, \tag{2.15}$$

where $\beta$ is a value for which

$$\sum_{a \in \mathcal{A}} e^{\beta(Q(s,a) - mm_\omega Q(s))}(Q(s,a) - mm_\omega Q(s)) = 0 \qquad (2.16)$$

and

$$mm_\omega Q(s) = \frac{\log(\frac{\sum_{a \in \mathcal{A}} e^{\omega Q(s,a)}}{|\mathcal{A}|})}{\omega}. \qquad (2.17)$$

The mm operator (2.17) is a softmax operator with the desirable property of differentiability of the Boltzmann operator (2.14), but, differently from the Boltzmann, it is also a contractive operator, thus being an interesting operator for computing the target of the BE.

**Temporal Difference Learning**

Given a policy, RL is able to evaluate it from raw experience differently from DP, which needs to know the complete MDP model. A well-known class of methods to do this is called Monte Carlo (MC) [68] and consists of collecting samples for an entire episode and then updating the considered value functions using the discounted cumulative reward obtained during the run:

$$V(s) \leftarrow V(s) + \alpha[G - V(s)], \qquad (2.18)$$

where $G$ is the cumulative reward obtained from state $s$ till the end of the episode and $\alpha$ is the learning rate. The drawback of these techniques is that they need the episode to be finished before updating the value function estimates, thus slowing down the learning time.

To keep the desirable property of MC to be able to estimate value functions from raw experience without losing the good property of DP to be able to update the same estimates after each step (i.e. *bootstrapping*), Temporal-Difference (TD) methods have been studied in the past literature and are still among the most used methods in RL. The simplest TD method is known as TD(0) and updates the value function after each step with

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]. \qquad (2.19)$$

TD methods can be used for control if, instead of estimating the state-value function, the action-value function is estimated. This class of methods is split into two subclasses according to the criterion used to compute the target of the TD update:

- **on-policy:** following the policy that the agent is learning;

- **off-policy:** following another policy.

**SARSA** The SARSA algorithm is a well-known on-policy TD method for control. Being an on-policy control method, given a policy $\pi$ its purpose is to estimate the action-value function $Q_\pi(s,a)$ for each $(s,a)$ tuple via the update formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)], \qquad (2.20)$$

where $a' \sim \pi(s')$. Being a TD(0) variant for control, the same convergence properties of TD(0) apply for SARSA.

$Q$**-Learning**  The corresponding off-policy TD algorithm for control w.r.t. SARSA is $Q$-Learning (QL) [97]. Unlike SARSA, QL directly approximates the optimal action-value functions using a different policy to compute the target of the TD update. In particular, QL uses the greedy policy to compute the target, resulting in the update formula

$$Q(s, a) \leftarrow Q(s, a) + \alpha_t(s, a) \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right). \qquad (2.21)$$

According to the difference between on-policy and off-policy methods, SARSA and QL are two algorithms that may behave very differently in some problems where the agent can reach some catastrophic states. For instance, if the desired goal state is close to other undesirable catastrophic states, for some exploration policies SARSA will learn a policy that lets the agent stay away from the undesired states and reach the goal safely but slowly; while for the same exploration policies QL will learn a policy moving close to undesired states, but reaching the goal faster than SARSA. Thus, SARSA is more suitable for real applications, e.g. in Robotics, where the cost to reach catastrophic situations is very high and is better to be careful not to damage the physical system; on the contrary, QL is more suitable when the cost of reaching bad states is not high, e.g. simulations.

**Fitted $Q$-Iteration**  A well-known TD batch variant of $Q$-Learning is the Fitted $Q$-Iteration (FQI) algorithm [30]. The idea of FQI is to reformulate the RL problem as a sequence of supervised learning problems. Given a set of samples $\mathcal{D} = \{\langle s_i, a_i, r_i, s'_i \rangle\}_{1 \leq i \leq N}$ previously collected by the agent according to a given sampling strategy, at each iteration $t$, FQI builds an approximation $\hat{Q}$ of the optimal $Q$-function by fitting a regression model on a bootstrapped sample set

$$\mathcal{D}_t = \left\{ \langle (s_i, a_i), r_i + \gamma \max_{a'} \hat{Q}(s'_i, a') \rangle \right\}_{1 \leq i \leq N}. \qquad (2.22)$$

Under some conditions, the algorithm is proved to converge to the optimal $Q^*$ in a finite number of steps. FQI has been presented for the first time using extra-trees regression [36], but it has been proved to work well with other kinds of regression techniques, e.g. neural network in the Neural Fitted $Q$-Iteration (NFQI) algorithm [67] or Gaussian Process (GP) regression [66] in the Weighted Fitted $Q$-Iteration algorithm [25].

**Deep Reinforcement Learning**

The groundbreaking works that revived the research on RL in recent years are certainly the ones belonging to the DRL field which consists in the use of DL methodologies to address the RL problem in complex high-dimensional MDPs. Indeed, the very large state and action spaces together with the high number of features represent an obstacle to learning with shallow RL making them computationally impractical. The recent advent of powerful computational resources, in particular the use of Graphics Processing Units (GPUs) for training deep neural networks in DL, allowed to overcome this issue favoring the use of powerful deep function approximators in RL.

**Deep $Q$-Network**  The pioneering work that showed the potential of DRL and started the research about it is the Deep $Q$-Network (DQN) algorithm [55]. It consists in the
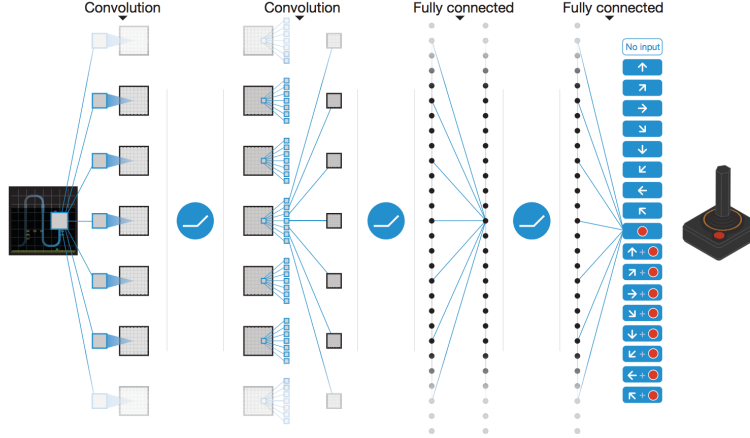
**Figure 2.3:** *A synthetic scheme of a deep neural network that takes as the input the screen of a video game and outputs the action to be performed in that state.*

use of a deep neural network and in the storing of a replay memory of past transitions $\langle s_t, a_t, r_t, s'_t \rangle$ to approximate the action-value function $\hat{Q}$ of the problem computing the target

$$y_i = \begin{cases} r_i & \text{if episode terminates at episode } i+1 \\ r_i + \gamma \max_{a'} \hat{Q}(s'_i, a') & \text{otherwise} \end{cases} \qquad (2.23)$$

and learning it in a supervised way.

The algorithm shows outstanding performance on the previously almost intractable problems of Atari games [11], a set of more than $50$ old but challenging video games, where DQN achieves state-of-the-art results even beating a human expert in some of them. The most remarkable quality of DQN is its relatively simple implementation which consists in using raw-pixel frame as input of a convolutional network and the application of simple gradient descent step to update the network. In this way, the complexity of the input, which made the solution of these problems intractable in the past, is addressed with the power of the deep convolutional network to extract a highly abstract representation significantly reducing its dimensionality and making it suitable for approximating the action-value function effectively. Despite its good properties, DQN has some issues about sample efficiency and the learning stability which are common problems among DRL techniques. Indeed, the number of samples required by DQN to learn is in the millions and corresponds to more than a month of human play; moreover, in some games, the algorithm shows very unstable performance even bringing to completely forgetting the good policy that it learned after many steps. For these reasons, without forgetting the promising research line that DQN revealed, the following literature has focused on many variants of DQN to improve some of the issues it has. For instance, Double DQN [40] has been proposed to improve the estimate of the action-value function, whereas Bootstrapped DQN [58] aims to improve exploration capabilities; both these algorithms will be considered and explained later in this thesis, respectively in Chapter 3 and 5.

# Part II

# Bellman Update

CHAPTER $3$

---

# Maximum Expected Value Estimation

---

The estimation of the Maximum Expected Value (MEV) of a set of random variables is required in several applications. Indeed, almost any decision-making process of acting involves the optimization of an expected utility function. For example, in everyday life decisions are usually made by considering the possible outcomes of each action based on partial information. While sometimes only the order of preference of these alternatives matters, many applications require an explicit estimation of the maximum utility. For instance, in RL the optimal policy can be found by taking, in each state, the action that attains the maximum expected cumulative reward. The optimal value of an action in a state, in turn, depends on the MEVs of the actions available in the reached states. Since the errors propagate through all state-action pairs, a bad MEV estimator negatively affects learning speed [90].

The most used approach to this estimation problem is the Maximum Estimator (ME) which simply takes the maximum estimated utility. As proved in [79], this estimate is positively biased and, if used in iterative algorithms, can increase the approximation error step-by-step [90]. More effective estimators have been proposed in recent years. The Double Estimator (DE) [91] approximates the maximum by splitting the sample set into two disjoint sample sets. One of these sets is used to pick the element with the maximum approximate value and its value is picked from the other set. This must also be done in the opposite way by switching the role of the two sets. Eventually, the average (or a convex combination) of the two values is considered. This approach has been proven to have a negative bias [91] which, in some applications, allows to overcome the problem of ME.

During my years of Ph.D. research, we analyzed this problem and proposed the Weighted Estimator (WE) [26] which approximates the MEV of a set of random variables by a sum of different values weighted by their probability of being the maximum.

WE can have both negative and positive bias, but its bias always stays in the range between the ME and DE biases.

All the mentioned approaches are limited to a finite set of random variables, thus, in a subsequent work, we extended the study to problems with an infinite set of random variables and proposed an extension of WE [25] to address them.

## 3.1 Problem definition

Given a finite set of $M \geq 2$ independent random variables $X = \{X_1, ..., X_M\}$, for each variable $X_i$ we denote with $f_i : \mathbb{R} \to \mathbb{R}$ its Probability Density Function (PDF), with $F_i : \mathbb{R} \to \mathbb{R}$ its Cumulative Density Function (CDF), with $\mu_i$ its mean, and with $\sigma_i^2$ its variance. The MEV $\mu_*(X)$ is defined as

$$\mu_*(X) = \max_i \mu_i = \max_i \int_{-\infty}^{+\infty} x f_i(x) \, \mathrm{d}x. \tag{3.1}$$

Unfortunately, $\mu_*(X)$ cannot be found analytically when the PDFs are unknown. However, it can be approximated using a given set of noisy samples $S = \{S_1, ..., S_N\}$ retrieved by the unknown distributions of each $X_i$ finding an accurate estimator $\hat{\mu}_*(S) \approx \mu_*(X)$. The random samples means $\hat{\mu}_1, ..., \hat{\mu}_N$ are unbiased estimators of the true means $\mu_1, ..., \mu_N$. Eventually, the PDF and CDF of $\hat{\mu}_i(S)$ are denoted by $\hat{f}_i^S$ and $\hat{F}_i^S$.

### 3.1.1 Related works

Several methods for estimating the MEV have been proposed in the literature. The most straightforward one is the ME which consists in approximating the MEV with the maximum of the sample means:

$$\hat{\mu}_*^{ME}(S) = \max_i \hat{\mu}_i(S) \approx \mu_*(X). \tag{3.2}$$

Unfortunately, as proved in [79], this estimator has a positive bias that may cause issues in applications of ME. For instance in the QL algorithm the overestimation of the state-action values due to the positive bias can cause an error that increases step after step. However, the expected value of the ME is different from the MEV in (3.1). Consider the CDF $\hat{F}_{\max}(x)$ of the ME $\max_i \hat{\mu}_i$ corresponding to the probability that ME is less than or equal to $x$. This probability is equal to the probability that all other estimates are less than or equal to $x$:

$$\hat{F}_{\max}(x) = P(\max_i \hat{\mu}_i \leq x) = \prod_{i=1}^{M} P(\hat{\mu}_i \leq x) = \prod_{i=1}^{M} \hat{F}_i(x).$$

Considering the PDF $\hat{f}_{\max}$, the expected value of the ME is $E\left[\hat{\mu}_*^{ME}\right] = E[\max_i \hat{\mu}_i] = \int_{-\infty}^{\infty} x \hat{f}_{\max}(x) dx$. This is equal to

$$E\left[\hat{\mu}_*^{ME}\right] = \int_{-\infty}^{\infty} x \frac{d}{dx} \prod_{j=1}^{M} \hat{F}_j(x) \, \mathrm{d}x = \sum_{i}^{M} \int_{-\infty}^{\infty} x \hat{f}_i(x) \prod_{i \neq j}^{M} \hat{F}_j(x) \, \mathrm{d}x.$$

The presence of $x$ in the integral correlates with the monotonically increasing product $\prod_{i\neq j}^{M} \hat{F}_j(x)$ and causes the positive bias.

To solve this overestimation problem, a method called DE has been proposed in [90] and theoretically analyzed in [91]. DE uses a sample set $S$ retrieved from the true unknown distribution like ME, but splits it into two disjoint subsets $S^A = \{S_1^A, ..., S_N^A\}$ and $S^B = \{S_1^B, ..., S_N^B\}$. If the sets are split in a proper way, for instance randomly, the sample means $\hat{\mu}_i^A$ and $\hat{\mu}_i^B$ are unbiased, like the means $\hat{\mu}_i$ in the case of the ME. An estimator $a^*$, such that $\hat{\mu}_{a^*}^A(X) = \max_i \hat{\mu}_i^A(X)$, is used to pick an estimator $\hat{\mu}_{a^*}^B$ that is an estimate for $\max_i E[\hat{\mu}_i^B]$ and for $\max_i E[X_i]$. Obviously, this can be done in the opposite way, using an estimator $b^*$ to retrieve the estimator value $\hat{\mu}_{b^*}^A$. DE takes the average of these two estimators. The expected value of DE can be found in the same way as for ME with

$$E\left[\hat{\mu}_*^{DE}\right] = \sum_i^M E\left[\hat{\mu}_i^B\right] \int_{-\infty}^{\infty} \hat{f}_i^A(x) \prod_{j\neq i}^{M} \hat{F}_j^A(x)\, \mathrm{d}x \qquad (3.3)$$

when using an estimator $a^*$ (the same holds by swapping A and B). This formula can be seen as a weighted sum of the expected values of the random variables where the weights are the probabilities of each variable being the maximum. Since these probabilities sum to one, the approximation given by DE results in a value less than or equal to the MEV. Even if the underestimation does not guarantee a better estimate than the ME, it may be helpful to avoid an incremental approximation error in some learning problems. For instance, Double $Q$-Learning (DQL) [90] is a variation of QL that exploits this technique to avoid the previously described issues due to overestimation. DQL has been tested in some very noisy environments and succeeded to find better policies than QL. Another remarkable application of DE is presented in [99] where it achieves better results than ME in a sponsored search auction problem.

## 3.2 **Weighted Estimator**

Differently from ME and DE that output the sample mean of the variable that is estimated to be the one with the largest expected value, the proposed WE estimates the MEV $\mu_*(X)$ computing a weighted average of all the sample means:

$$\hat{\mu}_*^{WE}(S) = \sum_{i=1}^{M} \hat{\mu}_i(S) w_i^S. \qquad (3.4)$$

Ideally, each weight $w_i^S$ should be the probability of $\hat{\mu}_i(S)$ being larger than all other sample means:

$$w_i^S = P\left(\hat{\mu}_i(S) = \max_j \hat{\mu}_j(S)\right).$$

If we knew the PDFs $\hat{f}_i^S$ for each $\hat{\mu}_i(S)$ we could compute the Distribution-Aware Weighted Estimator (DWE):

$$\hat{\mu}_*^{DWE}(S) = \sum_{i=1}^{M} \hat{\mu}_i(S) \int_{-\infty}^{+\infty} \hat{f}_i^S(x) \prod_{j\neq i} \hat{F}_j^S(x)\, \mathrm{d}x. \qquad (3.5)$$

We know that the sample mean $\hat{\mu}_i(S)$ is a random variable whose expected value is $\mu_i$ and whose variance is $\frac{\sigma_i^2}{|S_i|}$. Unfortunately, its PDF $\hat{f}_i^S$ depends on the PDF $f_i$ of variable $X_i$ which is assumed to be unknown. In particular, if $X_i$ is normally distributed, then, regardless of the sample size, the sampling distribution of its sample mean is normal too: $\hat{\mu}_i(S) \sim \mathcal{N}\left(\mu_i, \frac{\sigma_i^2}{|S_i|}\right)$. On the other hand, from the Central Limit Theorem (CLT), the sampling distribution $\hat{f}_i^S$ of the sample mean $\hat{\mu}_i(S)$ approaches the normal distribution as the number of samples $|S_i|$ increases, regardless of the distribution of $X_i$. Leveraging these considerations, we propose to approximate the distribution of the sample mean $\hat{\mu}_i(S)$ with a normal distribution, where we replace the (unknown) population mean and variance of variable $X_i$ with their (unbiased) sample estimates $\hat{\mu}_i(S)$ and $\hat{\sigma}_i(S)$:

$$\hat{f}_i^S \approx \tilde{f}_i^S = \mathcal{N}\left(\hat{\mu}_i(S), \frac{\hat{\sigma}_i^2(S)}{|S_i|}\right),$$

so that WE is computed as:

$$\hat{\mu}_*^{WE}(S) = \sum_{i=1}^{M} \hat{\mu}_i(S) \int_{-\infty}^{+\infty} \tilde{f}_i^S(x) \prod_{j \neq i} \tilde{F}_j^S(x) \, \mathrm{d}x. \tag{3.6}$$

It is worth noting that WE is consistent with $\mu_*(X)$. In fact, as the number of samples grows to infinity, each sample mean $\hat{\mu}_i$ converges to the related population mean $\mu_i$, and the variance of the normal distribution $\tilde{f}_i$ tends to zero, so that the weights of the variables with an expected value less than $\mu_*(X)$ go to zero, so that $\hat{\mu}_*^{WE} \rightarrow \mu_*(X)$.

### 3.2.1 Generalization to infinite random variables

As far as we know, the previous literature has focused only on the finite case and no approaches that natively handle continuous sets of random variables (e.g. without discretization) are available. Let us consider a continuous space of random variables $\mathcal{Z}$ equipped with some metric (e.g. a Polish space) and assume that the variables in $\mathcal{Z}$ have some spatial correlation. Here, we consider $\mathcal{Z}$ to be a closed interval in $\mathbb{R}$ and that every variable $z \in \mathcal{Z}$ has unknown mean $\mu_z$ and variance $\sigma_z^2$. Given a set of samples $S$, we assume to have an estimate $\hat{\mu}_z(S)$ of the expected value $\mu_z$ for any variable $z \in \mathcal{Z}$ (in the next section we will discuss the spatial assumption and explain how to obtain this estimate). As a result, the weighted sum of Equation (3.4) generalizes to an integral over space $\mathcal{Z}$:

$$\hat{\mu}_*^{\mathrm{WE}}(S) = \int_{\mathcal{Z}} \hat{\mu}_z(S) \, \mathfrak{f}_z^*(S) \mathrm{d}z, \tag{3.7}$$

where $\mathfrak{f}_z^*(S)$ is the probability density for $z$ of *being the variable with the largest mean*, which plays the same role of the weights used in (3.4). Given the distribution $f_{\hat{\mu}_z}^S$ of $\hat{\mu}_z(S)$, the computation of such density is similar to what is done in (3.6) for the computation of the weights $w_i^S$, with the major difference that in the continuous case we have to (ideally) consider a product of infinite cumulative distributions. Let us provide a tractable formulation of such density function:

$$\mathfrak{f}_z^*(S) = f\left(\hat{\mu}_z(S) = \sup_{y \in \mathcal{Z}} \hat{\mu}_y(S)\right)$$

$$= \int_{-\infty}^{\infty} f(\hat{\mu}_z(S) = x)\, P\left(\hat{\mu}_y(S) \le x,\ \forall y \in \mathcal{Z} \setminus \{z\}\right) dx$$

$$= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\, P\left(\bigwedge_{y \in \mathcal{Z} \setminus \{z\}} \hat{\mu}_y(S) \le x\right) dx \tag{3.8}$$

$$= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\, \frac{P\left(\bigwedge_{y \in \mathcal{Z}} \hat{\mu}_y(S) \le x\right)}{P\left(\hat{\mu}_z(S) \le x\right)} dx \tag{3.9}$$

$$= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\, \frac{\prod_{\mathcal{Z}} F_{\hat{\mu}_y}^S(x)^{dy}}{F_{\hat{\mu}_z}^S(x)} dx$$

where (3.8)-(3.9) follow from the independence assumption. The term $\prod_{\mathcal{Z}} F_{\hat{\mu}_y}^S(x)^{dy} = P\left(\bigwedge_{y \in \mathcal{Z}} \hat{\mu}_y(S) \le x\right)$ is the product integral [39] defined in the geometric calculus which is the generalization of the product operator to continuous supports and can be related to the classical calculus through the relation

$$\prod_{\mathcal{Z}} F_{\hat{\mu}_y}^S(x)^{dy} = \exp\left(\int_{\mathcal{Z}} \ln F_{\hat{\mu}_y}^S(x) dy\right). \tag{3.10}$$

**Spatially Correlated Variables**

The issues that remain to be addressed are:

1. the computation of the empirical mean $\hat{\mu}_z(S)$;

2. the computation of the density function $f_{\hat{\mu}_z}^S$ (for each random variable $z \in \mathcal{Z}$).

In order to face the former issue, we assumed the random variables to be spatially correlated. In this way, we can use any regression technique to approximate the empirical means and generalize over poorly or unobserved regions.

To address the second issue, we need to restrict the regression class to the methods for which it is possible to evaluate the uncertainty of the outcome. Let $g$ be a generic regressor whose predictions are the mean of a variable $z$ and the *confidence (variance) of the predicted mean* $\left(\text{i.e. } \hat{\mu}_z, \hat{\sigma}_{\hat{\mu}_z}^2 \leftarrow g(z)\right)$. As done in the discrete case, we exploit the CLT to approximate the distribution of the sample mean $f_{\hat{\mu}_z}^S$ with a normal distribution $\tilde{f}_{\hat{\mu}_z}^S = \mathcal{N}\left(\hat{\mu}_z, \hat{\sigma}_{\hat{\mu}_z}^2\right)$.

As a result, the WE for the continuous case can be computed as follows:

$$\hat{\mu}_*^{\text{WE}}(S) = \int_{\mathcal{Z}} \int_{-\infty}^{\infty} \frac{\hat{\mu}_z(S)\tilde{f}_{\hat{\mu}_z}^S(x)}{\tilde{F}_{\hat{\mu}_z}^S(x)} e^{\int_{\mathcal{Z}} \ln \tilde{F}_{\hat{\mu}_y}^S(x) dy} dx\, dz. \tag{3.11}$$

Since in the general case no closed-form solution exists for the above integrals, as in the finite case, the WE can be computed through numerical integration.

**Gaussian Process Regression**

While several regression techniques can be exploited (e.g. linear regression), the natural choice, in this case, is the GP regression since it provides both an estimate of the mean and variance of the process. Having a GP trained on a dataset of $N$ samples $\mathcal{D} = \{z_i, q_i\}_{i=1}^{N}$, where $q_i$ is a sample drawn from the distribution of $z_i$. Our objective is to predict the target $q_*$ of an input variable $z_*$ such that $q_* = f(z_*) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Given a kernel function $k$ used to measure the covariance between two points $(z_i, z_j)$ and an estimate of the noise variance $\sigma_n^2$, the GP approximation for a certain variable $z^*$ is $q_* \sim \mathcal{N}\left(\hat{\mu}_{z^*}, \hat{\sigma}_{\hat{\mu}_{z_*}}^2 + \sigma_n^2 I\right)$ where:

$$\hat{\mu}_{z_*} = \mathbf{k}_*^T \left(K + \sigma_n^2 I\right)^{-1} \mathbf{q}, \tag{3.12}$$

$$\hat{\sigma}_{\hat{\mu}_{z_*}}^2 = \mathrm{Cov}\left(\mu_{z_*}\right) = k(z_*, z_*) - \mathbf{k}_*^\top \left(K + \sigma_n^2 I\right)^{-1} \mathbf{k}_*,$$

and $\mathbf{k}_*$ is the column vector of covariances between $z_*$ and all the input points in $\mathcal{D}$ ($\mathbf{k}_*^{(i)} = K(z_i, z_*)$), $K$ is the covariance matrix computed over the training inputs ($K^{(ij)} = k(z_i, z_j)$), and $\mathbf{q}$ is the vector of training targets. Given the mean estimate in (3.12), the application of ME and DE is straightforward, while using WE requires to estimate also the *variance of the mean estimates*. The variance of the GP target $q_*$ is composed by the variance of the mean ($\hat{\sigma}_{\hat{\mu}_{z_*}}^2$) and the variance of the noise ($\sigma_n^2$) [66]. As a result, by only considering the mean contribute, we approximate the distribution of the sample mean by $\tilde{f}_{\hat{\mu}_z}^S = \mathcal{N}\left(\hat{\mu}_z, \hat{\sigma}_{\hat{\mu}_z}^2\right)$ as defined in Equations (3.12).

## 3.3 Analysis of Weighted Estimator

In this section, we theoretically analyze the estimation error of $\hat{\mu}_*^{WE}(S)$ in terms of bias and variance, comparing it with the results available for ME and DE. Although DWE cannot be used in practice, we include it in the following analysis since it provides an upper limit to the accuracy of WE.

### 3.3.1 Bias

We start with summarizing the main results about the bias of ME and DE reported in [91]. For what concerns the direction of the bias, ME is positively biased, while DE is negatively biased. If we look at absolute bias, there is no clear winner. For instance, when all random variables are distributed identically, DE is unbiased, while the same setting represents the worst case for ME. On the other hand, when the MEV is sufficiently larger than the expected values of the other variables, the absolute bias of ME may be significantly smaller than the one of DE. The bias of ME is bounded by:

$$\mathrm{Bias}\left(\hat{\mu}_*^{ME}\right) \leq \sqrt{\frac{M-1}{M} \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}}.$$

**Figure 3.1:** *Comparison of the bias of the different estimators varying the difference of the means*
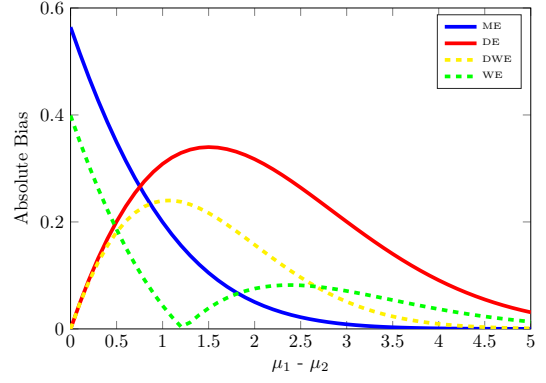


**Figure 3.2:** *Comparison of the absolute bias of the different estimators varying the difference of the means.*

For the bias of DE, [91] conjectures the following bound (which is proved for two variables):

$$\text{Bias}\left(\hat{\mu}_*^{DE}\right) \geq -\frac{1}{2}\left(\sqrt{\sum_{i=1}^{M}\frac{\sigma_i^2}{|S_i^A|}} + \sqrt{\sum_{i=1}^{M}\frac{\sigma_i^2}{|S_i^B|}}\right).$$

In the next theorem we provide a relationship between the bias of WE and the one of ME.

**Theorem 1.** *For any given set $X$ of $M$ random variables:*

$$\text{Bias}(\hat{\mu}_*^{WE}) \leq \text{Bias}(\hat{\mu}_*^{ME}) \leq \sqrt{\frac{M-1}{M}\sum_{i=1}^{M}\frac{\sigma_i^2}{|S_i|}}.$$

*Proof.* The proof of this Theorem follows directly from observing that $\hat{\mu}_*^{WE}$ is always smaller than $\hat{\mu}_*^{ME}$. In fact, the ME estimator can be seen as a weighted estimator that gives probability one to the variable associated to the largest sample mean $\hat{\mu}_i$, so that any other weighting cannot produce a larger value. □

This does not mean that the absolute bias of WE is necessarily smaller than the one of ME, since (as we will see later) the bias of WE can be also negative. In order to better characterize the bias of WE, we put it in relation with the bias of DE.

**Theorem 2.** *For any given set $X$ of $M$ random variables:*

$$\text{Bias}(\hat{\mu}_*^{WE}) \geq \text{Bias}(\hat{\mu}_*^{DE}).$$

*Proof.* If we compare the expected value of DE reported in Equation (3.3) with the value of the estimator WE in Equation (3.4), we can notice strong similarities. The main difference is that in DE the sample mean of variable $X_i$ and its probability of being the maximum are computed w.r.t. two independent set of samples, while in WE these two quantities are positively correlated. It follows that WE has a positive bias w.r.t. DE. □
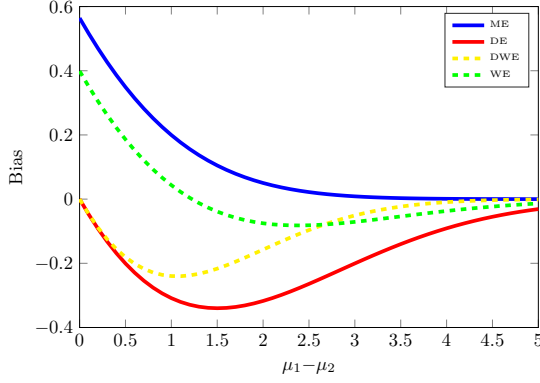
27
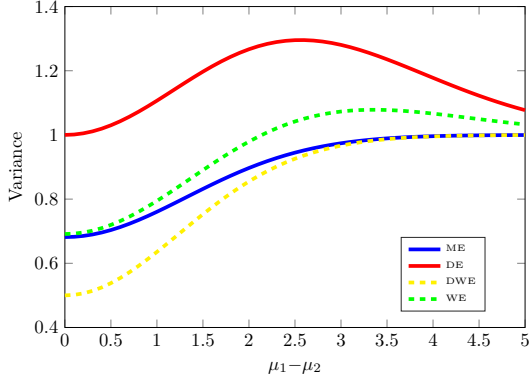
**Figure 3.3:** *Comparison of the variance of the different estimators varying the difference of the means.*
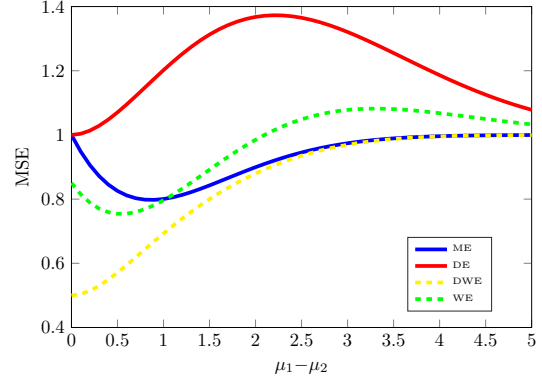


**Figure 3.4:** *Comparison of the Mean Squared Error (MSE) of the different estimators varying the difference of the means.*

**Example** In Figures 3.1 and 3.2 we visualize the bias of the different MEV estimators in a setting with two normally distributed random variables ($X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$) as a function of the difference in their expected values. Both variables have a variance equal to 10 ($\sigma_1^2 = \sigma_2^2 = 10$) and we assume to have 100 samples for each variable ($|S_1| = |S_2| = 100$). Figure 3.1 confirms the previous theoretical analysis: the bias of ME is always positive, while the biases of DWE and DE are always negative, with the latter always worse than the former. The bias of WE can be positive or negative according to the situation, but it always falls in the range identified by the biases of ME and DE. Looking at the absolute biases shown in Figure 3.2, we can notice that there is not clear winner. As previously mentioned, when the variables have the same mean, both DE and DWE are unbiased, while it represents the worst case for the bias of ME and WE. It follows that, when the difference between the two means is small (less than 0.5 in the example), DE suffers less absolute bias than ME and WE. For moderate differences of the means (between 0.5 and 1.8 in the example), WE has the lowest absolute bias, while ME is preferable for larger differences. Such results can be generalized as follows: DE suffers a small bias when there are several random variables that have expected values close (w.r.t. their variances) to the maximum one, while ME provides the best estimate when there is one random variable whose expected value is significantly larger (w.r.t. the variances) than all expected values of all other variables. In all other cases, WE is less biased.

### 3.3.2   Variance

We cannot evaluate the goodness of an estimator by analyzing only its bias. In fact, since the MSE of an estimator is the sum of its squared bias and its variance, we need to take into consideration also the latter.

[91] proved that both the variance of ME and the one of DE can be upper bounded with the sum of the variances of the sample means: $\mathrm{Var}\left(\hat{\mu}_*^{ME}\right) \leq \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}$ and $\mathrm{Var}\left(\hat{\mu}_*^{DE}\right) \leq \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}$. The next theorem shows that the same upper bound holds also for the variance of WE.

**Theorem 3.** *The variance of WE is upper bounded by*

$$\mathrm{Var}\left(\hat{\mu}_*^{WE}\right) \leq \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}.$$

*Proof.* Starting from the definition of WE 3.4, we can derive the bound to the variance as follows

$$\mathrm{Var}\left(\hat{\mu}_*^{WE}\right) = \mathrm{Var}\left(\sum_{i=1}^{M} \hat{\mu}_i(S)w_i^S\right)$$

$$\leq \mathrm{Var}\left(\sum_{i=1}^{M} \hat{\mu}_i(S)\right)$$

$$= \sum_{i=1}^{M} \mathrm{Var}\left(\hat{\mu}_i(S)\right),$$

where the inequality is a consequence of the maximization of each weight $w_i^S$ with one and the last equality comes from the independence of the sample means. $\square$

The bound in Theorem 3 is overly pessimistic; in fact, even if each weight $w_i^S$ is correlated to the other weights and to the sample mean $\hat{\mu}_i(S)$, their sum is equal to one. For sake of comparison, we upper bound the variance of DWE.

**Theorem 4.** *The variance of DWE is upper bounded by*

$$\mathrm{Var}\left(\hat{\mu}_*^{DWE}\right) \leq \max_{i \in 1,\ldots,M} \frac{\sigma_i^2}{|S_i|}.$$

*Proof.* Since the weights $w_i$ computed by DWE are not random variables, it follows

$$\mathrm{Var}\left(\hat{\mu}_*^{DWE}\right) = \mathrm{Var}\left(\sum_{i=1}^{M} \hat{\mu}_i(S)w_i\right)$$

$$= \sum_{i=1}^{M} w_i^2 \mathrm{Var}\left(\hat{\mu}_i(S)\right)$$

$$\leq \max_{i \in 1,\ldots,M} \frac{\sigma_i^2}{|S_i|},$$

where the inequality is motivated by $w_i^2 \leq 1, \ \forall i$. $\square$

**Example** As done for the bias, in Figure 3.3 we show the variance of the different estimators under the same settings described above. As the difference of the means of the two variables grows, the variance of all the estimators converges to the variance of the sample mean of the variable with the maximum expected value. DE is the estimator with the largest variance since its sample means are computed using half the number of samples w.r.t. the other estimators. WE exhibits a variance slightly larger than the one of ME, while, as expected, the variance of DWE is always the smallest.

Finally, in Figure 3.4 we show the MSE ($variance + bias^2$) of the different estimators. When the difference between the two means is less than one, WE suffers from a lower MSE than the other two estimators. On the other hand, ME is preferable when there is a variable with an expected value that is significantly larger than the other ones.

---

**Algorithm 4** Weighted $Q$-learning

---

1: Initialize $Q(s,a) = 0$, $\mu(s,a) = 0$, $\sigma(s,a) = \infty$ and $s$
2: **repeat**
3:     $a \leftarrow$ drawn from policy $\pi(\cdot|s)$ (e.g., $\varepsilon$-greedy)
4:     $s', r \leftarrow \text{MDP}(s,a)$
5:     $\tilde{f}_m^S \leftarrow \mathcal{N}(\mu(s,a_m), \sigma^2(s,a_m)) \quad \forall a_m \in \mathcal{A}$
6:     $w_m \leftarrow \int_{-\infty}^{+\infty} \tilde{f}_m^S(x) \prod_{k \neq m} \tilde{F}_k^S(x) \mathrm{d}x \quad \forall a_m \in \mathcal{A}$
7:     $W(s') \leftarrow \sum_{a_m \in \mathcal{A}} w_m Q(s', a_m)$
8:     $Q(s,a) \leftarrow Q(s,a) + \alpha(s,a)(r + \gamma W(s') - Q(s,a))$
9:     Update $\mu(s,a)$ and $\sigma(s,a)$ using tuple $\langle s,a,r \rangle$
10:    $s \leftarrow s'$
11: **until** terminal condition

---

## 3.4 Maximum Expected Value estimation in Reinforcement Learning

Among value-based methods, we consider online and offline algorithms that approximate the optimal action-value function $Q^*$ without the need for a model of the environment. We mostly consider MDPs with discrete action spaces, except for the batch algorithm based on WE that can also be extended to MDPs with continuous action spaces.

### 3.4.1 Online

It is demonstrated that since QL (see Section 2.3.2) is a stochastic approximation algorithm, under the assumption that every state-action pair is visited infinitely often and the step sequence satisfies certain conditions, $Q_k$ converges to $Q^*$ [97]. However, under particular conditions, such as wrong tuning of parameters and noisy environments, the $Q$-function may converge to $Q^*$ too slowly. One of the main reasons is that the QL uses the ME to estimate the current maximum $Q$-value of the next state $s'$. Since this estimator is positively biased, and since the error is propagated at each step, the $Q$-function can be wrongly estimated and the algorithm may fail. In the last years, different approaches have been proposed trying to overcome this issue [12, 51, 102]. In particular, the most successful one is the DQL algorithm [90] which replaces the ME used in QL with DE. The underestimation of the $Q$-function performed by DQL allows to learn a good policy in very noisy environments where QL fails.

**Weighted $Q$-Learning**

We propose to replace ME with WE in the Weighted $Q$-Learning (WQL) algorithm [26]. WQL maintains an estimate of the mean value of the $Q$-function and its variance to compute the weights of WE (Algorithm 4). While the mean value corresponds to the current estimate of the $Q$-function, the variance is not straightforward to be computed. Indeed, it is not simply the variance of the $Q$-function approximator, but it is the variance of the process consisting of an update formula with a variable learning rate. Considering this, it can be shown that the variance can be computed incrementally at each step $t$ with:

$$\sigma_t^2(s,a) \leftarrow n_t(s,a) \frac{(Q_{2_t}(s,a) - Q_t(s,a)^2)}{n_t(s,a) - 1} \omega_t(s,a),$$

---

**Algorithm 5** Weighted Fitted $Q$-Iteration (finite actions)

---

**Inputs:** dataset $\mathcal{D} = \{s_i, a_i, r_i, s'_i\}_{i=1}^{K}$, GP regressor $\widehat{Q}$, horizon $T \in \mathbb{N}$, discrete action space $\mathcal{A} = \{a_1, \ldots, a_M\}$
Train $\widehat{Q}_0^{\bar{a}}$ on $\mathcal{T}_0 = \{\langle s_i, r_i\rangle \text{ s.t. } a_i = \bar{a}\}$ ($\forall \bar{a} \in \mathcal{A}$)
**for** t=1 **to** T **do**
    **for** j=1 **to** K **do**
        **for** m=1 **to** M **do**
            $\hat{\mu}_m, \sigma^2_{\hat{\mu}_m} \leftarrow \widehat{Q}_{t-1}^{a_m}(s'_j)$ (evaluate GP)
            $\tilde{f}^S_{\hat{\mu}_m} \leftarrow \mathcal{N}(\hat{\mu}_m, \sigma^2_{\hat{\mu}_m})$ ($\tilde{F}^S_{\hat{\mu}_m}$ is the associated CDF)
            $w_{a_m} \leftarrow \int_{-\infty}^{+\infty} \tilde{f}^S_{\hat{\mu}_m}(x) \prod_{k \neq m} \tilde{F}^S_{\hat{\mu}_m}(x)\mathrm{d}x$
        **end for**
        $\mathcal{T}_t \leftarrow \mathcal{T}_t \cup \{(s_j, a_j), r_j + \gamma \sum_{a_m \in \mathcal{A}} w_{a_m} \mu_{a_m}\}$
    **end for**
    Train $\widehat{Q}_t^{\bar{a}}$ on $\mathcal{T}_t = \{\langle s_i, r_i\rangle \text{ s.t. } a_i = \bar{a}\}$ ($\forall \bar{a} \in \mathcal{A}$)
**end for**

---

**Algorithm 6** Weighted Fitted $Q$-Iteration$_\infty$ (continuous actions)

---

**Inputs:** dataset $\mathcal{D} = \{s_i, a_i, r_i, s'_i\}_{i=1}^{K}$, GP regressor $\widehat{Q}$, horizon $T \in \mathbb{N}$
Train $\widehat{Q}_0$ on $\mathcal{T}_0 = \{\langle (s_i, a_i), r_i\rangle\}$
**for** t=1 **to** T **do**
    **for** i=1 **to** K **do**
        $\hat{\mu}_z, \sigma^2_{\hat{\mu}_z} \leftarrow \widehat{Q}_{t-1}(s'_i, z)$ (evaluate GP)
        $\tilde{f}^S_{\hat{\mu}_z} \leftarrow \mathcal{N}(\hat{\mu}_z, \sigma^2_{\hat{\mu}_z})$ ($\tilde{F}^S_{\hat{\mu}_z}$ is the associated CDF)
        $v_i \leftarrow \int_{-\infty}^{\infty} \exp\left(\int_{\mathcal{Z}} \ln \tilde{F}^S_{\hat{\mu}_y}(x)\mathrm{d}y\right) \int_{\mathcal{Z}} \frac{\hat{\mu}_z(S)\tilde{f}^S_{\hat{\mu}_z}(x)}{\tilde{F}^S_{\hat{\mu}_z}(x)}\mathrm{d}z\mathrm{d}x$
        $\mathcal{T}_t \leftarrow \mathcal{T}_t \cup \{(s_i, a_i), r_i + \gamma v_i\}$
    **end for**
    Train $\widehat{Q}_t$ on $\mathcal{T}_t$
**end for**

---

where $Q_t(s, a)$ is the current $Q$-value of action $a$ in state $s$, $n_t(s, a)$ is the current number of updates of $Q(s, a)$ and

$$Q_{2_t}(s, a) = Q_{2_{t-1}}(s, a) + \frac{(r_t + \gamma W_t(s'))^2 - Q_{2_{t-1}}(s, a)}{n_t(s, a)},$$

$$\omega_t(s, a) \leftarrow (1 - \alpha_t(s, a))^2 \omega_{t-1}(s, a) + \alpha_t(s, a)^2$$

where $W_t(s')$ is the current value of WE in state $s'$.

### 3.4.2 Batch

The FQI update (see Section 2.3.2), similarly to the QL update, requires the computation of ME which causes the same overestimation problem of QL. Intuitively, the replacement of ME with DE or WE can help to solve this issue also in FQI.

**Weighted Fitted $Q$-Iteration**

The FQI variant that replaces ME with DE is called Double Fitted $Q$-Iteration (DFQI), and the variant that we propose using WE is called Weighted Fitted $Q$-Iteration (WFQI) [25]. WFQI uses GP regression to compute the mean $Q$-value and its variance in continuous state spaces (Algorithm 5). The interesting aspect of WFQI is that it can handle also infinite action spaces, as explained in Section 3.2.1 and shown in Algorithm 6. To the
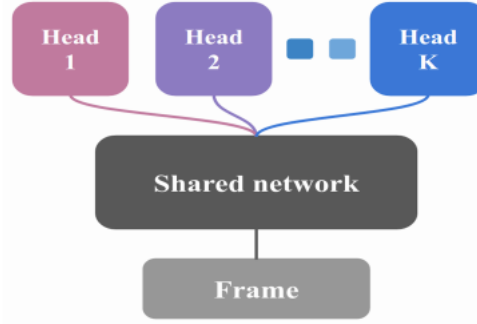
**Figure 3.5:** *Architecture of the neural network used in BDQN.*

best of our knowledge, this makes it the only value-based algorithm able to deal with infinite action spaces.

### 3.4.3 Deep Reinforcement Learning

We also analyzed the case of DRL where the overestimation problem caused by ME occurs also in the DQN algorithm resulting in critical loss of performance in some problems due to instability. The Double Deep $Q$-Network (DDQN) algorithm [40] replaces ME with DE and shows considerable improvements in performance and stability. We worked on the replacement of ME with WE even though in this case it is not straightforward due to the difficulties in computing the variance of the approximation. The GP regression, used in WFQI, is not commonly used in DRL and a deep neural network adapts better in important DRL problems, such as the well-known Atari domain [11]. The estimation of the mean and variance with a neural network is possible, but the computational complexity increases with the number of parameters such that makes it unfeasible in deep neural networks.

**Weighted Deep $Q$-Network**

We propose to estimate the variance of the approximation using an ensemble of target networks, following the neural network architecture proposed in another algorithm called Bootstrapped Deep $Q$-Network (BDQN) [58] shown in Figure 3.5. This work follows the DQN algorithm described in Section 2.3.2 with few, but important changes. The output of the neural network is split into $K$ heads that share the same first hidden layers. To perform bootstrap, a binary mask $w_1, \ldots, w_K \in \{0, 1\}$ is assigned to each sample to indicate the heads assigned to it. The binary mask is generated with a binomial distribution with probability $p$. The exploration policy of BDQN uniformly samples a head at the beginning of the episode, and follows the greedy policy learned by it. During the learning phase, the different heads are updated following the update rule of DDQN to avoid the overestimation problem.

We propose to use the architecture of BDQN and use WE to compute the target of the $K$-th head, while using the target provided by ME for the other $K-1$ heads. Moreover, we use only the $K$-th head trained with WE for exploration. In this way, the training of the $K-1$ heads does not differ from the one of BDQN, while the $K$-th

head is trained with WE exploiting the uncertainty on the action values provided by the ensemble. Given a dataset of transitions $\langle s_t, a_t, r_t, s_t' \rangle$, the resulting formulas to compute the target $y^k$ of each head $k$ are:

$$y_i^k = \begin{cases} r_i & \text{if episode terminates at episode } i+1 \\ r_i + \gamma \max_{a'} \hat{Q}^k(s_i', a') & \text{otherwise} \end{cases} \tag{3.13}$$

with $k \in \{1, \ldots, K-1\}$ and

$$y_i^K = \begin{cases} r_i & \text{if episode terminates at episode } i+1 \\ r_i + \gamma \sum_{j \in 1, \ldots, \#\mathcal{A}} w_i^j Q^K(s_i', a_j; \theta_K^-) & \text{otherwise} \end{cases} \tag{3.14}$$

for the $K$-th, head where $w_i^j$ is the percentage of the $K-1$ heads where action $a_j$ is the one with the highest action-value and $\theta_K^-$ are the parameters of the target network of the head $k$. To compute the action under the given policy, only the $K$-th head is used. The resulting algorithm is a slight change to the original BDQN that allows to use the WE in the DQN framework without differences in computational time and memory requirements w.r.t. BDQN. We call this algorithm Weighted Deep $Q$-Network (WDQN).

## 3.5 Empirical results

We evaluate the performance of ME, DE and WE in MAB and RL problems. We start by considering discrete state and action spaces, then we move onto the continuous ones and, eventually, to DRL problems.

### 3.5.1 Discrete states and action spaces

**Internet Ads**

We consider this MAB problem as formulated in [91]. The goal in this problem is to select the most convenient ad to show on a website among a set of $M$ possible ads, each one with an unknown expected return per visitor. It is assumed that each ad has the same return per click, therefore the best ad is the one with the maximum Click-Through Rate (CTR). Since the CTRs are unknown, they have to be estimated from data. In our setting, given $N$ visitors, each ad is shown the same number of times, so that we have $N/M$ samples to compute the sample CTR. It is desirable to obtain a quick and accurate estimate of the maximum CTR in order to effectively determine future investment strategies. We compare the results of ME, DE and WE in three different settings. We consider a default configuration where we have $N = 300000$ visitors, $M = 30$ ads and mean CTR uniformly sampled from the interval $[0.02, 0.05]$. In the first setting, we vary the number of visitors $N = \{30000, 60000, ..., 270000, 300000\}$, so that the number of impressions per ad ranges from $1000$ to $10000$. In the second setting, we vary the number of ads $M = \{10, 20, ..., 90, 100\}$ and the number of visitors is set to $N = 10000M$. In the last setting, we modify the interval of the mean CTR by changing the value of the upper limit with values in $\{0.02, 0.03, ..., 0.09, 0.1\}$, with the lower limit fixed at $0.02$.

(a) Increasing number of impressions.   (b) Increasing number of ads.   (c) Increasing value of maximum CTR.
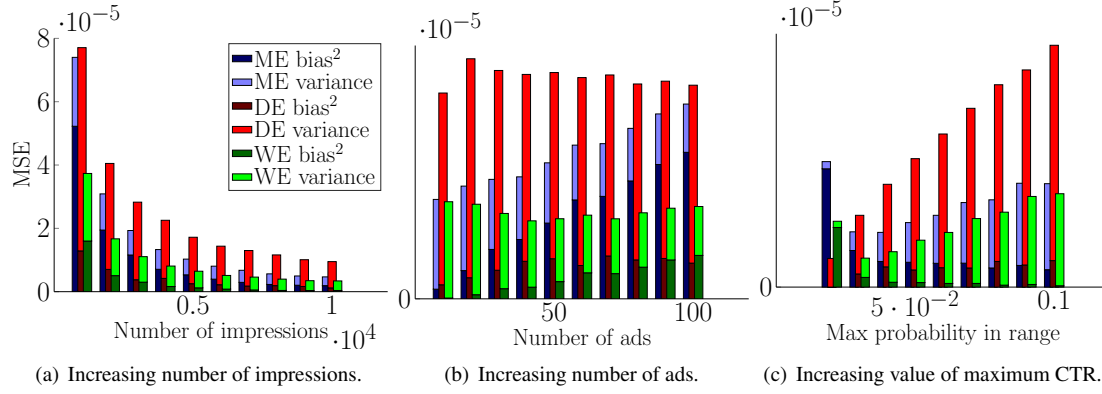
**Figure 3.6:** *MSE for each setting. Results are averaged over 2000 experiments.*

In Figure 3.6, we show the $MSE = bias^2 + variance$ for the three settings comparing the results obtained by each estimator. In the first setting (Figure 3.6(a)) the MSE reasonably decreases for all the estimators as the number of impressions increases and WE has the lowest MSE in all cases. Interestingly, the ME estimator has a very large bias in the leftmost case, showing that the ME estimator suffers large bias when the variances of the sample means are large due to the lack of samples (as shown also in Figure 3.2). Figure 3.6(b) shows that an increasing number of actions has a negative effect on ME and a positive effect on DE due to the fact that a larger number of ads implies a larger number of variables with a mean close to the MEV that represents a worst case for ME and a best case for DE. The MSE of WE is the lowest in all cases and does not seem to suffer the increasing number of actions. The same happens in Figure 3.6(c) when all the ads share the same CTR ($0.02$), where DE is the best. However, it starts to have a large variance as soon as the range of probabilities increases (Figure 3.3). The MSE of WE is the lowest MSE, but it gets similar to the MSE of ME as the range increases.

**Sponsored Search Auctions**

We consider this MAB problem as described in [99]. In this problem a search engine runs an auction to select the best ad to show from a pool of candidates with the goal of maximizing over a value that depends on the bid of each advertiser and its click probability. When an ad is clicked, the advertiser is charged from the search engine of a fee that depends on the bids $b$ of the advertisers and the CTRs $\rho$ of the ads. CTRs are generally unknown, therefore the search engine should use the data to estimate which is the best ad (i.e., the one that maximizes $b \cdot \rho$) and the payment in case of a click; reasonably, wrong estimates may significantly harm the revenue. On the other hand, advertisers have to decide the value of their bid $b_i$ according to the true values $v_i$ of a click. A desirable condition in auctions, called *incentive compatibility*, requires advertisers to maximize their utility by truthfully bidding $b_i = v_i$. Incentive compatibility may not occur when the estimates of the click probabilities are not accurate. We want to evaluate how the estimators favor the incentive compatibility. We measure the utility gain of advertiser 1, whose true value per click is $v_1 = 1$, for different bid $b_1$ values and competing with four other advertisers whose bids are $b_{-1} = \{0.9, 1, 2, 1\}$. The CTRs
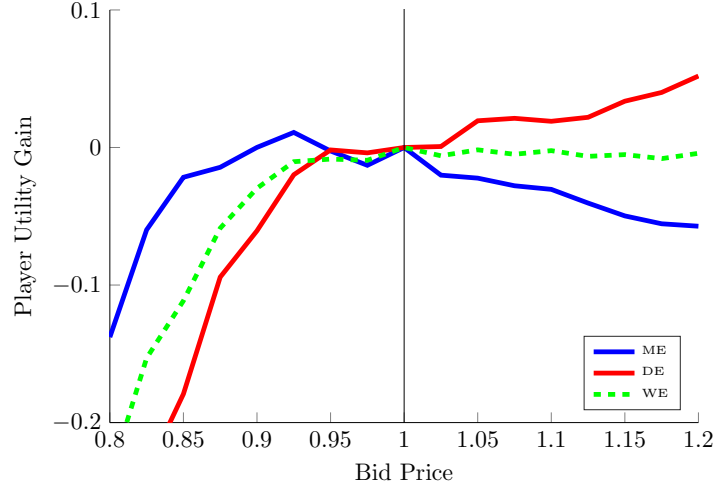
**Figure 3.7:** *Relative player 1 utility gain for different value of the bid defined as $\frac{utility(b)}{utility(v)} - 1$. Results are averaged over 2000 experiments.*

are: $\rho = \{0.15, 0.11, 0.1, 0.05, 0.01\}$. CTRs are estimated from data collected using the UCB1 algorithm [5] in a learning phase consisting of $10,000$ rounds of exploration (i.e. impressions), as done in [99].

Figure 3.7 shows the utility gain of advertiser 1 when using ME, DE and WE.[1] The true bid price is highlighted with a black vertical bar. ME results to be only one not able to achieve incentive compatibility, since the utility has positive values before the true bid price. On the contrary with DE and WE, the advertiser has no incentive to underbid, but there is an incentive to overbid using DE. Therefore, WE is the only estimator that succeeds to achieve incentive compatibility.

**Grid World**

This simple MDP consists of a $3 \times 3$ grid world where the start state in the lower-left cell and the goal state in the upper-right cell [90]. In this domain, we compare the three estimators together with the performance of an algorithm called Bias-Corrected QL, a modified version of QL which, assuming Gaussian rewards, corrects the positive bias of ME by subtracting to each $Q$-value a quantity that depends on the standard deviation of the reward and on the number of actions [51, 52]. Moreover, we also test WQL using a different policy, that we call *weighted policy*, which samples the action to be performed in a state from the probability distribution of the weights of WE. We use an $\varepsilon$-greedy policy with $\varepsilon = \frac{1}{\sqrt{n(s)}}$ where $n(s)$ is the number of times the state $s$ has been visited. The learning rate is $\alpha_t(s, a) = \frac{1}{n_t(s,a)^{0.8}}$ where $n_t(s, a)$ is the current number of updates of that action value and the discount factor is $\gamma = 0.95$. In DQL we use two learning rates $\alpha_t^A(s, a) = \frac{1}{n_t^A(s,a)^{0.8}}$ and $\alpha_t^B(s, a) = \frac{1}{n_t^B(s,a)^{0.8}}$ where $n_t^A(s, a)$ and $n_t^B(s, a)$ are respectively the number of times when table A and table B are updated. The reward function is considered in three different settings: Bernoulli, $-12$ or $10$ randomly at each step, Gaussian with mean $\mu = -1$ and standard deviation $\sigma = 5$,

---

[1]The debiasing algorithm proposed in [99] is a cross validation approach, but differs from the estimators considered in this work. It averages the values used for selection and the values used for estimation, thus being a hybrid of DE and ME.
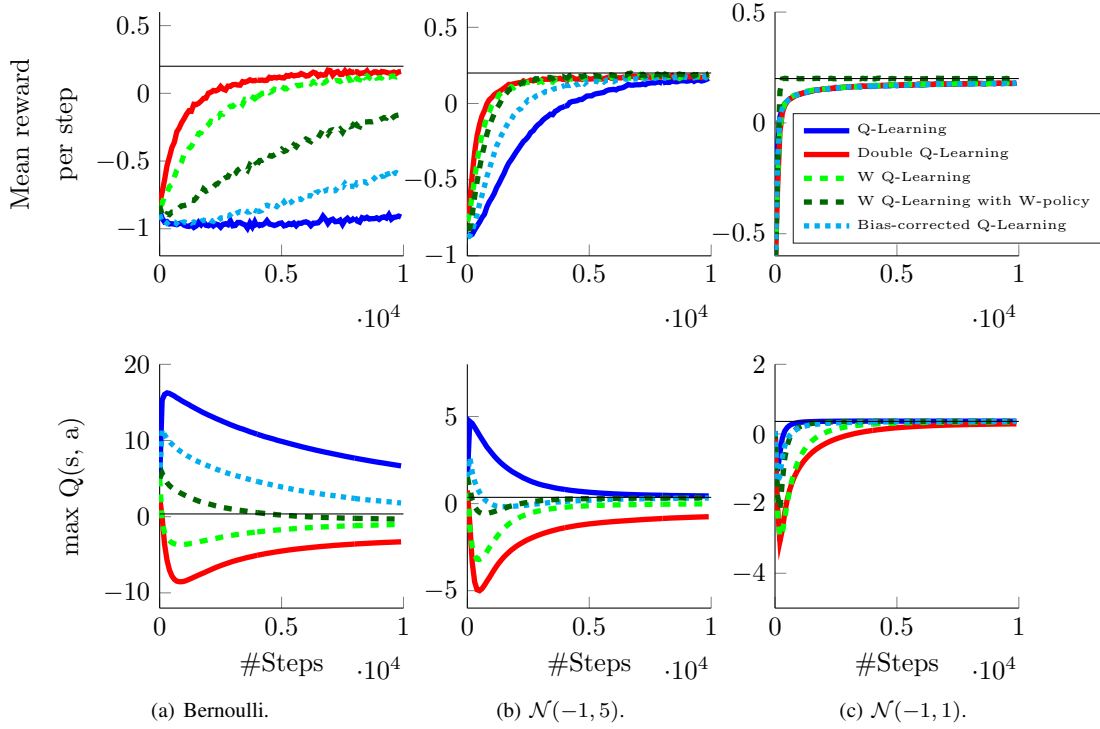
**Figure 3.8:** *Grid world results with the three reward functions averaged over 10000 experiments. Optimal policy is the black line.*

Gaussian with mean $\mu = -1$ and standard deviation $\sigma = 1$. Once in the goal state, each action ends the episode and returns a reward of $5$. The optimal policy ends the episode in five actions, therefore the optimal average reward per step is $0.2$. Moreover, the optimal value of the action maximizes the $Q$-value is $5\gamma^4 - \sum_{k=0}^{3} \gamma^k \approx 0.36$. In Figure 3.8, the top plots show the average reward per step obtained by each algorithm and the plots at the bottom show the estimate of the maximum state-action value at the starting state for each algorithm. Figures 3.8(a) and 3.8(b) show that, regardless of the bad approximation of the $Q$-function, the underestimation of DQL allows to learn the best policy faster than other algorithms in these noisy settings. Bias-Corrected QL estimates the $Q$-function better than DQL, but performs worse than the other algorithms, except for QL, when the variance is large. WQL shows much less bias than the other estimators in all settings; moreover, the use of the weighted policy generally reduces the bias of the estimation and achieves the best performance in the case with $\sigma = 1$ (see Figure 3.8(c)). These good results are explained considering that the weighted policy is able to reduce exploration faster than $\varepsilon$ greedy due to the exploiting of the good approximation of the $Q$-function computed by WQL. It is worth noting that WQL works well for both Gaussian and Bernoullian rewards, showing that WE is also effective with non-Gaussian distributions even if it uses a Gaussian approximation of $Q$-values,

**Forex**

We evaluate the performance of the three estimators in a more challenging discrete MDP. We build an MDP based on the Foreign Exchange Market (Forex), an environ-

(a) Training phase.

(b) Test phase.
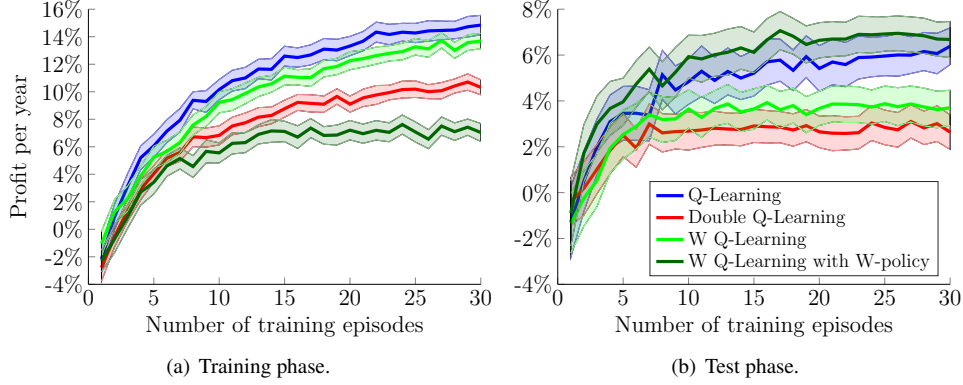
**Figure 3.9:** *Profit per year averaged over 100 experiments.*

ment with acknowledged hardly predictable dynamics that complicate the estimation of the $Q$-values and, therefore, of the expected profit. The MDP we build is a simplified version of the real Forex market. In our Forex MDP, the agent always enters in the market with 1\$ and each time the agent enters on long or short position a fixed spread value of 0.0002\$ is paid. The possible actions taken by the agent can be $-1$, $0$ or $1$, which mean respectively *'enter on a short position'*, *'close a position'* and *'enter on long position'*. The state space is composed of the suggestion (in terms of actions) provided by 7 common Forex indicators and the action chosen by the agent at the previous time step. The state space is $S = \{-1, 0, 1\}^8$ with $s_{i=1...7}(t) = \{-1, 0, 1\}$ and $s_8(t) = a(t-1)$. The action taken by the agent is $a(t) = \{-1, 0, 1\}$. The reward $r(t)$ is a function of the previous and current action chosen and of the difference between the current closing price $c(t)$ and the previous closing price $c(t-1)$:

$$r(t) = a(t-1)(c(t) - c(t-1)) + 0.5 * spread|a(t) - a(t-1)|.$$

The same algorithms used in the grid world domain, except for Bias-Corrected QL, were trained using the daily historical data of GBP/USD exchange rate from 09/22/1997 to 01/10/2005 and tested on data from 01/11/2005 to 05/27/08. During the training phase, we set the learning rate $\alpha(s, a) = \frac{1}{n(s,a)}$, discount factor $\gamma = 0.8$ and $\varepsilon = \frac{1}{\sqrt{n(s)}}$.

Figure 3.9 shows the annual profit, w.r.t. the number of training episodes, of the four algorithms during the training phase (Figure 3.9(a)) and during a test phase where the learned policy is run with a greedy policy (Figure 3.9(b)). In the training phase, an $\varepsilon$-greedy policy is used for QL and DQL, while WQL uses both the $\varepsilon$-greedy policy and the weighted policy. During the training phase, QL performs better than DQL and also than WQL. Interestingly, WQL with the weighted policy reaches the worst performance in the training phase, but reaches the best performance in the test phase. This happens because more exploration is induced by the weighted policy w.r.t. the $\varepsilon$-greedy policy, resulting in bad performance during the training phase, but also in better estimates of the $Q$-values. DQL performs worse than QL and WQL in both training and testing phase. The reason is that in many states there is a significantly better action than the others, which represents the case where ME gives the best results, and the case where DE suffers the most.

37
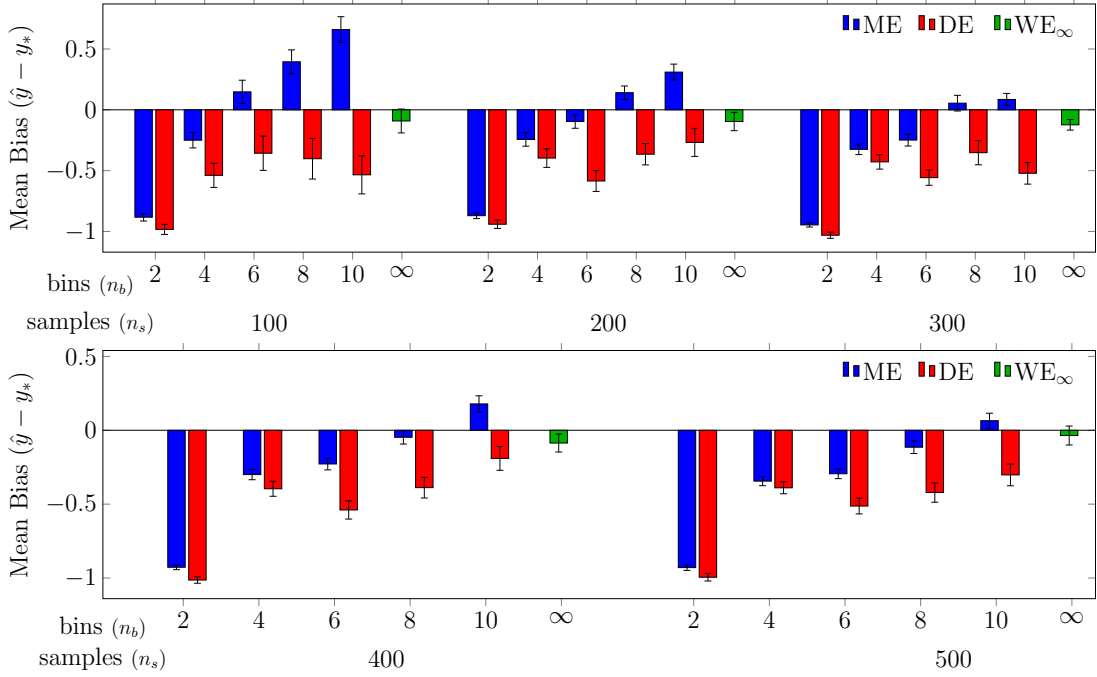
**Figure 3.10:** *Mean bias obtained by ME, DE and WE$_\infty$ with different sample sizes and bins (only for ME and DE).*

### 3.5.2 Continuous state spaces

**Pricing Problem**

This problem consists in estimating the MEV of the gross profit in a pricing problem. In this MAB problem we validate the WE with infinite random variables (WE$_\infty$) and we compare its performance against ME and DE whose support (actions) has been discretized. It is crucial to estimate the value of the gross profit accurately in order to evaluate, for example, an investment decision or to analyze the profitability of products. The support (action) space is bounded but continuous, and represents the price $p$ to show to the user ($p \in [0, 10]$). The reserve price $\tau$, which is the highest price that a buyer is willing to pay, is modeled as a mixture of 3 Gaussian distributions with mean $\mu = \{2, 4, 8\}$, covariances $\sigma^2 = \{0.01, 0.01, 0.09\}$ and weights $w = \{0.6, 0.1, 0.3\}$. The revenue function $r_\tau(p)$ is $p$ when $\tau \geq p$ and $0$ otherwise. The maximum revenue is about $2.17$. In each test the algorithms are fed with a set of samples $\mathcal{D} = \{\langle p_i, r_i \rangle\}_{i=1}^{n_s}$. Each sample is obtained by sampling a reserve price $\tau_i$ from the Gaussian mixture, a price $p_i$ from a uniform distribution over the price range, and by evaluating the revenue function ($r_i = r_{\tau_i}(p_i)$). Clearly, the reserve price is unknown to the algorithm. Results are averaged on $50$ runs in order and confidence intervals at $95\%$ are shown. WE exploits a Gaussian process with a squared exponential kernel to generalize over the continuous price (GP parameters are learned from $\mathcal{D}$), while ME and DE discretize the price space into $n_b$ evenly spaced bins. As shown in Figure 3.10, the number $n_b$ of optimal bins varies with the number $n_s$ of available samples. This means that, once the samples have been collected, ME and DE need an optimization phase to select the appropriate number of bins (not required by WE). WE is able to achieve the lowest or a
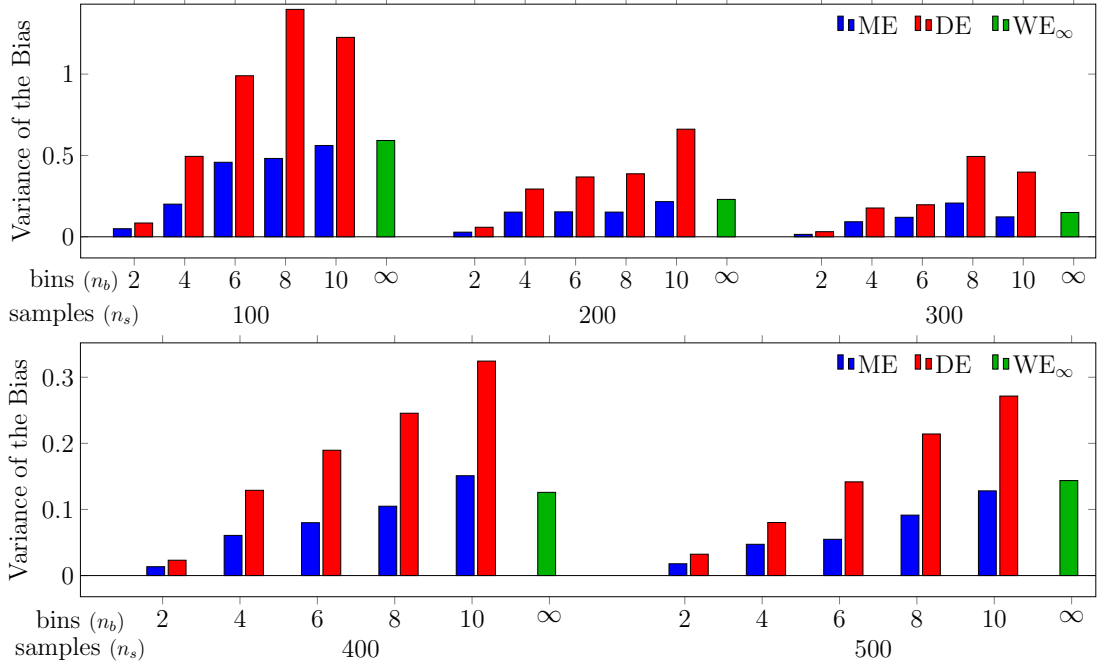
**Figure 3.11:** *Variance of the bias obtained by ME, DE and WE$_\infty$ with different sample sizes and bins.*

comparable level of bias with every batch dimension even through it exploits a sensibly wider action space (infinite). In fact, as shown by the experiments, the performance of ME and DE may degrade as the number of bins increases, i.e. the action space increases. This means that, if you want to be accurate, you cannot increase the number of bins arbitrarily (it is somehow counterintuitive). Additionally, Figure 3.11 shows that the higher complexity of WE has practically no impact on the variance of the estimate. The variance is always comparable to the one of the best configuration of WE and DE. Finally, several applications do not consider positive and negative bias to be the same, in particular, in iterative application positive bias can lead to large overestimates that have proven to be critical (e.g. in RL). This is not the case for pricing because this problem is not iterated. From Figure 3.10 we can see that ME is prone to provide positive bias, while WE bias is almost always the smallest or stays between ME and DE. The reason why the ME bias is not always positive, as stated by its theoretical property (for finite case), is due to the use of binning for the discretization of the continuous MAB. This discrete approximation introduces an additional (here negative) term to the bias.

**Swing-Up Pendulum**

A more complex scenario is represented by the continuous control problem analyzed in this section: the Swing-Up Pendulum with limited torque [27]. The aim of these experiments is to compare the newly proposed extensions of FQI (DFQI and WFQI) in a continuous state domain with both discrete and continuous actions. The peculiarity of this domain resides in the fact that the control with a limited torque ($u \in [-5, 5]$) makes the policy learning non-trivial. The continuous state space is $x = (\theta, \omega)$, where $\theta$ is the angle and $\omega$ is the angular velocity. An episode starts with $x_0 = (\theta_0, 0)$ where $\theta_0 \sim \mathcal{U}(-\pi, \pi)$, evolves according to the the dynamic system $\dot{\theta} = \omega$ and $ml^2\dot{\omega} =$

(a) Discrete actions.  (b) Continuous actions.
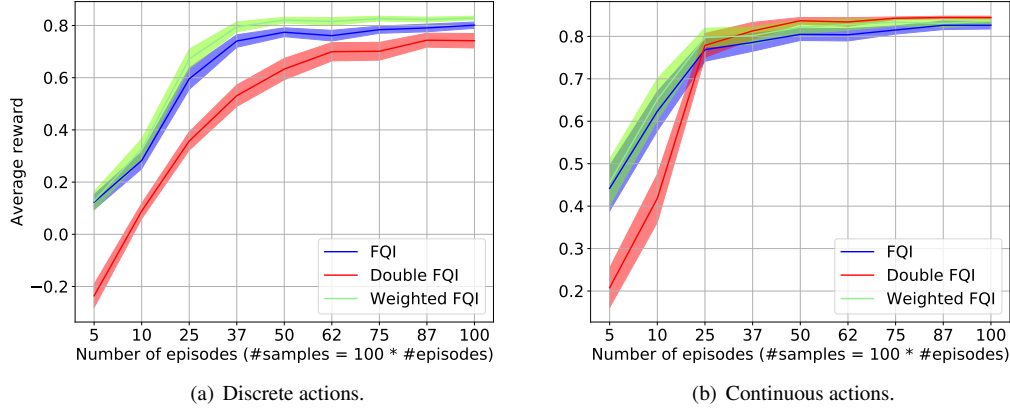
**Figure 3.12:** *Swing-Up Pendulum performance averaged over* 100 *experiments.*

$-\mu\omega + mgl\sin(\theta) + u$, and terminates after 100 steps. The physical parameters are mass $m = 1$, length $l = 1$, $g = 9.8$, step time $\tau_0 = 0.01$. The reward depends on the height of the pendulum: $r(x) = \cos(\theta)$. The problem is discounted with $\gamma = 0.9$. The GP uses a squared exponential kernel with an independent length scale for each input dimension (ARD SE). The hyperparameters are fitted on the samples and the input values are normalized between $[-1, 1]$. We collected training sets of different sizes using a random policy. The FQI horizon is 10 iterations. The final performance of the algorithm is the *average reward*, calculated starting from 36 different initial angles $\theta_0 = \{\frac{2\pi k}{36} | k = \{0, 1, \dots, 35\}\}$. We consider two settings of this problem: one with a discrete set of 11 uniformly spaced torque values in $[-5, 5]$ and another with a continuous action space. In the former setting we use a different GP for each action. Figure 3.12 shows that WFQI for discrete actions 5 and FQI are robust with respect to the number of episodes and WFQI reaches the highest average reward in each case (with statistical confidence obtained over 100 runs and 95% level). DFQI performance is reasonably poor with a few examples since it uses half of the training set to train each regressor. In the latter setting, the only algorithm able to directly handle the continuous space is the WFQI defined in Algorithm 6. The other algorithms use a GP with 100 actions to approximate the maximum.

# Exploiting uncertainty of the Bellman operator components to deal with highly stochastic problems

It is well known that a key issue of RL applications is the accuracy of the estimation of the action-value with a limited number of samples. Although most algorithms guarantee the convergence of the estimates of the action values to the optimal values with infinite samples, in practice the presence of stochastic components leads to slow learning. Actually, most real-world problems have significant sources of stochasticity: the environment may have stochastic transitions and this may affect the estimation of the effectiveness of an action; most of the times it is necessary to use stochastic policies to guarantee that all states are potentially visited infinitely many times; the reward function is often corrupted by noisy observations and, in other cases, the reward function is stochastic itself. Moreover, it usually happens that some deterministic environments are partially observable and, thus, are perceived by the agent as stochastic decision processes (e.g. Blackjack).

Since Monte Carlo estimates of action values are affected by the high variance of returns, the most successful RL algorithms are based on bootstrapping (e.g. *Q*-Learning [97]), which trades off the variance of the estimation with a consistent, but biased estimator. However, with a finite number of samples, the bias of the estimates could be significantly relevant when propagating the action-values to the next state and, recursively, to all other states. Recent works tried to deal with this issue, in particular by focusing on the estimation of the MEV, as discussed in Chapter 3.

However, an inaccurate estimation of the action value function does not always imply poor performance; indeed, most policies do not depend on the accuracy of the action-values, but, instead, rely on their ordering. Starting from these considerations,

we address this problem from another point of view. Indeed, we do not focus on the estimation of the MEV, but we care about weighing the new samples according to the uncertainty of current estimates. We consider that it is not sufficient to accurately estimate the MEV of the $Q$-function of the next state, but we also need to analyze how this information is propagated to other states. The problem of dealing with the uncertainty and the propagation of information across the states of the MDP has been addressed in many works on RL [56, 88]. Some of them consist in tuning the meta-parameters of the learning process according to uncertainty, e.g. the dynamic changes of the environment [46, 73, 100].

The interesting aspect of this approach is that any MEV estimator can be used and can be easily extended to an on-policy scenario. Since we believe that the choice of the estimator is not a relevant issue in our approach, we choose the ME as it is the most common.

## 4.1 Preliminaries

Recalling the optimal action-value function

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[ r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right], \tag{4.1}$$

as the expected value is a linear operator, we can always write (4.1) as:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[ r(s, a, s') \right] + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[ \max_{a'} Q^*(s', a') \right]. \tag{4.2}$$

Now, let us define two functions, $\tilde{R}$ and $\tilde{Q}$, as:

$$\tilde{R}(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[ r(s, a, s') \right],$$

$$\tilde{Q}(s, a) = \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[ \max_{a'} Q^*(s', a') \right]. \tag{4.3}$$

We can give an interpretation of these two functions: $\tilde{R}(s, a)$ is the expected immediate reward of action $a$ in state $s$; $\tilde{Q}(s, a)$ is the expected discounted return of the states reached after performing action $a$ in state $s$ (i.e. the expected gain of the reached state). We can now write the optimal value function as:

$$Q^*(s, a) = \tilde{R}(s, a) + \gamma \tilde{Q}(s, a). \tag{4.4}$$

Our approach shifts the focus of the RL task from finding a good estimator for the optimal action-value function, to the task of finding good estimators for $\tilde{R}$ and $\tilde{Q}$. The main motivation is that the sources of uncertainty of the two components of the action-value function are different: $\tilde{R}$ only depends on the transition and reward models, while $\tilde{Q}$ depends also on the optimal policy.

## 4.2 The proposed method

In the following section, we derive our method from (4.4). We propose the general schema, using the ME, and we show the relationships with the standard $Q$-Learning

update. We call this method $RQ$-Learning as it decomposes the TD-Error into a reward component and an action-value component.

### 4.2.1 Decomposition of the TD error

The standard $Q$-Learning algorithm, given the tuple $(s, a, r, s')$, computes the temporal difference error w.r.t. the current action-value estimates, and then updates such estimate proportionally to the error. The amount of correction in the direction of the new sample is measured by the learning rate: if the learning rate is $1$, the new sample substitutes the old estimate; if the learning rate is $0$, the new sample is discarded and the old estimate is kept unchanged. As shown in (4.4) the action-value function could be decomposed into two different components. Our method is based on the idea of giving separate estimates for these two components by computing the error w.r.t. each component of the action-value function, instead of computing the TD error:

$$\tilde{R}_{t+1}(s, a) \leftarrow \tilde{R}_t(s, a) + \alpha_t(R(s, a, s') - \tilde{R}_t(s, a)), \tag{4.5}$$

$$\tilde{Q}_{t+1}(s, a) \leftarrow \tilde{Q}_t(s, a) + \beta_t(\max_{a'} Q_t(s', a') - \tilde{Q}_t(s, a)). \tag{4.6}$$

Separating the two components of the value function can be useful, as the two components have inherently different sources of stochasticity. Moreover, the information stored in $\tilde{R}$ is local to each state-action pair and does not contain the uncertainty of estimating other states. Instead, the information stored in $\tilde{Q}$ depends only on the action-value function of the states that could be reached after performing action $a$ in state $s$, which depends, recursively, on the other action-value functions. It is clear that the propagation of uncertain values affects only the $\tilde{Q}$ component. As the actual action-value function is the sum of the two estimates, we can write an equivalent update for the Q-value:

$$\begin{aligned} Q_{t+1}(s, a) \leftarrow & \tilde{R}_t(s, a) + \alpha_t(R(s, a, s') - \tilde{R}_t(s, a)) \\ & + \gamma \left( \tilde{Q}_t(s, a) + \beta_t(\max_{a'} Q_t(s', a') - \tilde{Q}_t(s, a)) \right) \\ = & Q_t(s, a) + \alpha_t(R(s, a, s') - \tilde{R}_t(s, a)) \\ & + \gamma \beta_t(\max_{a'} Q_t(s', a') - \tilde{Q}_t(s, a)). \end{aligned} \tag{4.7}$$

### 4.2.2 Analysis of the decomposed update

Now, we discuss the relationship of our method with the standard temporal difference methods. Let $t$ be the learning step. As a first step of our analysis we can consider the simplest case $\alpha_t = \beta_t, \forall t$ by combining (4.7) and (4.4) we obtain:

$$Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha_t(R(s, a, s') + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a)). \tag{4.8}$$

That is the classical $Q$-Learning update.

We consider now the setting $\alpha_t \geq \beta_t > 0, \forall t$. Let $\beta_t = \delta_t \alpha_t$, we obtain:

$$
\begin{aligned}
Q_{t+1}(s,a) \leftarrow & Q_t(s,a) + \alpha_t(R(s,a,s') + \gamma\delta_t \max_{a'} Q_t(s',a') \\
& - (\tilde{R}_t(s,a) + \gamma\delta_t\tilde{Q}_t(s,a))) \\
= & Q_t(s,a) + \alpha_t(R(s,a,s') + \gamma_t' \max_{a'} Q_t(s',a') \\
& - (\tilde{R}_t(x,u) + \gamma_t'\tilde{Q}_t(s,a))) \\
= & Q_t(s,a) + \alpha_t((R(s,a,s') + \gamma_t' \max_{a'} Q_t(s',a')) \\
& - Q_t'(s,a))
\end{aligned}
\tag{4.9}
$$

with $\gamma_t' = \gamma\delta_t$. Notice that $Q_t'(s,a)$ is the Q-value at time step $t$, but computed with a different discount factor. If $\beta_t = \delta_t\alpha_t$, then we can see our method as a variable discount factor learning. If we consider that $\delta_t$ increases monotonically in the interval $[0,1]$, our method works by increasing the effective horizon at each step starting from trying to solve a greedy myopic problem and moving towards the real one. Similar approaches have been used in practice to solve infinite-horizon problems when the discount factor $\gamma \approx 1$ [8, 22, 33].

Finally, we can observe that, if the reward function and the transition model are deterministic, we can set $\alpha = 1$ and consider only $\beta_t = \delta_t$.

### 4.2.3 Variance dependent learning rate

To improve the quality of the estimation, we would like to weigh the error of each sample w.r.t. the current estimate depending on how much we trust the current value of our estimate. Thus, we propose a learning rate that depends on the variance of the current variable estimate.

First of all, we need to compute the variance of each estimator. To perform such computation, we must assume that the learning rate is independent of the data. This assumption is needed to have a closed form for the variance of the estimator, and it works well in practice. We will also assume that the samples $X_i$ are i.i.d., with mean $\mu$ and variance $\sigma^2$. Consider the general form of the estimator:

$$
\widetilde{X}_{n+1} = (1 - \alpha_t)\widetilde{X}_n + \alpha_t X_n.
\tag{4.10}
$$

We now compute the expected value and the variance of this estimator:

$$
\mathbb{E}\left[\widetilde{X}_{n+1}\right] = \mu \sum_i^n \alpha_i \prod_{j=i+1}^n (1 - \alpha_j)
\tag{4.11}
$$

$$
\text{Var}\left[\widetilde{X}_{n+1}\right] = \sigma^2 \sum_i^n \alpha_i^2 \prod_{j=i+1}^n (1 - \alpha_j)^2 = \sigma^2 \omega
\tag{4.12}
$$

with $\omega = \sum_i^n \alpha_i^2 \prod_{j=i+1}^n (1 - \alpha_j)^2$. Notice also that we can use the sample covariance $S_{n-1}$ of the random variable $X$ to estimate the real covariance $\sigma^2$. It is possible to compute incrementally way both the sample covariance, in the traditional way, and $\omega$:

$$
\omega_{n+1} = (1 - \alpha_n)^2 \omega_n + \alpha_n^2.
\tag{4.13}
$$

A weak assumption of this model is that the variables are identically distributed. While this may be true for the reward function, if the MDP is stationary, this is not true for the action-value function values whose distribution is affected by the policy and by the current estimates of the other states. However, a good approximation could be to consider data collected in a time window in which the distribution is approximately stationary. Using this approach, we can compute the variance of the process in a given time window by forgetting the old values that can lead to a biased estimation of the current window variance. Although this approach is not formally correct, as the derivation of the variance estimates makes the assumption of i.i.d. variables, this approximation leads to very good results in practice as we show in the empirical Section 4.3.

Finally, we can choose a learning rate that depends on the covariance. Let $\sigma_e^2(t)$ be an estimate of $\text{Var}\left[\widetilde{X}_t\right]$. We propose the following learning rate for each component of the action-value function:

$$\alpha_t = \frac{\sigma_e^2(t)}{\sigma_e^2(t) + \eta} \tag{4.14}$$

where $\eta$ is the amount of the estimator variance for which the learning rate is $0.5$ (i.e. when $\sigma_e = \eta$ the learning rate is $0.5$). It can be seen as a soft threshold to tune the rate of decrease in the learning rate w.r.t. the estimator variance.

If we consider the case $\beta_t = \alpha_t \delta_t$, then we have to use a different learning rate. As we want an increase in the discount factor faster than the decrease in the general learning rate, we can use an exponentially increasing learning rate for the delta parameter:

$$\delta_t = e^{\frac{\sigma_e^2}{\eta} \log \frac{1}{2}}, \tag{4.15}$$

where $\eta$ has the same interpretation as in (4.14) thanks to the $\log \frac{1}{2}$ factor.

### 4.2.4 Discussion on convergence

Convergence of the $Q$-Learning algorithm is guaranteed under some conditions, including some properties on the learning rates [31, 97]:

$$0 \leq \alpha_t < 1, \forall t$$

$$\lim_{N \to \infty} \sum_{t=1}^{N} \alpha_t = \infty,$$

$$\lim_{N \to \infty} \sum_{t=1}^{N} \alpha_t^2 < \infty. \tag{4.16}$$

To provide some preliminary results about the convergence of $RQ$-Learning and to motivate the choice of the formulas of learning rates $\alpha$ and $\delta$, we show that, under some assumptions, the proposed formulas (4.14) (4.15) satisfy (4.16). Suppose that the variance of the estimator $\sigma_e^2(t)$ is the variance of the sample mean. It is well known, by the central limit theorem, that the variance of the sample mean is $\sigma_\mu(t) = \frac{\sigma^2}{t}$, where $\sigma$ is the variance of the process that generates the samples. This assumption does not hold in general, as we can see from the formula of the learning rate; however, this is true for the sample mean, which is a special case of the generic estimator.

Now, just consider the learning rate proposed in (4.14), replacing the variance of the sample mean into the variance of the estimator:

$$\alpha_t = \frac{\sigma^2}{\sigma^2 + \eta t} \tag{4.17}$$

it can be easily shown that:

$$\lim_{N \to \infty} \sum_{t=1}^{N} \frac{\sigma^2}{\sigma^2 + \eta t} = \infty, \tag{4.18}$$

$$\lim_{N \to \infty} \sum_{t=1}^{N} \left( \frac{\sigma^2}{\sigma^2 + \eta t} \right)^2 < \infty. \tag{4.19}$$

Now, consider $\beta_t = \alpha_t \delta_t$. In this scenario, we propose an exponential learning rate. Let be $\alpha_t = \frac{1}{t}$ for simplicity, and consider the learning rate (4.15) to replace the sample mean into the variance of the estimator:

$$\beta(t) = \frac{1}{t} e^{\frac{\sigma^2}{\eta t} \log \frac{1}{2}}. \tag{4.20}$$

Then:

$$\lim_{N \to \infty} \sum_{t=1}^{N} \frac{1}{t} e^{\frac{\sigma^2}{\eta t} \log \frac{1}{2}} = \infty, \tag{4.21}$$

$$\lim_{N \to \infty} \sum_{t=1}^{N} \left( \frac{1}{t} e^{\frac{\sigma^2}{\eta t} \log \frac{1}{2}} \right)^2 < \infty. \tag{4.22}$$

This analysis considers only the estimate of the variance of the sample mean. The analysis of the generic variance estimator is more complex, but we conjecture that convergence could be guaranteed under some mild conditions on $\eta$ and $\sigma$.

## 4.3 Experimental results

We compare $RQ$-Learning against $Q$-Learning [97], Double $Q$-Learning [90], Weighted $Q$-Learning [26] and Speedy $Q$-Learning[1] [37] in three discrete MDPs. We choose this set of algorithms because we want to analyze the impact of the maximum expected value estimator in the learning process. Indeed, while [90] strongly suggests that a negatively biased estimator should be used to deal with stochastic MDPs, the empirical results, reported in the following, show that positively biased estimators are able to achieve better performance in highly stochastic problems as well. Instead, we show that the main point is to exploit the data in the best way, in particular in the estimation of the action-value, giving higher relevance to more recent samples. We conjecture that the trade-off between keeping the old estimate and updating it with the new one is the key issue in highly stochastic environments. Both $RQ$-Learning and Speedy $Q$-Learning exploit this idea; in particular, the latter uses an increasing learning rate on the

---

[1]In these experiments we consider the asynchronous version of Speedy $Q$-Learning.

**Figure 4.1:** *Mean reward per step (top) and maximum action-value estimate in the initial state (bottom) in the Noisy Grid World problem of all the other algorithms and of the best setting of RQ-Learning for this experiment. Results are averaged over 10000 experiments.*

difference between the target computed with the current estimation and the one computed with the previous estimation, while $RQ$-Learning weigh the update considering the uncertainty of the current estimation.

We analyze the performance of $RQ$-Learning with different choices of learning rates. All the other algorithms use a decaying learning rate $\alpha(s,a) = \frac{1}{n(s,a)^k}$ where $n(s,a)$ and $k$ are respectively the number of updates for each action $a$ in state $s$ and a coefficient to tune the rate of decay.[2]

### 4.3.1 Noisy Grid World

This environment, proposed in [90] and [26], consists of a $3 \times 3$ grid with the initial position in the lower-left cell and the goal state in the upper-right cell. Every action performed in a non-goal state obtains a reward $-12$ and $10$ with equal probability. In the goal state, each action obtains a reward of $5$ and terminates the episode. The discount factor is $\gamma = 0.95$. The policy is $\varepsilon$-greedy with $\varepsilon = \frac{1}{\sqrt{n(s)}}$, where $n(s)$ is the number of visits of the state $s$. The optimal average reward per step is $0.2$ and the maximum action-value function of the initial state is $5\gamma^4 - \sum_{k=0}^{3} \gamma^k \approx 0.36$.

Figure 4.1 shows the average reward per step and the approximation of the maximum action-value in the initial state computed by the other algorithms and $RQ$-Learning with $\alpha$ as used by the other algorithms, but with the separated variance-dependent learning rate $\beta$ for the action-value estimate using $\eta = 1$. Notice how the performance of $RQ$-Learning for the reward is the best one (except for $k = 0.8$ where all the algorithms

---

[2]Assuming that Double $Q$-Learning splits the action-value table in a table A and a table B, also the learning rate is split in $\alpha_A(s,a) = \frac{1}{n_A(s,a)^k}$ and $\alpha_B(s,a) = \frac{1}{n_B(s,a)^k}$, where $n_A(s,a)$ and $n_B(s,a)$ are the number of updates for each action $a$ in state $s$, respectively in table A and table B.
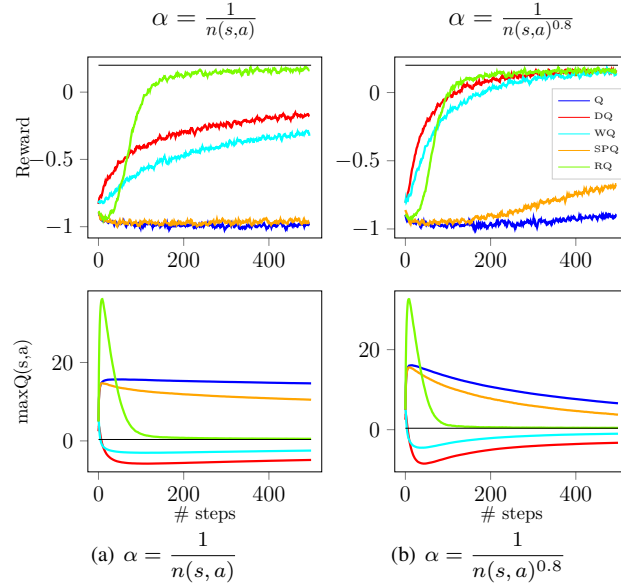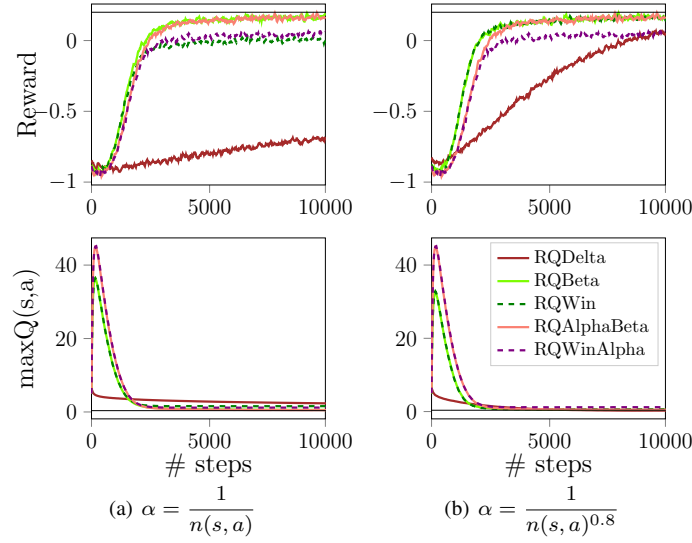
**Figure 4.2:** *Mean reward per step (top) and maximum action-value estimate in the initial state (bottom) in the Noisy Grid World problem of the best setting of RQ-Learning for this experiment together with other less effective setting of RQ-Learning. Results are averaged over* 10000 *experiments.*

perform similarly) and, also, how the estimate of the action-value is the best one. With $k = 1$, Speedy $Q$-Learning outperforms both Double $Q$-Learning and Weighted $Q$-Learning considering the average reward per step, even with a divergent estimate of the action-value function. This is an empirical evidence of our conjecture: the bias of the estimation is not necessarily correlated to performance. Moreover, as expected, the performance of $RQ$-Learning is not affected by the exponent used in the learning rate. The other algorithms achieve optimal performance only in the setting with the highest learning rate, confirming the advantage of giving more importance to the newer samples.

In Figure 4.2, we compare different variants of $RQ$-Learning: "RQBeta" is the same configuration used in Figure 4.1; "RQDelta" uses $\beta = \alpha\delta$ with $\eta = 1$; "RQWin" uses a windowed estimation of the variance with a window of length 50 and $\eta = 0.5$. "RQAlphaBeta" uses a variance-dependent learning rate also for $\alpha$ with $\eta = 100$ and $\beta$ with $\eta = 1$; "RQWinAlpha" is the same as configuration of the previous one, but uses a windowed $\beta$ with $\eta = 0.5$. Note that $\eta$ has a larger value in configurations without windowed variance estimation because such configurations are likely to over-estimate the current variance of the process. The "RQDelta" configurations result in a cautious learning that leads to very slow improvements, but avoids the overestimation of the action-value slowly converging to the optimal value. While "RQDelta" performance is not comparable to other configurations of $RQ$-Learning, it still outperforms $Q$-Learning. The other configurations perform similarly to the best one.

In Figure 4.3 we show the same information as Figure 4.1 and 4.2 for $RQ$-Learning with different values of $\eta$ to give a sense of how this parameter influences the learning process. It is clear how lower values of $\eta$ (i.e. a higher learning rate) let the results converge faster, confirming that a high value of the learning rate helps to speed up the learning process in highly stochastic environments like this.

(a) *RQ*-Learning  (b) Windowed *RQ*-Learning
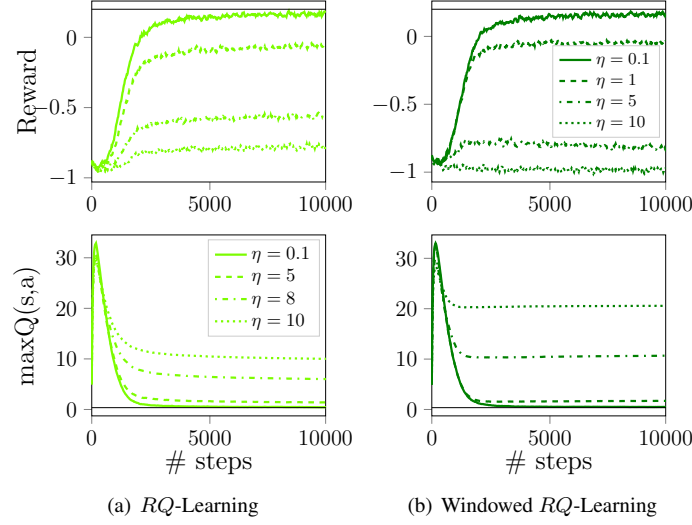
**Figure 4.3:** *Mean reward per step (top) and maximum action-value estimate in the initial state (bottom) in the Noisy Grid World problem of RQ-Learning (Figure 4.3(a)) and windowed RQ-Learning (Figure 4.3(b)) with different values of $\eta$ and $k = 0.8$. Results are averaged over $10000$ experiments.*
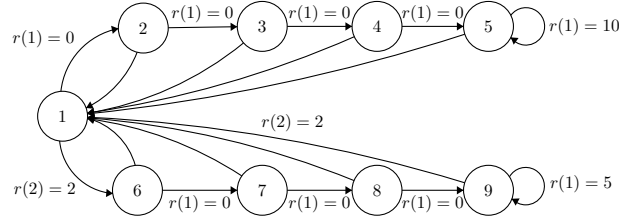


**Figure 4.4:** *Structure of the Double Chain problem.*

## 4.3.2 Double Chain

This is a problem proposed in [64], which consists of a Markov chain with two branches (Figure 4.4). In state $1$, action $1$ yields a reward of $0$ and moves the agent into state $2$; action $2$ yields a reward of $2$ and moves the agent into state $6$. In all other states, action $2$ moves the agent into state $1$ and returns a reward of $2$; action $1$ moves the agent into the next state of the chain by returning a reward of $0$. In states $5$ and $9$, action $1$ yields a reward of $10$ and $5$ respectively. In all states, each action has a probability of success of $0.8$ and, if the action fails, the agent remains in the current state and yields a reward of $0$. The discount factor is $\gamma = 0.9$. The optimal policy is to take action $1$ in state from $1$ to $5$ and action $2$ in the other states. $RQ$-Learning uses $\eta = 10$. In this experiment we focus on the estimation of the action-value function, therefore we use a fully random policy to explore the environment.

Figure 4.5 shows the estimate of the maximum action value in state $1$ and $5$. State $5$ is the state with the highest maximum action-value. $RQ$-Learning approaches the optimal value faster than the other algorithms in both configurations. However, in state $1$, only $RQ$-Learning with windowed variance estimation converges to the optimal value because the non-windowed approach suffers from variance overestimation due to the fact that the distribution of the next action-values changes during learning; this issue,
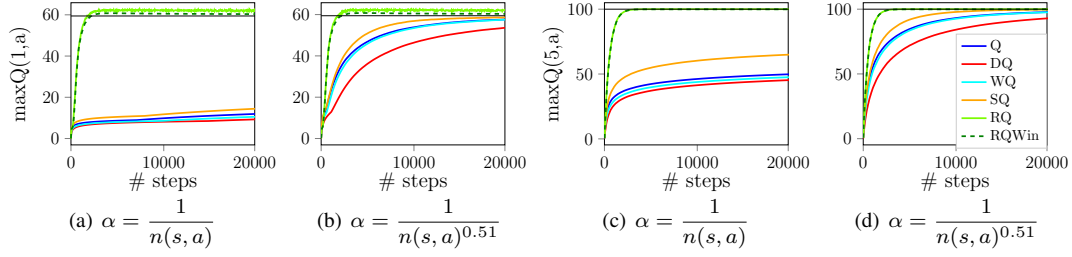
**Figure 4.5:** *Maximum action-value estimate in the Double Chain problem in state $1$ (4.5(a), 4.5(b)) and state $5$ (4.5(c), 4.5(d)). Results are averaged over $500$ experiments.*
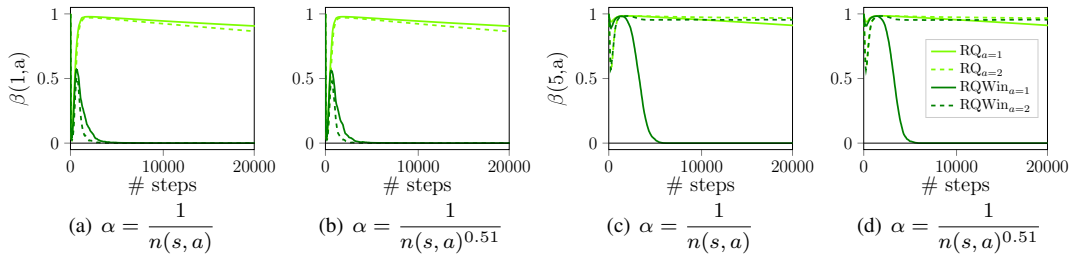


**Figure 4.6:** *Learning rate of the two actions in the Double Chain problem in state $1$ (4.6(a), 4.6(b)) and state $5$ (4.6(c), 4.6(d)) for RQ-Learning with and without windowed variance estimation. Results are averaged over $500$ experiments.*

together with the stochasticity of the transitions, causes the oscillation of the estimate and a slow convergence rate. This behavior is highlighted in Figure 4.6 where we show the learning rates of the action values in the considered states. While initially the learning rates are similar, the windowed learning rate converges to $0$, instead, in the non-windowed case, the learning rates decrease slowly. Note that in state $5$ the learning rate of action $2$ is almost stationary because of the complexity of the double chain structure. In this case, increasing $\eta$ can speedup the decreasing in the learning rate.

In this experiment, $RQ$-Learning not only approximates the value function very well, but is also able to converge to the optimal policy faster than the other algorithms. Figure 4.7 shows a comparison between $Q$-Learning and windowed $RQ$-Learning. We do not show the performance of the other algorithms since they behave similarly to $Q$-Learning. Notice that using $k = 1$, $Q$-Learning and the other algorithms, except from windowed $RQ$-Learning, are not able to converge to the optimal policy in state $9$. Indeed, the value of action $2$ in state $9$ is the most difficult to estimate, considering the structure of the MDP.

In this problem, where the only source of stochasticity is in the transition function, Double $Q$-Learning suffers the most. On the other hand, Speedy $Q$-Learning is still the best approach compared to others. This empirical result confirms our conjecture.

### 4.3.3 Grid World with Holes

This environment consists of a $5 \times 5$ grid with the initial position in the lower-left cell, there are $4$ actions and the transition model is deterministic, the goal position in the upper-right cell and four holes in the middle row in such a way that only the cell in
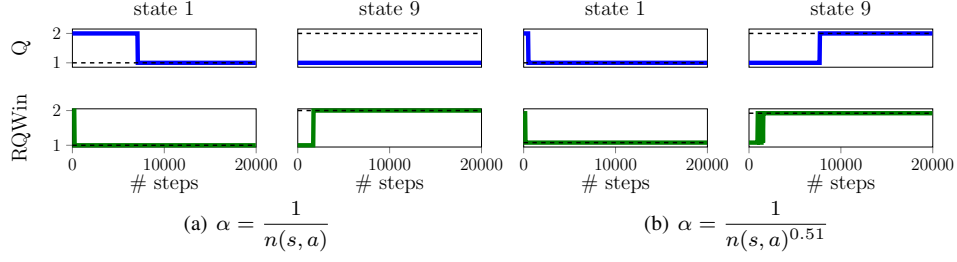
**Figure 4.7:** *Action with maximum value in the Double Chain problem in state $1$ and state $9$ for Q-Learning and windowed RQ-Learning.*
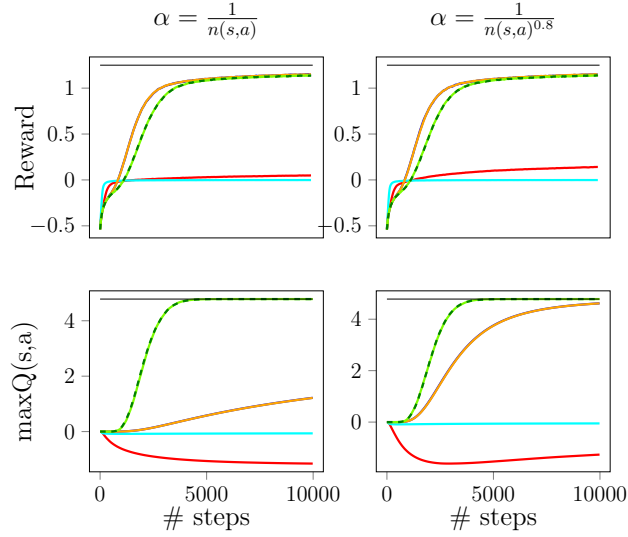


**Figure 4.8:** *Mean reward per step (top) and maximum action-value estimate in the Grid World with Holes problem in the initial state (bottom) of all the other algorithms and of the best setting of RQ-Learning for this experiment. Results are averaged over $10000$ experiments.*

the middle is walkable (Figure 4.9). The agent receives a reward of $0$ in all non-hole cells, a reward of $10$ when it reaches the goal state and a reward of $-10$ when it reaches a cell with a hole. The episode ends when the agent reaches a cell with a hole or the goal state. The discount factor is $\gamma = 0.9$. $RQ$-Learning uses $\eta = 1$. The learning rate settings are the same as the previous problem.

We consider this simple problem to highlight the limitations of pessimistic action-value estimates. In this MDP, the optimal policy consists in avoiding the hole cells stepping through the state in the middle. Notice that in this state the episode terminates with a negative reward with probability $\frac{\varepsilon}{2}$ due to the $\varepsilon$-greedy policy used for exploration, resulting in a very low value of the state, especially at the beginning of learning. Figure 4.8 shows that while $Q$-Learning, Speedy $Q$-Learning, and $RQ$-Learning behave similarly well, Double $Q$-Learning and Weighted $Q$-Learning obtain very poor results due to the pessimistic estimate of the value function of the state in the middle.
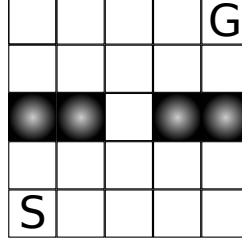
**Figure 4.9:** *Structure of the Grid World with Holes problem.*
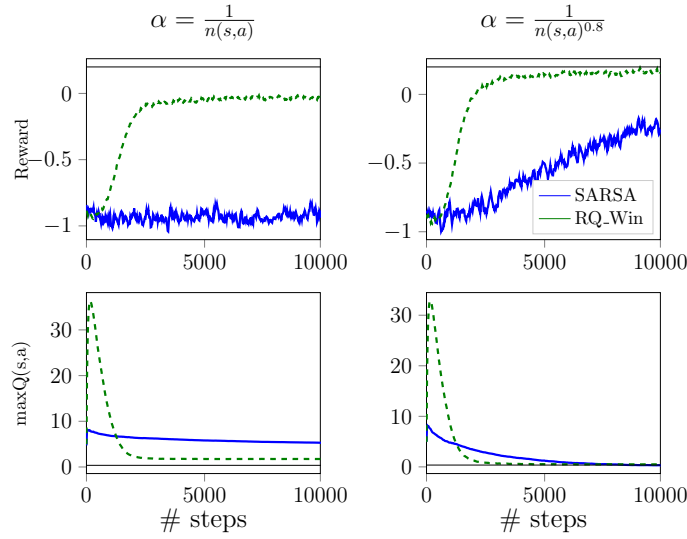


**Figure 4.10:** *Mean reward per step (top) and maximum action-value estimate in the Noisy Grid World problem in the initial state (bottom) of SARSA and of the on-policy windowed version of RQ-Learning for this experiment. Results are averaged over $1000$ experiments.*

### 4.3.4 On-policy learning

As we discussed in the previous sections, our approach can be used also in an on-policy setting. A simple on-policy version of our algorithm can be implemented by estimating the action-value function of the current policy in the same way of the SARSA algorithm, i.e. by using the action-value function of the next action. Let $a'$ be the next action sampled by the current policy in the current state, the on-policy update is:

$$\tilde{R}_{t+1}(s,a) \leftarrow \tilde{R}_t(s,a) + \alpha_t(R(s,a,s') - \tilde{R}_t(s,a)),$$
$$\tilde{Q}_{t+1}(s,a) \leftarrow \tilde{Q}_t(s,a) + \beta_t(Q_t(s',a') - \tilde{Q}_t(s,a)).$$

Figure 4.10 compares the windowed, on-policy version of $RQ$-Learning with the SARSA algorithm, in the Noisy Grid World environment. It is clear that our algorithm outperforms SARSA in this MDP. Since this is an on-policy setting, at each step the algorithm is estimating the action-value function of the current policy, not the optimal one. Indeed, by looking at the average reward per step, our approach estimates the current action-value function of the policy better than the SARSA algorithm, i.e. the estimated action-value function is coherent with the performance of the policy.

# Part III

# Uncertainty-Driven Exploration

# Thompson Sampling Based Algorithms for Exploration in Reinforcement Learning

A desirable property of sampling strategies is to have theoretical evidence of their efficiency in terms of the number of collected samples. Among others, the principle of Optimism in Face of Uncertainty (OFU) [49] has been well studied in the literature, where large evidence of its efficiency is given [43–45]. OFU states that actions with statistically uncertain values must be favored (e.g., through an exploration bonus) in the action-selection process compared to the more certain ones in order to improve knowledge of the environment. This optimistic sampling strategy accelerates the learning of action values and exploits them efficiently as the uncertainty about the environment is reduced, and thus the effect of optimism lessens. This sampling strategy has been firstly used in the context of MABs and is known as Thompson Sampling (TS) [89]. The idea of TS is to randomly choose an arm (i.e., an action) to be drawn according to its probability of being the optimal one. Interestingly, several recent works based on TS and showing promising results have been proposed [20, 38, 53, 74]. The use of TS in RL seems quite straightforward considering the actions of the MDP as the arms of the MABs; nevertheless, the presence of multiple states and dynamic transitions among them makes the estimation of uncertainty a challenging problem in this setting. Indeed, the estimation of the action-value uncertainty in each state of the MDP is the critical part of a sampling strategy based on optimism (e.g. TS) since computational and sample efficiency have to be taken into account for making optimism-based strategies feasible.

A widely used approach to modeling uncertainty is discussed in the literature about Intrinsic Motivation (IM) RL [21, 70] where the number of visits to a state is inversely related to the uncertainty of its value and is used to generate an intrinsic reward that guides the exploration [10, 85]. Other techniques to represent the interest of a state have

been proposed, for instance, relying on the amount of knowledge that a deep learning model has of a specific state [63] or considering the representativeness of a state w.r.t. the MDP [15]. Despite their effectiveness, these methods become more inefficient w.r.t. the dimensionality of the problem because of the complexity of representing the intrinsic reward.

Our work takes inspiration from the Bayesian RL framework in which a probability distribution is used to model the uncertainty over action value functions in each state. One of the first works that follow this approach is Bayesian Q-Learning [24] which proposes to estimate the action values by initializing them with a prior distribution and updating their posterior distribution each time a new sample is collected from the MDP. Our work is motivated by the fact that in the Bayesian approach the variance of the distribution can be used as a measure of the uncertainty of the action that allows to pursue OFU using TS.

We present several methodologies to efficiently compute the uncertainty of action values in online RL problems and to use the TS strategy. We propose two main techniques for accomplishing these tasks, one of which is suitable for low-dimensional problems and the other one specifically proposed for high-dimensional ones. We discuss the properties of these methods and consider MDPs in whichthe exploration is a critical aspect to empirically show how TS can outperform other sampling strategies in terms of performance and sample-complexity independently of the dimensionality of the problem.

## 5.1 Related work

The convergence of many RL algorithms is guaranteed when the exploration policy allows each state to be visited for an infinite number of times. Common sampling strategies, like $\varepsilon$-greedy or Boltzmann, guarantee convergence property, but they may suffer from unfeasible learning time. For this reason, finite-time bounds for convergence are a desirable property of exploration policies. The problem of finding finite-time bounds is addressed in several works that propose nearly-optimal sampling strategies with polynomial bounds for the convergence time. Most of these works are based on enhancing the probability of choosing actions that allow the agent to explore, i.e. to collect state-action samples never (or not sufficiently) seen before, thus pursuing the principle of OFU. Among them, UCRL [6] and UCRL2 [43] guarantee logarithmic regret bounds by applying the idea of using an upper bound on the estimate of action values and exploring with an optimistic policy, as done in the UCB1 [5] algorithm used in MABs. A similar approach is proposed in Delayed Q-Learning [80], where action values are initialized to high values, and an optimistic policy is used. Model-based methodologies based on the same reasonings have also been proposed, such as, among others, the $E^3$ [45] algorithm and the later R-MAX [16] algorithm. However, despite the desirable theoretical properties, these algorithms usually do not work well in practice, and, more importantly they do not scale well w.r.t. the dimensionality of the problem.

A different approach to pursuing OFU is to use a Bayesian framework. The literature on Bayesian RL offers a large number of works that can be split into model-based and value-based ones. Model-based approaches propose starting from a prior distribution over MDPs and computing the respective posterior distribution as newer samples

are collected from the MDP [23, 48]. Following this approach, Posterior Sampling for Reinforcement Learning (PSRL) [81] method consists in maintaining a probability distribution over MDPs, generating a hypothesis from it at the beginning of each episode and running the greedy policy w.r.t. the generated MDP. This method guarantees the convergence to an optimal policy for a stationary process with discrete states. An extension of this work showing a better upper bound on expected regret for PSRL is presented in [59]. Unfortunately, these algorithms do not scale well w.r.t. the complexity of the MDP, since building the model of them can be impractical. Value-based methods can overcome the drawbacks of model-based ones by using randomized value functions sampled by probability distributions. The first algorithm based on a Bayesian approach is Bayesian Q-Learning [24]. This algorithm starts from a prior over action values, instead of MDPs, updates the posterior distribution of these action values as new samples are collected from the MDP and run an optimistic policy, such as TS, w.r.t. them. The recent Weighted Q-Learning [26] algorithm, described in Section 3.4.1 follows a similar approach, but it simplifies the cumbersome action values update of Bayesian Q-Learning starting by the assumption that the probability distribution of action values can be approximated as a normal distribution according to the central limit theorem. Then, the variance of the normal distribution is used as the measure of uncertainty over the action values, and TS can be used. Other recent works that follow a value-based approach to the problem are Randomized Least Squares Value Iteration (RLSVI) [60] and its extension to the DRL setting BDQN [58], which we will consider later in this chapter.

## 5.2 Thompson Sampling in value-based Reinforcement Learning

We want to use TS as a sampling strategy in RL problems, i.e. to sample an action from each state according to its probability of being the optimal one. We consider an MDP with discrete actions and an unknown probability distribution associated with the action value function $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$. Let $f_i : \mathbb{R} \to \mathbb{R}$ be the PDF and $F_i : \mathbb{R} \to \mathbb{R}$ be the CDF of the action-value function $Q_i(s) = Q(s, a_i), \forall s \in \mathcal{S}$, and $\mu_i(s)$ and $\sigma_i^2(s)$ be the mean and the variance of these distributions respectively. The PDF and CDF of the approximate action-value function $\tilde{Q}_i(s)$ are denoted, respectively, as $\tilde{f}_i^s$ and $\tilde{F}_i^s$ with mean $\tilde{\mu}_i(s)$ and variance $\tilde{\sigma}_i^2(s)$. Thus, the probability that each action $a_i$ is the optimal in a given state $s$ can be computed as:

$$P\left(\tilde{Q}_i(s) = \max_j \tilde{Q}_j(s)\right) = P\left(\tilde{Q}_i(s) \geq \tilde{Q}_j(s), \forall j \neq i\right) = \int_{-\infty}^{+\infty} \tilde{f}_i^s(x) \prod_{j \neq i} \tilde{F}_j^s(x) dx.$$

(5.1)

Using the probabilities $P\left(\tilde{Q}_i(s) = \max_j \tilde{Q}_j(s)\right)$ computed in Equation 5.1, TS[1] can be done by sampling an action from the discrete density function resulting from them. Since we are interested only in selecting the action, we can avoid the expensive computation of the integral by merely sampling a value from each $\tilde{f}_i$ and selecting the action corresponding to the maximum sampled action value without loss of precision.

---

[1]Sometimes, in the RL literature, sampling strategies equivalent to TS are named in a different way (e.g. Q-value sampling [24]).

---

**Algorithm 7** Standard mean and variance update

---

1: **Inputs:** current mean $\overline{X}$, current variance $S^2$, new sample $x$, learning rate $\alpha$
2: $\Delta_{\overline{X}} \leftarrow x - \overline{X}$
3: $\overline{X} \leftarrow \overline{X} + \alpha\Delta_{\overline{X}}$
4: $S^2 \leftarrow (1-\alpha)(S^2 - \alpha\Delta_{\overline{X}}^2)$

---

**Algorithm 8** Mean and variance update using momentums

---

1: **Inputs:** current mean $\overline{X}$, current squared mean $\overline{X^2}$, current variance $S^2$, new sample $x$, learning rate $\alpha$
2: $\Delta_{\overline{X}} \leftarrow x - \overline{X}$
3: $\Delta_{\overline{X^2}} \leftarrow x^2 - \overline{X^2}$
4: $\overline{X} \leftarrow \overline{X} + \alpha\Delta_{\overline{X}}$
5: $\overline{X^2} \leftarrow \overline{X^2} + \alpha\Delta_{\overline{X^2}}$
6: $S^2 \leftarrow \overline{X^2} - \overline{X}^2$

---

The approximation of the unknown PDFs and CDFs is done relying on the CLT, as proposed in [26], which states that as the number of samples $N_i^s$ retrieved from the sampling distribution $\tilde{f}_i^s$ increases, $\tilde{f}_i^s$ approaches the normal distribution independently of the real distribution $f_i^s$, i.e., $\tilde{f}_i^s \approx \mathcal{N}\left(\tilde{\mu}_i(s), \frac{\tilde{\sigma}_i^2(s)}{N_i^s}\right)$, where $\tilde{\mu}_i(s)$ is the sample mean and $\tilde{\sigma}_i^2(s)$ is the sample variance.

## 5.3 Efficient uncertainty estimation

The use of the CLT explained in Section 5.2 allows representing the uncertainty of $\tilde{Q}_i(s) = \tilde{\mu}_i(s)$ with the sample variance $\frac{\tilde{\sigma}_i^2(s)}{N_i^s}$. The remaining problem is the computation of $\tilde{\sigma}_i^2(s)$ in an efficient and scalable way w.r.t. the complexity of the MDP. We propose two main methods for dealing with this problem. The former consists in updating the uncertainty online and is specifically proposed for use in discrete MDPs; nevertheless, we discuss its extension to continuous state spaces, which is however only practical for low-dimensional ones. The latter exploits the bootstrapping method to compute the variance, thus being more suitable for dealing with high-dimensional MDPs efficiently and, moreover, it does not need a Gaussian approximation of the action-value functions.

### 5.3.1 Online estimation

Firstly, we introduce the standard procedure to incrementally compute the update of the mean and the exponentially weighted update of the variance (Algorithm 7). An alternative method, which is more suitable for computational reasons but is slightly more biased than the previous one, is to estimate variance as the difference between the second order momentum and the square of the first order one (Algorithm 8). The previously described procedures can easily be integrated into the updating rule of many RL algorithms; in the following sections, we use SARSA as a reference case. Since we are interested in the variance of the estimator, we need to normalize the variance of the process $\sigma_{process}^2$ by the number of effective samples that is limited by the learning rate.

---

**Algorithm 9** SARSA with online variance update

---

1: **Inputs:** state $s$, action $a$, reward $r$, next state $s'$, learning rate $\alpha$, discount factor $\gamma$
2: $a' \leftarrow \pi(s')$
3: $\omega(s,a) \leftarrow (1-\alpha)\omega(s,a) + \alpha$
4: $\omega^2(s,a) \leftarrow (1-\alpha)^2\omega^2(s,a) + \alpha^2$
5: $n_{eff} \leftarrow \frac{\omega(s,a)^2}{\omega^2(s,a)}$
6: $target \leftarrow r + \gamma Q(s',a')$
7: $\sigma^2_{process}(s,a) \leftarrow (1-\alpha)\left(\sigma^2_{process}(s,a) + \alpha(target - Q(s,a))^2\right)$
8: $\sigma^2_{estimator}(s,a) \leftarrow \frac{\sigma^2_{process}(s,a)}{n_{eff}}$
9: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(target - Q(s,a)\right)$

---

Let $\alpha_i$ be the learning rate at the $i$-th update, we compute the effective sample size as:

$$n_{eff} = \frac{(\sum_{i=1}^n \omega_i)^2}{\sum_{i=1}^n \omega_i^2}, \qquad (5.2)$$

where $\omega_i = \alpha_i \prod_{k=i+1}^K (1 - \alpha_k)$ is the weight of the $i$-th sample and $K$ is the number of collected samples. Then, we derive the update rule for SARSA with variance estimation, as shown in Algorithm 9. In order to desirably boost exploration, the variance estimator $\sigma^2_{estimator}(s,a)$ can be replaced with an upper bound of it chosen among the ones provided in statistics.

1. In line with the Gaussian approximation, we recall the confidence interval on variance provided by the $\chi^2$ distribution:

$$\sigma^2_{upperbounded}(s,a) \leftarrow \frac{(N-1)\sigma^2_{estimator}(s,a)}{\chi^2_{\frac{\beta}{2},N-1}}, \qquad (5.3)$$

where $N = \lceil n_{eff} \rceil$ and $\beta$ is the significance level of the test. This upper bound may not be enough to induce enough exploration in MDPs with a sparse reward function, especially at the beginning of the learning process when the action value function is initialized to zero.

2. Another upper bound is given by the Hoeffding bound [42]. Let $x_1, ..., x_n$ be independent and identically distributed bounded random variables such that $x_i$ falls in the interval $[a,b]$, then for any $\varepsilon > 0$:

$$P(\overline{X} - E[\overline{X}] > \varepsilon) \leq e^{-\frac{2N\varepsilon^2}{(b-a)^2}}, \qquad (5.4)$$

where $\overline{X} = \frac{1}{N}\sum_{i=1}^N x_i$. From Equation 5.4 we can derive the bound on the error relative to a probability $(1-\delta)$ for the inequality to hold:

$$\varepsilon(N) = \sqrt{\frac{(b-a)^2 \log\frac{1}{\delta}}{2N}}. \qquad (5.5)$$

From these considerations, we can derive the variance estimation method shown in Algorithm 10 where $R_1$ and $R_2$ are the ranges of possible values of, respectively, $m_1$ and $m_2$ and where the upper bound is computed as the difference between the upper bound of the second order momentum and the lower bound of the

---

**Algorithm 10** SARSA with online variance update and Hoeffding upper bound

1: **Inputs:** state $s$, action $a$, reward $r$, next state $s'$, learning rate $\alpha$, discount factor $\gamma$,
2: $a' \leftarrow \pi(s')$
3: Update $n_{eff}(s, a)$ as in Algorithm 9
4: $target \leftarrow r + \gamma Q(s', a')$
5: $Q(s, a) \leftarrow Q(s, a) + \alpha(target - Q(s, a))$
6: $m_1(s, a) \leftarrow Q(s.a)$
7: $m_2(s, a) \leftarrow m_2(s, a) + \alpha \left(target^2 - m_2(s, a)\right)$
8: $\varepsilon_{m_1} = \sqrt{\frac{R_1^2 \log \frac{1}{\delta}}{2n_{eff}}}$
9: $\varepsilon_{m_2} = \sqrt{\frac{R_2^2 \log \frac{1}{\delta}}{2n_{eff}}}$
10: $\sigma_{process}^2(s, a) \leftarrow \min(m_2(s, a) + \varepsilon_{m_2}, R_{2_{max}}) - (\max(m_1(s, a) + \varepsilon_{m_1}, R_{1_{min}}))^2$
11: $\sigma_{estimator}^2(s, a) \leftarrow \frac{\sigma_{process}^2(s,a)}{n_{eff}}$

---

**Algorithm 11** SARSA with function approximation with variance update

1: **Inputs:** state $s$, action $a$, reward $r$, next state $s'$
2: $a' \leftarrow \pi(s')$
3: Update $n_{eff}(s, a)$ as in Algorithm 9
4: $Q_{target} \leftarrow r + \gamma Q[s', a'; \theta]$
5: $\Delta Q \leftarrow Q_{target} - Q[s, a; \theta]$
6: $\sigma_{process_{target}}^2 \leftarrow (1 - \alpha)\Delta Q^2$
7: $\Delta \sigma_{process}^2 \leftarrow \sigma_{process_{target}}^2 - \sigma_{process}^2[s, a; \theta']$
8: $\theta \leftarrow \theta + \alpha \Delta Q \nabla Q[s, a; \theta]$
9: $\theta' \leftarrow \theta' + \alpha \Delta \sigma_{process}^2 \nabla \sigma_{process}^2[s, a; \theta']$

---

first order one. In practice, this approach solves the problem of the initial lack of exploration, but since the range of the expected return is usually a large number, it delays too much the exploitation phase. One way to overcome this issue, is to sum the variance with a term, similar to the Hoeffding bound, which uses a parameter that allows to tune the amount of exploration in the early stages of learning:

$$\sigma_{upperbounded}^2 \leftarrow \sigma_{estimator}^2 + \frac{c}{\sqrt{n_{eff}}}. \tag{5.6}$$

3. The previous upper bound can be used in addition to the $\chi^2$ upper bound in order to boost the exploration when the number of samples is low and to have the benefits given by the $\chi^2$ upper bound during later stages of the learning process:

$$\sigma_{upperbounded}^2[s, a] \leftarrow \frac{(N - 1)\sigma_{estimator}^2[s, a]}{\chi_{\frac{\beta}{2}, N-1}^2} + \frac{c}{\sqrt{n_{eff}}}. \tag{5.7}$$

Previous techniques cannot be applied in MDPs with **continuous state spaces** since a tabular representation of variance is not suitable. In these problems, the variance of the process can be approximated using an adaptation of the procedures in Algorithm 7 and Algorithm 8. While the latter is immediate since it simply requires estimating the second-order momentum in addition to the first-order one, the former is a bit less straightforward, and we show the procedure in Algorithm 11. In this case, the problem that remains is the estimate of the effective sample size for each state-action tuple, but,

when the state space is finite and has only a few dimensions, discretization is the most straightforward alternative. For instance, this can be done using tilings features or, in case of large input spaces (e.g. pixel frames of a game), using an autoencoder as a hash-function [85].

### 5.3.2 Bootstrapping

We propose *bootstrapping* to compute the variance in MDPs with high-dimensional continuous state space where the previous method, based on the discretization of the state space, would be inefficient. Bootstrapping consists of training different regressors of the target function with datasets obtained through sampling with replacement from the original dataset [34]. The approximation of the variance of the estimate can be computed directly as the variance of the target function estimates provided by each regressor. To implement our idea, we consider the BDQN algorithm [58], a variant of the famous DQN algorithm [55], able to reach remarkable results in DRL problems. The main contribution of BDQN is the introduction of a policy based on bootstrapping and inspired by posterior sampling [59] which, at the beginning of each episode, randomly samples one of the regressor in the ensemble and follows the greedy policy derived by it.

**Thompson Sampling via Bootstrapping** We propose to use the framework provided by BDQN and replace its exploration policy with TS.[2] Since the bootstrapped network provides a distribution over action value functions, we claim that TS can be applied in this setting by picking, at each step and for each action, the action value from a randomly chosen head; then, executing the action whose action value is the highest among the sampled ones. BDQN consists in using an ensemble of $K \in \mathbb{N}_+$ action-value function approximators. This ensemble can be created either by using multiple neural networks, or by splitting the output of one single neural network into multiple heads, each of which represents the $\tilde{Q}_k$ estimate of the real action-value function $Q$. Bootstrapping is performed diversifying the heads with random initializations of their weights and assigning a randomly generated binary mask $m_1, \ldots, m_K \in [0, 1]$ to each sample to indicate which head should be trained with it. Moreover, the update of DDQN [93] is used instead of the standard DQN update to avoid the overestimation issue. Finally, the exploration policy consists of sampling a head at the beginning of each episode and following the greedy policy induced by it. On the other hand, evaluation is performed by ensemble voting.

The application of TS in BDQN simply consists in changing its exploration policy. Algorithm 12 shows the pseudocode of TS via bootstrapping.

**Bootstrapped Q-Learning** We slightly modify BDQN to adapt it to discrete MDPs. We use a tabular approximation of the action-value functions instead of using a deep neural network and update the action-value functions using the Double Q-Learning update rule (see Section 3.1.1). To perform bootstrapping, the action value functions are initialized randomly and masking is used as in BDQN.

---

[2]In their article, the authors of BDQN discuss a variant of the proposed policy which they call Thompson DQN. This simply modifies BDQN sampling a head at each step instead of at the beginning of each episode. However we think this does not correspond to an appropriate application of TS in this setting since the action values are not sampled from the distribution provided by the bootstrapped network.

---

**Algorithm 12** Bootstrapped DQN with Thompson Sampling

---

1: **Inputs:** action value function network $Q$ with $K$ outputs $Q_{k=1}^K$. Masking distribution $M$. Replay Buffer $B$ storing experience for training.
2: **repeat**
3:     Obtain state $s_t$ from the environment
4:     Sample action values $a_t^i$ from a randomly chosen head $Q_k$ for each action $i$
5:     Perform action $a_t$ which is the action $a_t^i$ with the largest value among the sampled ones, observe reward $r_t$ and reached state $s_{t+1}$
6:     Sample bootstrap mask $m_t \sim M$
7:     Add $(s_t, a_t, r_t, s_{t+1}, m_t)$ to replay buffer $B$
8: **until** End condition is satisfied

---

## 5.4 Experiments

Our focus on the empirical evaluation of TS over other exploration strategies is twofold since we want to show how TS can improve and/or stabilize learning in MDPs where exploration is a key issue, but we also want to evince how it can robustly deal with generic MDPs.

### 5.4.1 Discrete state space

**Taxi**

**Details** The **Taxi** problem consists of a discrete MDP, similar to a grid world, where the agent has to collect passengers before reaching the goal position. The need for exploration is intuitively critical in this problem considering the unknown location of the passengers and the sparse reward function. In particular, the balancing between exploitation-exploration is essential to avoid both suboptimal exploitive policies and slow exploratory policies. We use this problem as a small, but significant, experiment to evaluate both the online variance estimation and the bootstrapped method. We use the same configuration as the Taxi problem described in [4]. The Taxi grid is shown in Figure 5.1(b) where S is the initial position of the agent, P is a passenger and G is the goal position. The actions are four: *north*, *south*, *west*, and *east*. There is a probability of $0.1$ that an action fails, in this case the agent moves to one of the perpendicular directions w.r.t. the one selected. An action moving the agent towards a wall, results in no actions. The reward is always $0$, except when stepping into the goal position where it depends on the number of collected passengers: $0$ for zero, $1$ for one, $3$ for two and $15$ for three passengers. Since not only the number of passengers matters, but also which passengers have been collected, there are $264$ states in this problem. The discount factor is $\gamma = 0.99$.

**Setting** For the results on the left of Figure 5.1(a), we use $\varepsilon = 0.3$ for $\varepsilon$-greedy, temperature $\beta = 0.5$ for Boltzmann and $\omega = 1.15$ for mellowmax; the ones on the right are obtained with $\varepsilon = 0.3$ for $\varepsilon$-greedy, temperature $\beta = 1.15$ for Boltzmann and $\omega = 4.5$ for mellowmax. The learning rate is $\alpha = \frac{1}{n(s,a)^{0.3}}$ where $n(s,a)$ is the number of updates of the action value of action $a$ in state $s$. An evaluation run consists of $200$ steps. We average the results over $64$ different seeds.

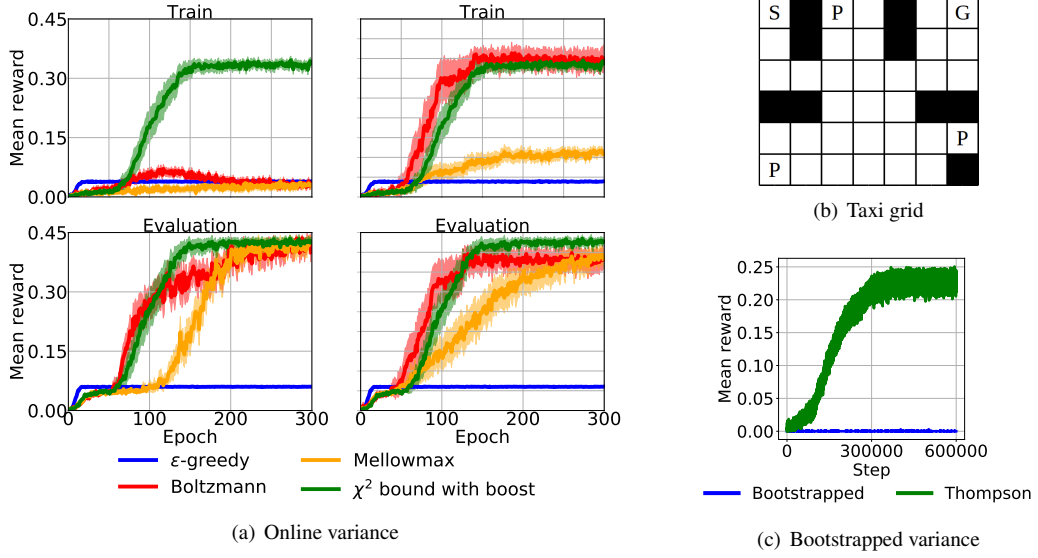The results on bootstrapping, shown in Figure 5.1(c), are obtained using $\varepsilon = 0$,

(a) Online variance

(b) Taxi grid

(c) Bootstrapped variance

**Figure 5.1:** *Results on the **Taxi** problem. Figure 5.1(a) shows the performance of online variance estimation in terms of average reward per step. The plots on the left show the results using the setting of the policy parameters that is optimal in evaluation; the ones on the right consider parameters optimal in training. An epoch corresponds to 1000 training steps. Figure 5.1(b) shows the structure of the grid where S is the initial position of the agent, P is a passenger, and G is the goal. Figure 5.1(c) shows the performance during training of the bootstrapping approach in terms of average reward per step.*

learning rate $\alpha = \frac{1}{n(s,a)^{0.3}}$ and 10 tabular approximators to do bootstrapping. The action-value functions are initialized sampling from a normal distribution $\mathcal{N}(0, 1)$. The boostrapping mask is set to 1 for all approximators. The results are averaged over 100 different seeds.

We also conduct further studies on the performance of the different algorithms described in Section 5.3. First, Figure 5.2 shows how the upper bounds to the variance can help to learn and also how $\chi^2$ upper bound with boost learns the optimal policy faster. Figure 5.3 shows how the standard Hoeffding bound is able to learn the optimal policy but, as discussed in Section 5.3, it delays too much the exploitation phase. It is also highlighted how parameter $c$ can be useful for accelerating learning, provided it is set to a sufficiently high value to avoid suboptimal performance. Eventually, in Figure 5.4 we show the improvement given by the $\chi^2$ upper bound to the Hoeffding bound with $c = 2$.

**Results** For the online case, we compare the performance of $\chi^2$ upper bound with boost against the classic exploration strategies $\varepsilon$-greedy and Boltzmann, together with a recently proposed variant of the Boltzmann called mellowmax [4]. For the bootstrapped case, we consider the tabular approximation and Double Q-Learning as the learning algorithm, opposed to BDQN that instead uses a deep neural network and Deep Double Q-Learning [93]; thus, we refer to this approach as *Bootstrapped Q-Learning*. Figure 5.1(a) shows that TS achieves better results in less time w.r.t. the other strategies. In particular, it shows how $\varepsilon$-greedy is completely ineffective in this problem, while Boltzmann and mellowmax allow to learn good policies, but still slower than TS. Thus,
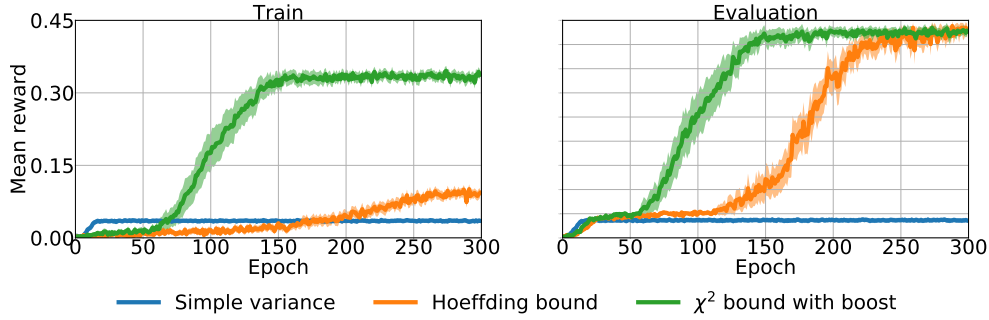
**Figure 5.2:** *Results of TS on Taxi with different upper bounds on variance.*
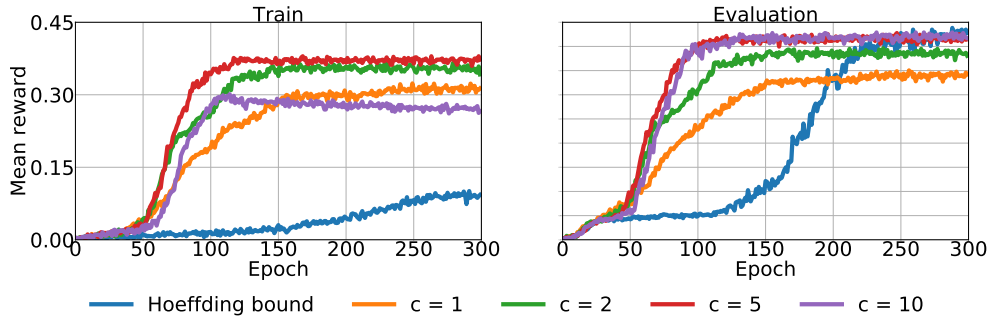


**Figure 5.3:** *Results of TS on Taxi using Hoeffding upper bound with different values of the c parameter.*

we can conclude that here TS shows the best balance between exploration and exploitation. Figure 5.1(c) shows similar results where surprisingly Bootstrapped Q-Learning is not able to move properly in the grid getting stuck against walls very often, while TS manages to move in the grid by learning a good policy even if not as good as in the online variance case.

### 5.4.2 Continuous state space

**Mountain Car**

**Details**    We use the *MountainCar-v0* implementation available in OpenAI Gym [17]. In this problem, a car must climb a hill in front of it to reach a flag, but its engine is not powerful enough to climb it starting from the initial position of the car. Therefore, the car must gain speed from the steep road behind it to reach the goal. The state is 2-dimensional representing the position and the velocity of the car. There are 3 actions: *push left*, *no push* and *push right*. The reward is $-1$ at every step. The episode ends when the car reaches the flag on the hill. The discount factor is $\gamma = 0.99$.

**Setting**    For the results in Figure 5.5(a), we use the parameter settings that perform the best in training: $\varepsilon = 0.01$ for $\varepsilon$-greedy, temperature $\beta = 10.5$ for Boltzmann and $\omega = 7.5$ for mellowmax. We sparsely encode the state space with 10 tilings of $10 \times 10$ tiles and approximate the action-value function with linear approximation with weights initialized uniformly random between $-10$ and 10. The learning rate is $\alpha = 0.1$. An evaluation epoch consists of averaging the cumulative reward per episode obtained in 5 episodes.
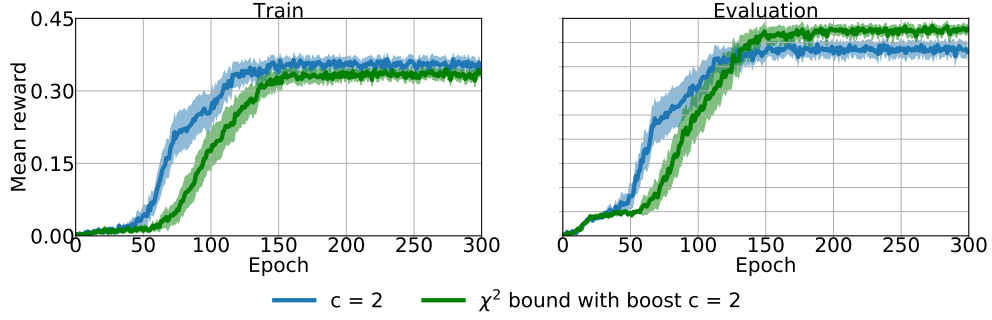
**Figure 5.4:** *Results of TS on Taxi using Hoeffding upper bound with $c = 2$ without and with $\chi^2$ upper bound boost.*



(a) Mountain Car
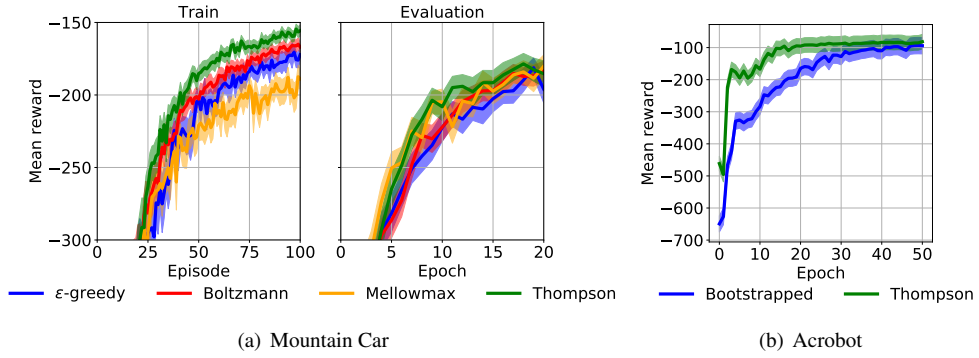
(b) Acrobot

**Figure 5.5:** *Figure 5.5(a) shows the mean cumulative reward in train and evaluation for Mountain Car, while Figure 5.5(b) shows the mean cumulative reward in evaluation for Acrobot. Evaluation epochs are performed every 20 training episodes for the former, and every 1000 training steps for the latter.*

**Acrobot**

**Details**  We use the *Acrobot-v1* implementation available in OpenAI Gym [17]. This problem consists of a pendulum with two links where only one of them is actuated. The goal is to swing the link that is the farthest to the fulcrum so as that to arrive to a height that has at least the length of the link closest to the fulcrum. The state is 6-dimensional representing trigonometric measures of the pendulum. There are 3 discrete actions to apply a torque of $-1$, $0$ or $1$ on the joint between the pendulum links. The reward is $-1$ at each step except for the step that goes into the goal state that returns $0$ and terminates the episode. The discount factor is $\gamma = 0.99$.

**Setting**  For the results in Figure 5.5(b), we use a replay memory of $5000$ samples initialized with $100$ randomly collected samples. We use a neural network with $10$ heads composed of $2$ layers of $80$ ReLu units and a linear output unit for each action value. We optimize using Adam optimizer with learning rate $0.0001$. The batch size is $100$ and mask $m_k = 1$. The target network is updated every $100$ weight updates and the training is stopped after $50,000$ steps. An evaluation epoch consists of averaging the cumulative reward per episode obtained by moving for $1,000$ steps.

**Results**  The need for exploration in the previously described two problems is given by the sparse reward function, even if exploration is not as critical as in the Taxi prob-

(a) Pong

(b) Breakout
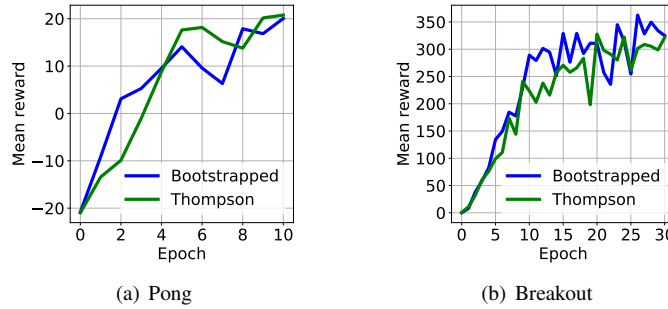
**Figure 5.6:** *Mean cumulative reward in evaluation. An epoch is performed every* $250,000$ *steps.*

lem. Nevertheless, we want to consider these problems to show how TS is robust w.r.t. generic RL problems. Since Mountain Car has a small continuous state space, we test our online variance estimation with SARSA (Algorithm 11) for the continuous case in this problem, while we consider the more complex Acrobot for bootstrapping. Figure 5.5 shows how TS reaches better performance than the others and how exploration with TS can help to speed up the convergence of the algorithm. In particular, Figure 5.5(a) highlights the effectiveness of the exploration driven by TS during the training phase.

### 5.4.3 Deep Reinforcement Learning

**Details** We use the *PongNoFrameskip-v4* and *BreakoutNoFrameskip-v4* implementations of **Pong** and **Breakout** available in OpenAI Gym [17]. The state is represented by $210 \times 160$ raw pixel frames and there are $4$ discrete actions in both of them. The reward function is sparse. In Pong, the episode ends when the score of one of the players reaches $21$. In Breakout, the episode ends when the agent finishes its lives.

**Setting** We use the same setting described in [58]. However, in this work, it is not clear whether $\varepsilon = 0$ for Bootstrapped DQN or not. For the purpose of our work, we decide to use $\varepsilon = 0$.

**Results** We evaluate TS via bootstrapping in DRL considering the **Atari** games [9] of **Pong** and **Breakout**. We have not been able to try a broader set of games for time issues; therefore we preferred quality over quantity by evaluating only these two games, which can be learned faster than others, with $3$ different seeds for both. Figure 5.6 shows that in these games TS does not give improvements w.r.t. the BDQN policy. However, the small number of experiments we used to average the results, the need of a better hyper-parameter search for TS which is likely to need different settings than the one used in BDQN and the not high relevance of exploratory actions in these two simple games, make these experiments just a preliminary evaluation of TS in the Atari domain.

# Exploration Driven by an Optimistic Bellman Equation

As described in Chapter 5, OFU-based techniques encourage exploration of unknown states hoping they are more convenient than already known ones, hence the definition of optimism. There are two types of optimism:

- **confidence interval estimation**, such as IEQL+ [54] and UCRL [6], which directly estimates $Q$-value confidence intervals;

- **optimistic estimate** of the action value, a method proposed for instance in [82] about which subsequently [32] proved the convergence to a near-optimal solution.

While the work in Chapter 5 deals with the first category of optimism-based algorithms, the work described in the following deals with the latter. This approach is inspired by the broad category of algorithms based on IM [78], which despite having less theoretical guarantees than Bayesian approaches, have been able to obtain impressive results [10]. As already discussed in Chapter 5, IM algorithms define an *intrinsic* reward replacing the extrinsic reward (or combining both), which acts as an exploration bonus. Often, the intrinsic reward is defined using heuristics, such as counting state visits and rewarding less visited states [61], or by the concept of *surprise*, which is the error in predicting future states [63]. However, the drawback of IM techniques is their lack of a principled definition of the intrinsic reward for exploration.

Considering this premise, we worked on the proposal of the Optimistic Bellman Equation (OBE), a novel variant of BE that results into an optimistic action-value estimate from an ensemble of action-value functions where the optimistic estimate is obtained from a maximum-entropy principle. For the exploration bonus that OBE implicitly defines, we can prove that the bonus decreases consistently with the number of

state visits. Our proposed algorithm can be seen as a mixture of different techniques: as an approximate Bayesian method [29, 95] we estimate uncertainty with an ensemble [58]; as optimism-based methods [7, 16, 45, 49] we select optimistic estimates, and, as IM, we propagate an implicit exploration bonus [71, 78, 98].

## 6.1 Learning value function ensembles with optimistic estimate selection

Ensemble methods [57] constitute a popular set of ML technique in which multiple models are used to learn the same target function. In addition to being commonly used to improve the generalization of the prediction, ensemble methods offer a simple way to estimate its uncertainty. We consider the application of ensemble methods in the RL framework with the purpose of approximating the action-value functions while having an estimate of their uncertainty, in order to apply the OFU principle in action selection.

### 6.1.1 An optimistic Bellman Equation for action-value function ensembles

Starting from the purpose of overestimating the action-value functions with the result of encouraging exploration, we propose OBE which propagates an optimistic estimate of the action-value function. In particular, OBE is a BE that incorporates the information about the uncertainty provided by an ensemble of action-value functions. We want to emphasize that, when all the action-value functions of the ensemble are identical, we assume that there is no uncertainty and, under this condition, the OBE will behave in exactly the same ways as the classic BE. Indeed, we can prove that the fixed point $Q^*$ of OBE is the same of the classic BE; in other words, the OBE differs from the classic BE when the estimates in the ensemble are different from each other, which usually happens when the action-value functions are approximated with a limited number of samples or with function approximators (e.g. neural networks). This makes sense since when the perfect solution is available there is no need for optimism and exploration. Note that the fact that OBE enjoys the properties of the classic BE, such as contractivity and the existence of a unique fixed point, potentially enables its usage in value-based and actor-critic algorithms.

The diversity in the action-value ensemble should ideally be consistent with the uncertainty of the estimate: when the estimate is certain, all the values in the ensemble should agree on the same value, otherwise the ensemble should have discordant values. Given an ensemble of action-value functions $\{Q_m\}_{m=1}^M$, we want to work out an optimistic estimate from the diverse estimates provided by the ensemble. The simplest and most optimistic solution is to select the highest value

$$\max_m Q_m(s, a). \tag{6.1}$$

However selecting the highest estimate makes poor use of the information provided by the ensemble and can be sensitive to noise. In order to mitigate this effect, we introduce a notion of *belief* over the estimates where $b_m(s, a)$ is the belief of $Q_m(s, a)$. The main idea is to add an entropic regularization term to the objective

$$\max_{b(s,a)} \sum_m b_m(s, a) Q_m(s, a) - c \sum_m b_m \log b_m \tag{6.2}$$

or to bound the information loss

$$\sum_m b_m \log b_m < \psi. \tag{6.3}$$

The hard constraint on the information loss is more appealing since the introduced hyper-parameter $\psi$ does not depend on the magnitude of the rewards, but has no closed-form solution. In contrast, the penalization weighing constant $c$ introduced by the soft-constraint regularization term is sensitive to the magnitude of the rewards, but admits a closed-form solution. Basing on these considerations, we define two different problems where we use an optimistic estimate of the action-value function.

**Entropy-regularized optimistic action-value selection**

We define a BE over the action-value function ensemble by introducing an optimistic estimate penalized by an entropic regularization term.

**Problem 1** (Regularized version)**.**

$$Q_i(s,a) = \max_{b(s,a)\in\mathcal{P}^M} R(s,a) + \gamma \sum_m b_m(s,a)V'_m(s,a) - \tfrac{1}{\eta}D_{\mathrm{KL}}\big(b_m(s,a)\big\|u\big)$$

$$\text{s.t. } \sum_{m=1}^M b_m(s,a) = 1$$

$$\forall s, a, i \in \mathcal{S} \times \mathcal{A} \times \{1, \dots, M\}$$

where $u_m = 1/M$, $V'_m(s,a) = \sum_{s'} P(s'|s,a)\max_{a'} Q_m(s',a')$ and $D_{KL}(b(s,a)\|u)$ is the Kullback-Leibler (KL) divergence between the belief $b(s,a)$ and the uniform distribution $u$. The choice of using the relative entropy (i.e. KL-divergence) instead of the absolute one has two main advantages: it admits a solution for $\eta \to 0$ and provides a normalization factor. Since Problem 1 is a convex constrained problem, it is solvable by dual optimization. Introducing $\lambda$ as Lagrangian multiplier for the constraint, we write the Lagrangian

$$L_i(s,a) = f(s,a;b(s,a)) - \frac{1}{\eta}D_{\mathrm{KL}}\big(b(s,a)\big\|u\big)$$
$$+ \lambda\bigg(\sum_m b_m(s,a) - 1\bigg). \tag{6.4}$$

Requiring the partial derivatives of $L_i$ w.r.t $p_m$ and $\lambda$ to be zero yields

$$b_m(s,a) = \frac{e^{\eta\gamma V'_m(s,a)}}{\sum_{k=1}^M e^{\eta\gamma V'_k(s,a)}}. \tag{6.5}$$

By substituting $b_m$ in Equation 6.4, we obtain the solution to the problem:

$$Q_i(s,a) = \begin{cases} \overline{R}(s,a) + \frac{1}{\eta}\log\frac{\sum_{m=1}^M e^{\eta\gamma V'_m(s,a)}}{M} & \text{if}\,\eta \neq 0 \\ \overline{R}(s,a) + \frac{\gamma}{M}\sum_{m=1}^M V'_m(s,a) & \text{otherwise} \end{cases}. \tag{6.6}$$

Notice that $\eta > 0$ leads to a positive (optimistic) biased estimate, while $\eta < 0$ leads to a negative (pessimistic) estimate; in this work, we will always assume $\eta > 0$ (and therefore we refer to the equation as optimistic). However, in general, the choice of $\eta$ is difficult since it depends on the magnitude of the reward function. For this reason, we introduce the constrained version of the proposed problem.

**Optimistic action-value selection bounding the information loss**

We bound the information loss between the distribution $b_m$ and the uniform distribution to maintain compatibility with Problem 1. The information loss is bounded between $-\log M$ and $0$ where $-\log M$ stands for complete information loss (i.e., only one model is selected) while $0$ corresponds to no information loss (i.e., uniform belief distribution). Constraining the information loss has succeeded in prior work, for instance in policy search methods such as [65].

**Problem 2** (Constrained version).

$$Q_i(s,a) = \max_{b(s,a) \in \mathcal{P}^M} \quad R(s,a) + \gamma \sum_m b_m(s,a) V'_m(s,a)$$
$$s.t. \quad D_{\mathrm{KL}}\big(b(s,a) \| u\big) \leq \iota_{\max}$$
$$\sum_{m=1}^M b_m(s,a) = 1$$
$$\forall s,a,i \in \mathcal{S} \times \mathcal{A} \times \{1,\ldots,M\}$$

By letting $\beta$ be the Lagrangian multiplier associated with the KL constraint, we obtain the Lagrangian

$$L_i = f(s,a;b(s,a)) + \beta(D_{\mathrm{KL}}(b(s,a)\|u) - \iota_{\max})$$
$$+ \lambda(\sum_m b_m(s,a) - 1). \tag{6.7}$$

Substituting $\beta$ with $-1/\eta$ we note that Equation 6.7 becomes identical to 6.4 except for a constant factor. Since we can not solve $\eta$ (or $\beta$) analytically, we obtain an approximate solution by iteratively optimizing $\eta$ (or $\beta$) and $b_m$ subsequently. OBE takes its name from the fact that when $\eta > 0$, the *logsumexp* acts as a softmax operator. Such operator is also well known as an *entropic mapping*, as it can be derived from a maximum-entropy principle.

   The use of the entropic mapping is not new in RL: an interesting use of the entropic mapping as a soft-max over the action in the BE is proposed in [4]; another approach proposing an entropic regularization over the state-action distribution is described in [65].

### 6.1.2   Relation to Intrinsic Motivation

To highlight the connection between OBE and IM, we reformulate OBE using the unbiased average of the estimates instead of the logsumexp, and by introducing the resulting exploratory bonus $U$ which includes the positive bias

$$Q_i(s,a) = \overline{R}(s,a) + U(s,a) + \gamma \sum_{m=1}^M \frac{V'_m(s,a)}{M} \tag{6.8}$$

with

$$U(s,a) = \frac{1}{\eta} \log \sum_{m=1}^{M} \frac{e^{\eta\gamma V_m'(s,a)}}{M} - \gamma \sum_{m=1}^{M} \frac{V_m'(s,a)}{M}. \tag{6.9}$$

Noticing that $\frac{\sum_{i=1}^{N} e^{\eta x_i}}{N}$ is the *sample moment generator* w.r.t. samples $\{x_i\}_{i=1}^{N}$ we can rephrase the exploration bonus as

$$
\begin{aligned}
U(s,a) &= \lim_{N \to +\infty} \frac{1}{\eta} \log[1 + \sum_{n=2}^{N} \frac{(\eta\gamma)^n}{n!} \mathcal{M}_n(s,a)] \\
&= \eta\gamma \mathcal{M}_2(s,a) + O(\eta^2),
\end{aligned} \tag{6.10}
$$

where $\mathcal{M}_n$ is the $n^{\text{th}}$ central moment of the random variable $V_m'$

$$\mathcal{M}_n(s,a) = M^{-1} \sum_{m=1}^{M} [(V_m'(s,a) - \overline{V}(s,a))^n]$$

with

$$\overline{V}(s,a) = M^{-1} \sum_{m=1}^{M} V_m'(s,a).$$

Equation (6.8) shows that OBE is equivalent to BE with an additional bonus defined by Equation (6.10). The bonus $U$ (for any positive $\eta$) is always positive, and provides a measure of the uncertainty w.r.t. $Q$. This is why OBE can be interpreted as a special principled form of IM.

**Explicit exploration**

A general problem affecting IM algorithms is that the policy greedy w.r.t. the obtained action-value function is not optimized for the original problem. As a solution to this issue, we approximate two functions: $\tilde{Q}$, which will be updated using the true reward and $Q_E$ which will be updated using only the intrinsic reward [83]. In this way, we obtain both the IM policy $\pi_o(s) = \arg\max_a \tilde{Q}(s,a) + Q_E(s,a)$ and the classic policy $\pi_u(s) = \arg\max_a \tilde{Q}(s,a)$. Define

$$\tilde{Q}_i(s,a) = R(s,a) + \gamma \sum_{m=1}^{M} \frac{\tilde{V}_m'(s,a)}{M} \quad \text{with} \tag{6.11}$$

$$\tilde{V}_m'(s,a) = \sum_{s'} P(s'|s,a) \max_{a'} \tilde{Q}_m(s',a') \tag{6.12}$$

to obtain an unbiased estimate of the action-value function, yielding

$$
\begin{aligned}
Q_E(s,a) &= \sum_{t=0}^{T} \gamma^t U(s_t, a_t) \quad \text{where} \quad s_0 = s, a_0 = a \\
&= \eta^{-1} \log \frac{\sum_{k=1}^{M} e^{\eta\gamma \max_{a'} \tilde{Q}_k(s',a') + Q_E(s',a')}}{M} \\
&\quad - \frac{\sum_{k=1}^{M} \gamma \max_{a'} \tilde{Q}_k(s',a')}{M}.
\end{aligned} \tag{6.13}
$$

By a simple equation rearrangement, it is possible to show that $\tilde{Q}_i(s,a) + Q_E(s,a)$ is equivalent to $Q_i(s,a)$ as defined in the OBE (6.6).

## 6.2 Optimistic value function estimators

The OBE offers a theoretical framework in which it is possible to develop optimistic value based algorithms. In fact, OBE enjoys all the desirable properties of the BE (e.g. max-norm contractivity). We present briefly two practical applications of the OBE that are optimistic variants of $Q$-learning and DQN that we call, respectively, Optimistic $Q$-Learning (OQL) and Optimistic Deep $Q$-Network (ODQN).

### 6.2.1 Optimistic $Q$-Learning

Motivated by the idea of employing an ensemble of regressors as it is done in BDQN [58], we assume to have $M$ randomly initialized $Q$-tables. Inspired by the well-known $Q$-learning update rule (see Section 2.3.2), we derive an optimistic version that is consistent with the OBE as follows:

$$
\begin{aligned}
Q_{i,t+1}(s,a) = {} & (1 - \alpha_t)Q_{i,t}(s,a) \\
& + \alpha_t(r_t + \frac{1}{\eta} \log M^{-1} \sum_{j=1}^{M} e^{\gamma \max_{a'} Qj,t(s',a')}).
\end{aligned}
$$

We show that, with the proposed update rule, given infinite visits of each state-action pair, all the tables will converge to the same values, and more precisely, after each update, the $n$-th central moment of the updated cell is scaled exactly by $(1 - \alpha_t)^n$:

$$
\mathcal{M}_{n,t+1}(s,a) = (1 - \alpha_t)^n \mathcal{M}_{n,t}(s,a) \tag{6.14}
$$

where

$$
\mathcal{M}_{n,t}(s,a) = M^{-1} \sum_{i=1}^{M} (Q_{i,t}(s,a) - \sum_{k=1}^{M} \frac{Q_{k,t}(s,a)}{M})^n.
$$

This implies that a cell updated $N$ times, with learning rates $\{\alpha_i\}$, will have the $n$-th central moments scaled by $\Pi_{\alpha_i}(1 - \alpha_i)^n$ w.r.t. the initial one. This leads us to some considerations:

1. the bonus decreases accordingly to the number of state visits;

2. differing from several count-based approaches, our algorithm takes into account the impact of the learning rate;

3. in the limit of an infinite number of visits, the exploration bonus converges to $0$.

All the considerations made so far provide a deeper insight into the functioning of the algorithm and its properties. However, in more complex settings, (e.g., function approximation) convergence to $0$ of the exploration bonus is not guaranteed in general.

---

**Algorithm 13** Optimistic Deep $Q$-Network

---

**Input:** $\{Q_k\}_{k=1}^K$, $\iota_{\max}$, $\eta_{\text{init}}$, $\chi$, $N$, $C$
Let $B$ be a replay buffer storing the experience for training.
$\eta = \eta_{\text{init}}$.
Let $i \sim \text{Uniform}\{1 \dots M\}$ and $\psi = 1$ w.p. $\chi$ otherwise $\psi = 0$
**for** $N$ epochs **do**
  **for** $C$ steps **do**
    Observe $s$
    Choose $a = \arg\max_a Q_i(s,a) + \psi Q_1(s,a)$
    Observe reward $r$, next state $s'$, end of episode $t$
    If $t$ is terminal, $i \sim \text{Uniform}\{2 \dots M\}$ and
       $\psi = 1$ w.p. $\chi$ otherwise $\psi = 0$
    Store $< s, a, r, s', t >$ in buffer $B$
    Sample mini-batch $B_{\text{batch}}$
    Update $\{Q_k\}_{k=1}^K$ using equation (6.15)
    $V \leftarrow V + |$ violated constraints (6.16) in $B_{\text{batch}}|$
  **end for**
  Let $\rho = \frac{V}{C*\text{batch\_size}}$
  Update $\eta$ by (6.17)
  Update target network
**end for**

---

## 6.2.2 Optimistic Deep $Q$-Network

In addition to the novel OQL algorithm described above that can be used for limited discrete state spaces, we propose another algorithm for continuous state spaces based on our OBE taking inspiration from the framework provided by BDQN [58] that uses an ensemble of neural networks for estimating the action-value function. To get an unbiased performance evaluation, we decided to update $M - 1$ components of the ensemble with the update rule provided by BDQN and to use the remaining $M$-th component of the ensemble to approximate $Q_E$. Using the first component to approximate $Q_E$, we get for our new algorithm ODQN the loss

$$
\begin{aligned}
\mathcal{L}_O(s,a) = (\eta^{-1} \log \frac{\sum_{k=2}^M e^{\eta\gamma\max_{a'} Q_k^T(s',a') + Q_1^T(s',a')}}{M} \\
- \frac{\sum_{k=2}^M \gamma\max_{a'} Q_k^T(s',a')}{M} - Q_1(s,a))^2 \\
+ \sum_{k=2}^M (r + \gamma\max_{a'} Q_k^T(s',a') - Q_k(s,a))^2.
\end{aligned}
\tag{6.15}
$$

The exploratory bonus represented by $Q_E = Q_1$ in the proposed OQL and ODQN algorithms is needed to guide exploration during learning. During the evaluation, we use majority voting on the remaining $M - 1$ components $\{Q_k\}_{k=2}^M$. While we always select an optimistic policy in OQL during the training phase, in ODQN the neural network function approximator may have problems in learning to approximate the optimal policy: if there are not enough unbiased samples, the approximator may learn to model only the optimistic biased samples. Note that in the tabular case, this is not a problem since there is no action-value function approximation. To mitigate this problem, we introduce a hyper-parameter $\chi$ that denotes the probability of selecting an optimistic policy $\pi_o$ in place of the unbiased one $\pi_u$. In this way, we can balance the number of unbiased and optimistic samples. Algorithm 13 shows the pseudocode of ODQN.

**Automatic hyper-parameter adaptation**   Recalling that the regularization coefficient $\eta$ in the OBE is hard to tune, we want to focus our attention on Problem 2. We propose a way, inspired by [72], to optimize $\eta$. One of the optimization techniques proposed in this work is to measure the "degree" of constraint violation and to update the Lagrangian multiplier accordingly. We have to adapt the technique to multiple constraints, as the problem is defined for each state-action pair: we count the number of times the constraints have been violated and then we update $\eta$. More in detail, suppose to have $N$ state-action pairs and for each pair $(s_i, a_i)$

$$\sum_m b_m(s_i, a_i)(\log b_m(s_i, a_i) + \log M) \leq \iota_{\max}, \tag{6.16}$$

where $\iota_{\max}$ is defined in Problem 2, while $b_m(s_i, a_i)$ is defined by (6.5). We define $\rho$ as the ratio of violated constraints. We update $\eta$ according to the following rule

$$\eta_{T+1} = \frac{\eta_T}{(0.5 + 10\rho)}. \tag{6.17}$$

In ODQN, we decided to count the number of constraints violated every $C$ time-steps (basically every update of the target network), using the samples of all the extracted mini-batches. See Algorithm 13 for further details.

**Ensuring a prior distribution**   As already discussed, it is important to maintain diversity in our ensemble and this diversity should reflect the degree of uncertainty. For this reason, we should introduce a sort of prior distribution, as happens in the Bayesian framework. In the case of OQL, we observe that it is sufficient to randomly initialize each element of the ensemble, since diversity between estimates is a sufficient condition to obtain a positive bonus. For ODQN, as is done in BDQN, we choose to maintain the diversity between approximation, through a random initialization of the parameters of each component.

## 6.3   Experimental evaluation

In the experiments, we compare OQL with a tabular variant of BDQN that we call Bootstrapped $Q$-Learning (BQL), classical $Q$-learning and Optimistically Initialized $Q$-Learning (OIQL) [82]; on the other hand, we preliminarily evaluate ODQN method with BDQN and classical DQN in a simple problem.

### 6.3.1   Settings

The environments are chosen to cover different types of dynamics and having sparse rewards: they are Frozen Lake as implemented in [17], Taxi as presented in Chapter 5 and the chain in Figure 6.1.

**Initialization of the action-value function**   In the tabular setting, for OIQL we initialize the action-value function to $15$ in Taxi and to $1$ in Chain and Frozen Lake. For the other algorithms, we initialize $Q(s, a) \sim \mathcal{N}(\mu = 0, \sigma = 2)$, except $Q_E$ of OQL which is initialized to $0$. In the taxi environment, we use a shared convolutional layer with multiple heads as described in [58]. In the case of ODQN we initialize the output layer corresponding to $Q_E$ to small parameter values, in order to obtain initially $Q_E \approx 0$.
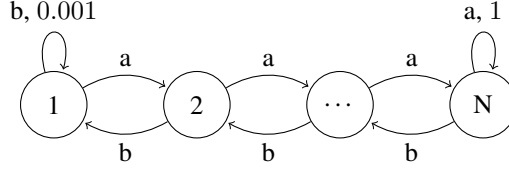
**Figure 6.1:** *Structure of the Chain.*
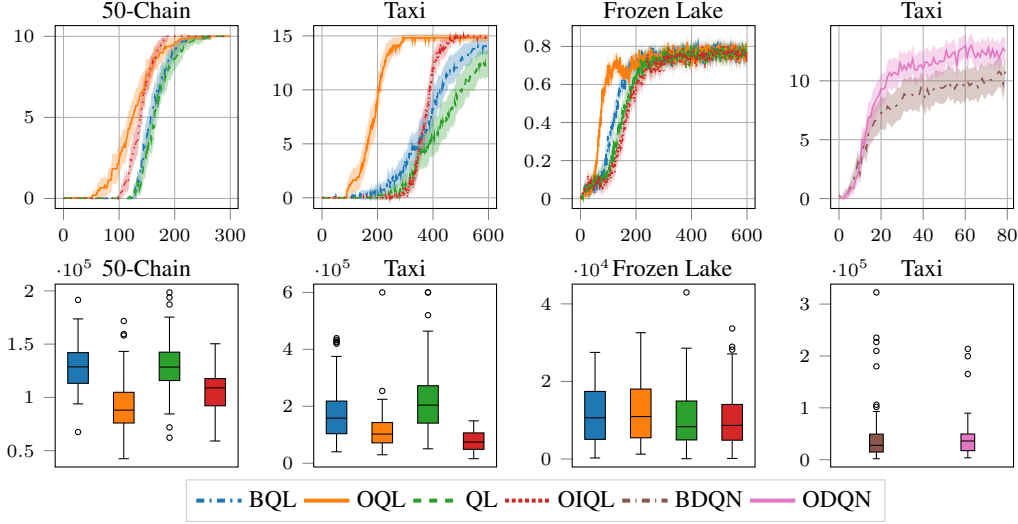


**Figure 6.2:** *The first row shows the average return for each tabular algorithm in the $N$-Chain, Taxi, and Frozen Lake environments and for each neural network based algorithm in the Taxi and Acrobot environments together with $95\%$ confidence intervals. The second row shows the distribution over the number of time steps before observing the maximum reward in the MDP.*

**Hyper-parameters tuning** For tabular settings, we did not run any hyper-parameter optimization. With neural networks, we compare ODQN and BDQN using the best hyper-parameter setup found for BDQN. In OQL we use $\eta = 10$ and for ODQN, we use $\chi = 0.25$ and $\iota_{\max} = 1$.

### 6.3.2 Results

Figure 6.2 summarizes the results obtained by averaging over $64$ different seeds for the tabular algorithms, and $100$ seeds for DQN and BDQN. OQL learns faster than the other tabular algorithms. In the Chain environment and in Taxi, our algorithm OQL finds the highest reward faster than BQL or QL. On the other hand, also OIQL seems to find high rewards fast, as shown in the box-plots. However, OIQL requires more training epochs to escape the high initial optimistic values of the value function. In contrast, OQL quickly finds high values in all problems while converging to a near-optimal solution.

With the neural network approximation, ODQN outperforms BDQN in the Taxi environment demonstrating that the principles of OBE can apply also with function approximation. However, the simplicity of the environment makes this experiment only a preliminary evaluation of ODQN.

# Part IV

# Final Remarks

CHAPTER 7

# Conclusion

The end of three years of Ph.D research is surely a significant moment. I started my Ph.D. without being sure of what the world of research was like and whether I would have been able to do a good Ph.D or not. Happily, after all this experience I can be very satisfied of what I have achieved and learned. This thesis is the result of three years in which I studied and worked on amazing topics that are one of the most important of the current decade and, presumably, of the next ones. In particular, having the feeling of being part of the huge community of researchers aiming to the progress of such a revolutionary scientific topic, represented a very significant achievement for me. Moreover, considering that I had the possibility to participate in the most important conferences around the world, to work in an excellent team and to be followed by a great advisor who I thank for his help, I can say this Ph.D. gave me what I hoped and maybe more.

## 7.1  Recap of the thesis

This thesis describes the results we obtained during my three years of Ph.D. research about the exploitation of uncertainty in Reinforcement Learning (RL). After a first part with the introduction about general considerations on RL and the description of some preliminary concepts, the thesis is split into two parts where it is showed how the uncertainty on action-value estimates can be exploited to improve the performance of state-of-the-art algorithms. In the former part, uncertainty is used to improve the estimate of the components of the equation of the action values by means of the Bellman operator; in the latter part, uncertainty is used to drive exploration in order to improve sample-efficiency of exploration policies. The exploitation of uncertainty is not a new line of research in RL, thus our works are mainly inspired by the methodologies available in the literature that we considered to work out our own novel ones.

### 7.1.1 Bellman update

**What we studied**

The first work we dealt with, after the initial study of the RL literature and the state-of-the-art works, was the Double $Q$-Learning (DQL) [90]. This paper addresses the problem of overestimating the Maximum Expected Value (MEV) in the context of action-value function estimation. This happens, for instance, in $Q$-Learning (QL) [79] because of the Maximum Estimator (ME) involved in the update rule via the Bellman Equation (BE). The importance of DQL stands on the proposal of a variant of QL that replaces the ME with the Double Estimator (DE) making DQL able to avoid overestimation providing, on the contrary, an underestimation of the optimal action values. The underestimation helps to have a good learning in some problems with high stochasticity and, more generally, in problems where there is not clearly an action that is better than the others. One of the interesting aspects of the DE is that it can be used in several value-based RL algorithms without major changes. For instance, the offline value-based algorithm of Fitted $Q$-Iteration (FQI) [30] can be easily modified to make it use DE, resulting in what we call Double Fitted $Q$-Iteration (DDFQI) algorithm. Moreover, the ideas behind DQL have been also applied in Deep RL (DRL) with the Double Deep $Q$-Network (DDQN) [40], a variant of Deep $Q$-Network (DQN) [55] that we also considered during my research on this topic.

**What we did**

To address the problem of overestimating the MEV, we worked on introducing a new estimator called Weighted Estimator (WE) that computes an weighted average of action-values where the weights are the probability of each action to be the best one. We showed how the WE can be both positively or negatively biased, but its absolute bias is always lower than the ones of ME and DE. We applied this estimator to QL leading to Weighted $Q$-Learning (WQL) which we compare to QL and DQL in several discrete problems. Then, we proposed an extension of WE to continuous problems by means of a variant of FQI, called Weighted Fitted $Q$-Iteration (WFQI), which uses WE and Gaussian Process (GP) regression to estimate the uncertainty of the prediction. The interesting aspect of this method is that it adapts naturally to problems with continuous actions, being one of the few value-based approaches that can handle infinite action spaces. Eventually, we studied the application of WE also in DRL by introducing a variant of DQN called Weighted Deep $Q$-Network. The adaptation of WE in DQN is not straightforward since the estimation of the uncertainty is cumbersome to obtain in neural networks. To make it practical, we took inspiration from the network architecture proposed in Bootstrapped Deep $Q$-Network (BDQN) [58] which uses a single network that is split into multiple outputs resulting in an ensemble of action-value function estimates. Then, we used the estimates provided by the ensemble to work out the uncertainty to use in the computation of WE. This method is a first practical solution to use WE in DQN, but the unsatisfactory preliminary results and the possibility of using different methods to compute the uncertainty make this work only a first solution in this direction.

The study of the overestimation of MEV is not the only way we considered improving the updating of action-value function estimate via the BE. Our second work about

this introduces the $RQ$-Learning algorithm which is the result of the effort put into decomposing the BE and exploiting the uncertainty of its components, i.e. the reward and the maximum action-value. This is done considering the fact that the sources of uncertainty of the two components of the BE are different and, thus, make sense to consider them separately. Together with this consideration, a different learning rate is used for the two components and adapted according to the measure of uncertainty. The method is compared with several variants of QL (e.g. DQL and WQL) showing significant results and robustness in heterogeneous problems. We also proposed an on-policy variant of the methodology and compared it with SARSA achieving good results in this case too.

### 7.1.2 Exploration

**What we studied**

The problem of exploration in RL is one of the most addressed in the RL literature. It deals with the agent's purpose to cover the a significant portion of the state space of the environment in order to improve the quality of the learned policy. However, the balance between exploration and exploitation is critical in order to obtain higher performance in terms of the cumulative discounted reward; therefore, an exploration policy that minimizes the number of samples needed to learn an effective exploitative policy is desirable. Among the trivial $\varepsilon$-greedy strategy that makes no use of the information acquired by the agent, the Boltzmann and mellowmax policy [3] use the current estimate of the action-value function, computing the next action to be executed by a soft-max operator.

More complex works to deal with exploration are based on the concept of Optimism in the Face of Uncertainty (OFU) which encourages the execution of actions that lead to unknown regions of the state space. This is pursued, among others, by strategies based on Thompson Sampling (TS) [89] and by the algorithms belonging to the Intrinsic Motivation (IM) [70] category. In exploration strategies based on TS, e.g. $Q$-value sampling [24], the actions are sampled from the distribution modeling their probability of being the ones with the highest action value. This has been widely studied in the context of Multi-Armed Bandit problems, but it has been proven to be an effective way to balance exploration and exploitation also in the context of RL [6]. On the other hand, the IM strategies add an intrinsic reward, which is summed to the reward returned by the environment, expressing the quality of the reached state in terms of novelty. In particular, the intrinsic reward is usually proportional to the amount of new knowledge of the state space obtained by the agent when reaching it; in this way, the agent is intuitively encouraged to visit new kinds of states. The challenging part of IM is to work out a measure to evaluate the novelty of a state, which could significantly slow down learning. Among others, some works address this issue by proposing different ways of counting the number of similar states reached [85] and others introduce new measure such as the agent's curiosity [63, 70].

**What we did**

We addressed the exploration problem by analyzing both the TS and IM strategies that are both based on OFU. In the first work, we introduced some algorithms to make the

use of exploration strategies based on TS practical in RL. Indeed, several methodologies based on TS have certainly desirable theoretical properties, but perform poorly in empirical applications. The main contribution of this work is the proposal of different ways of computing uncertainty and pursuing the OFU principle. In particular, the first approach is developed for discrete problems and exploits the computation of uncertainty explained in the work about WE [26] also applying statistical upper bounds to encourage exploration, while the second way is proposed as a solution for continuous high dimensional problems where the uncertainty is estimated via an ensemble of action-value function approximator as proposed in BDQN. The different algorithms are empirically evaluated in increasingly complex problems in which exploration is a critical aspect to solve them. The results highlight how they are able to achieve better performance faster than other sampling strategies, thus showing a better balancing between exploration and exploitation.

The second work in this direction is more based on IM and proposes a variant of the BE called Optimistic BE (OBE). In OBE, an ensemble of action-value estimates is used to compute the update of the action values based on a maximum-entropy principle. The entropy maximization aims to increase the action values proportionally to their uncertainty provided by the ensemble in such a way to encourage exploration. This work is inspired by the IM literature because of the intrinsic exploration bonus involved in the update of the action-value, but with the desirable property of not using any time-consuming way to explicitly measure uncertainty. The OBE is applied in variants of QL and DQN, which we called Optimistic QL (OQL) and Optimistic DQN (ODQN) respectively. These methodologies are theoretically studied, e.g. the convergence of OQL is proven, and empirically evaluated against Bootstrapped Q-Learning (BQL) and BDQN, showing better performance in the considered problems.

### 7.1.3 Comments

All the described works have been done in collaboration with colleagues and/or master students working on their MSc thesis, together with the help of my advisor. While I think the works about the improvement in the Bellman update BE have been studied enough in depth, I think there is still work to be done on exploration. In more detail, the study of exploration strategies based on TS brought to satisfactory results, but the development of theoretical guarantees is lacking to make this work interesting for publication. On the other hand, the work about the OBE is currently under review, but I think it needs to be better studied especially in its empirical evaluation.

As I focused on the problems described before, I have always considered the progress in the Deep Reinforcement Learning (DRL) literature. During my Ph.D., this field has become more and more important to the point that the empirical evaluation of the proposed novel works had to be done on DRL problems, e.g. Atari games [9], to make them more appealing. In three years, I tried to work on DRL studying novel ways to improve state-of-the-art methods, e.g. smart feature extraction in DQN, and these works brought to the publication of some MSc thesis, but were not significant enough to be published at conferences. The major problems I faced working on DRL were the very high computational demand in terms of resources and time to the point that I had to wait several days to obtain the results for a single experiment. Considering the slowness of working on DRL, I preferred to focus on practical classical RL problems
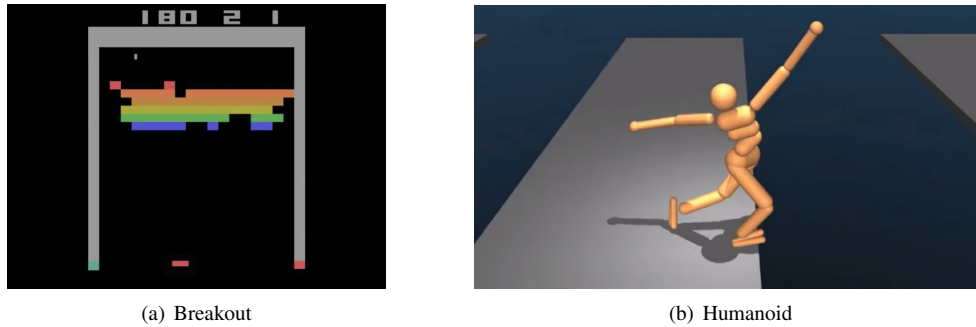
(a) Breakout

(b) Humanoid

**Figure 7.1:** *Graphical rendering of Breakout and Humanoid problems.*

and secondly extend these works in DRL applications, e.g. applying WE in DQN by introducing Weighted DQN (WDQN).

## 7.2 Future directions

All the works done for this thesis, the others I did but which are out of the focus, and all the ones I studied in the literature, gave me a lot of insights on the research in RL and on the important problems that need to be addressed to improve the state of the art of this field. In particular, the current effort in solving highly-dimensional complex problems is resulting in outstanding performance, as in the case of chess [76], but is sometimes ignoring the huge increase in computational resources to solve these problems. Indeed, typically a DRL model has an enormous amount of parameters, requires days of training time corresponding to many years of human play, often has unstable learning and most often the semantics of the learned representation is hard to understand to the point that it is often used in a black-box way [55]. As a matter of fact, despite the excellent performance that a DRL model and more in general a DL model is able to achieve, this last drawback causes serious problems of robustness of the learned representations leading to unsatisfactory results. For instance, in a recent work on image classification with deep neural networks, the authors show how it is possible to trick the model by means of samples specifically modified for this purpose [101]. These "adversarial examples" are usually samples of slightly modified images in such a way that, for instance, a single pixel is changed. The pixel is not changed randomly, on the contrary its value is smartly modified by studying the gradient of the function learned by the classifier. In the end, the original sample and the adversarial one look totally the same at a human eye, but the classifier sees them as completely different samples. The reasons behind this issue are multiple and the analysis of them requires a much longer discussion that is out of focus for this last section of the thesis, but substantially it can be concluded that this issue demonstrates the lack of robustness of the DL model and, more in general, the difficulty for the model to extract the real semantics of what it learns.

Considering that the models and their fitting algorithms used in DRL are the same as those of DL, it is intuitive how they can suffer from the previously described issues with an impact on the quality of the learned policy. Figure 7.1 shows two examples of environments to highlight problems that may arise when learning how to solve them. In

particular, Figure 7.1(a) refers to the Breakout game briefly described in Section 5.4.3. Recalling that in this game the purpose is to catch the ball with the platform on the bottom so as to make it bounce and hit the bricks on the wall at the top, it is intuitive how the number of features needed to solve this games is much less than the whole number of pixels of the raw frame. Thus, hopefully the deep $Q$-network should be able to extract only the few relevant features, such as the coordinates of the ball and the platform, in order to simplify the policy learning. However, this usually does not happen and the network learns a very abstract representation of the input whose semantics is practically not interpretable. On the other hand, Figure 7.1(b) refers to a control problem in which a humanoid walker is taught how to walk avoiding obstacles. In this case, the problem lies in the quality of the learned policy to make the humanoid walk. Indeed, usually the policy learned by the agent is effective for solving the environment, but let the humanoid walk in a very unnatural way, very different from the human way of walking.

These examples show that the outcome of the learning process of a DL model is almost always very unpredictable and it is unlikely to resemble the model learned by a human being. This happens because is very natural for a human to extract the semantics behind what he/she sees, while for a DL model this is generally not a natural way of learning. Taking these considerations into account, I believe that helping DRL models to extract the real semantics of a problem is a promising way of trying to improve the performance of DRL algorithms. To this end, the field of Multi-Task learning aims to train a single model on multiple problems. In this way, the knowledge about each task is shared with the one about the other tasks and this helps to force the model in extracting features with significant semantics, just like a regularization method. To this end, I think DRL is particularly suited thanks to the huge amount of training parameters that may intuitively allow the model to be able to learn multiple problems. Indeed, recently proposed works in DRL about Multi-Task [2, 41, 87] have shown promising results and I believe that more effort should be put in this direction to make DRL algorithms more and more efficient.

# Bibliography

[1] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39–1, 2012.

[2] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *International Conference on Machine Learning*, pages 1206–1214, 2014.

[3] Kavosh Asadi and Michael L. Littman. An alternative softmax operator for reinforcement learning. *arXiv preprint arXiv:1612.05628*, 2016.

[4] Kavosh Asadi and Michael L. Littman. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning*, pages 243–252, 2017.

[5] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

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

[7] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *Asteroids*, 2517(1516):108.

[8] Bing-Kun Bao, Bao-Qun Yin, and Hong-Sheng Xi. Infinite-horizon policy-gradient estimation with variable discount factor for markov decision process. In *Proc. ICICIC*, pages 584–584. IEEE, 2008.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, jun 2013.

[10] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[11] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[12] Marc G. Bellemare, Georg Ostrovski, Arthur Guez, Philip S. Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[13] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[14] Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2005.

[15] Andrea Bonarini, Alessandro Lazaric, Marcello Restelli, and Patrick Vitali. Self-development framework for reinforcement learning agents. In *International Conference on Development and Learning*, volume 178, pages 355–362, 2006.

[16] Ronen I. Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

# Bibliography

[17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[18] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[19] chainer.org. ChainerRL. https://github.com/chainer/chainerrl, 2017.

[20] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems*, pages 2249–2257, 2011.

[21] Nuttapong Chentanez, Andrew G. Barto, and Satinder P. Singh. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1281–1288, 2005.

[22] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Proc. NIPS*, pages 1017–1023, 1996.

[23] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.

[24] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI*, pages 761–768, 1998.

[25] Carlo D'Eramo, Alessandro Nuara, Matteo Pirotta, and Marcello Restelli. Estimating the maximum expected value in continuous reinforcement learning problems. In *AAAI*, pages XXX–XXX. AAAI Press, 2017.

[26] Carlo D'Eramo, Marcello Restelli, and Alessandro Nuara. Estimating maximum expected value through gaussian approximation. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1032–1040. JMLR.org, 2016.

[27] Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[28] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[29] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208. ACM, 2005.

[30] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[31] Eyal Even-Dar and Yishay Mansour. *Learning Rates for Q-Learning*, pages 589–604. Springer Berlin Heidelberg, 2001.

[32] Eyal Even-Dar and Yishay Mansour. Convergence of optimistic and incremental q-learning. In *Advances in neural information processing systems*, pages 1499–1506, 2002.

[33] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011*, 2015.

[34] Jürgen Franke and Michael H. Neumann. Bootstrapping neural networks. *Neural Computation*, 12(8), 2000.

[35] Alborz Geramifard, Christoph Dann, Robert H. Klein, William Dabney, and Jonathan P. How. Rlpy: A value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16:1573–1578, 2015.

[36] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[37] Mohammad Ghavamzadeh, Hilbert J. Kappen, Mohammad G. Azar, and Rémi Munos. Speedy q-learning. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Proc. NIPS*, pages 2411–2419. Curran Associates, Inc., 2011.

[38] Ole-Christoffer Granmo. Solving two-armed bernoulli bandit problems using a bayesian learning automaton. *International Journal of Intelligent Computing and Cybernetics*, 3(2):207–234, 2010.

[39] Michael Grossman and Robert Katz. *Non-Newtonian Calculus: A Self-contained, Elementary Exposition of the Authors' Investigations...* Non-Newtonian Calculus, 1972.

[40] Hasselt Hado van, Guez Arthur, and Silver David. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[41] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490, 2017.

[42] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

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

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

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

[46] Kunikazu Kobayashi, Hiroyuki Mizoue, Takashi Kuremoto, and Masanao Obayashi. *A Meta-learning Method Based on Temporal Difference Error*, pages 530–537. Springer Berlin Heidelberg, 2009.

[47] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[48] J. Zico Kolter and Andrew Y. Ng. Near-bayesian exploration in polynomial time. In *International Conference on Machine Learning*, pages 513–520. ACM, 2009.

[49] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[51] Daewoo Lee, Boris Defourny, and Warren B. Powell. Bias-corrected q-learning to control max-operator bias in q-learning. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 93–99. IEEE, 2013.

[52] Donghun Lee and Warren B Powell. An intelligent battery controller using bias-corrected q-learning. In *AAAI*. Citeseer, 2012.

[53] Benedict C. May and David S. Leslie. Simulation studies in optimistic bayesian sampling in contextual-bandit problems. *Statistics Group, Department of Mathematics, University of Bristol*, 11:02, 2011.

[54] Nicolas Meuleau and Paul Bourgine. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154, 1999.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[56] Salman Mohagheghi, Yamille del Valle, Ganesh Kumar Venayagamoorthy, and Ronald G. Harley. A proportional-integrator type adaptive critic design-based neurocontroller for a static compensator in a multi-machine power system. *IEEE Transactions on Industrial Electronics*, 54(1):86–96, 2007.

[57] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[58] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016.

[59] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

[60] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pages 2377–2386, 2016.

[61] Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

[62] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[63] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, volume 2017, 2017.

[64] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proc. AAAI*, 2010.

[65] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta, 2010.

# Bibliography

[66] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[67] Martin Riedmiller. Neural fitted q iteration. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[68] Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[69] Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke. Tensorforce: A tensorflow library for applied reinforcement learning. Web page, 2017.

[70] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *International Conference on Simulation of Adaptive Behavior: From animals to animats*, pages 222–227, 1991.

[71] Jürgen Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on Anticipatory Behavior in Adaptive Learning Systems*, pages 48–76. Springer, 2008.

[72] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[73] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[74] Steven L. Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[75] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[76] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[77] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[78] Satinder P. Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2004.

[79] James E. Smith and Robert L. Winkler. The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006.

[80] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. Pac model-free reinforcement learning. In *International Conference on Machine Learning*, pages 881–888. ACM, 2006.

[81] Malcolm Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, pages 943–950, 2000.

[82] Richard S. Sutton, Andrew G. Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[83] István Szita and András Lőrincz. The many faces of optimism: a unifying approach. In *Proceedings of the 25th international conference on Machine learning*, pages 1048–1055. ACM, 2008.

[84] Tambet, Matiisen. simple-dqn. https://github.com/tambetm/simple_dqn, 2015.

[85] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

[86] Brian Tanner and Adam White. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10(Sep):2133–2136, 2009.

[87] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[88] Ambuj Tewari and Peter L. Bartlett. *Bounded Parameter Markov Decision Processes with Average Reward Criterion*, pages 263–277. 2007.

[89] 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.

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

[91] Hado Van Hasselt. Estimating the maximum expected value: an analysis of (nested) cross-validation and the maximum sample average. *arXiv preprint arXiv:1302.7175*, 2013.

[92] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[93] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[94] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.

[95] Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart. Bayesian reinforcement learning. In *Reinforcement Learning*, pages 359–386. Springer, 2012.

[96] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[97] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[98] Martha White and Adam White. Interval estimation for reinforcement-learning algorithms in continuous-state domains. In *Advances in Neural Information Processing Systems*, pages 2433–2441, 2010.

[99] Min Xu, Tao Qin, and Tie yan Liu. Estimation bias in multi-armed bandit algorithms for search advertising. In Burges C.j.c., Bottou L., Welling M., Ghahramani Z., and Weinberger K.q., editors, *Advances in Neural Information Processing Systems 26*, pages 2400–2408. 2013.

[100] Naoto Yoshida, Eiji Uchibe, and Kenji Doya. Reinforcement learning with state-dependent discount factor. In *Proc. ICDL*, pages 1–6. IEEE, 2013.

[101] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint arXiv:1712.07107*, 2017.

[102] Zhang Zongzhang, Pan Zhiyuan, and Kochenderfer Mykel J. Weighted double q-learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3455–3461, 2017.

# Mushroom

The empirical evaluation of proposed methodologies in ML is a critical aspect to take into consideration when publishing a work. Indeed, considering that ML is mostly used in practical application, the empirical effectiveness of an algorithm is a desirable property. Because of this reason, the majority of ML articles describes a usually extensive empirical section presenting the improvement of the proposed work on the state-of-the-art. This focus on empirical results happens also in RL where the algorithms presented in the papers are empirically compared to other methodologies available in literature with the purpose to show some improvements w.r.t. them, e.g. in terms of cumulative discounted reward on several problems.

Several times during my Ph.D. research I needed to implement the code of state-of-the-art algorithms and of the RL environments from scratch, since especially at the beginning of my Ph.D. there were not reliable and/or maintained RL libraries. Unfortunately, the process of writing codes from scratch for every project was very time consuming and frustrating, especially considering that the same algorithms had to be coded differently just to adapt them to the structure of the ongoing project. Obviously, the problems of time spent in writing code and of reliability of the written code could have been solved with a unified software framework.

For this reason I decided to develop my own RL library, which I called *Mushroom*, for helping my research in terms of time and quality. Despite the fact that the development of Mushroom started just for my own purposes, I tried to make it as general as possible and month-by-month it became a general-purpose RL library to all effect. Mushroom has the purpose to provide a common interface to develop and run RL and it is designed to minimize the effort in writing the code of the experiment: in most cases, the user would only have to write a small script specifying the needed information such as the algorithm to run and the environment. One of the ideas behind Mushroom is to exploit widely used standard libraries such as Pytorch [62] and Gym [17], in order to avoid writing new code to implement known algorithms. Mushroom architecture is modular, so that it is possible to include only the needed modules to a system and to implement only the modules that may be needed.

## A.1  Related works

Despite the success that RL has acquired in the last decade, there are still no standard libraries to perform RL experiments. A common interface called RL-Glue [86] has been proposed in order to allow the communication of agents and environments written in different frameworks and programming languages. However, being only an interface, this library lacks of implementation of RL algorithms; furthermore, it is mainly focused on online RL algorithms and it is not supported anymore. Nevertheless, a large number of custom libraries are available, but they suffer from heterogeneous drawbacks: poor documentation, lack of algorithms, being very specific for certain tasks or simply no more maintained.

RLPy [35] is a good choice to do RL experiments with classical algorithms and problems, but currently it does not feature anything related to DRL; this is an important drawback considering the importance and the ongoing focus of the research on this topic. RLLab [28] is a more modern alternative to RLPy and a good choice for researchers focusing on continuous DRL. Other recent libraries mostly focused on DRL However, this library does not cover classical RL methodologies and this results in a library useful only for a limited number of researchers. More recent DRL libraries are TensorForce [69] and ChainerRL [19], that are RL extensions of deep learning libraries.

Eventually, there are many libraries featuring a single algorithm. For instance, simple-dqn [84] implements the DQN algorithm giving the possibility to replicate the experiments in [55] and the low complexity of the code allows to significantly customize it. Of course, the usefulness of such a library is limited to projects related to DQN.

## A.2  Ideas and Concepts

Mushroom is an easily accessible, yet powerful, RL library useful for research and didactic purposes.

**General purpose**  Mushroom adapts to heterogeneous learning tasks based on the interaction of an agent with an environment. This is achieved by a common interface shared by different algorithms, even suitable for different problems: batch and online algorithms, episodic and infinite horizon tasks, on-policy and off-policy learning, and many others.

**Lightweight**  Mushroom is both user-friendly and flexible: only a high-level interface is exposed to the user, hiding low-level aspects. For instance, the user should not care about the implementation details to use a function regressor for different tasks, since they are hidden by a simple common interface. However, we leave the check of consistency constraints to the user, e.g. avoiding the use of a tabular algorithm for an environment with continuous state space. Minimal interfaces simplify the implementation of new algorithms, as there are no hard constraints in the prototypes.

**Compatible**  Standard Python libraries useful for RL tasks have been adopted:

- *Scientific calculus*: numpy, scipy;
- *Basic ML*: scikit-learn;
- *RL benchmark*: gym;
- *Neural networks and GPU computation*: pytorch, tensorflow, theano;
- *Plotting*: matplotlib.

Mushroom provides an interface to these libraries, in order to integrate their functionalities in the framework, e.g. an interface for gym environments, support for regression with scikit-learn models.

**Easy to use**  Mushroom enables to develop and run experiments writing a minimal amount of code. In most of the tasks an experiment can be written in a few Python lines without the need of complex configuration files. The majority of the RL problems can be solved with experiments written following the structure of the library examples.

## A.3  Design

The main module of Mushroom is the `Core` whose purpose is to manage the interaction between the agent and the environment for both learning and testing tasks. The common interface `Environment`, extended by each problem implementation, contains the main properties of the environment: discount factor, horizon (i.e. maximum episode length), observation and action spaces. Each learning algorithm must extend the `Agent` class which also provides a common interface to interact with the environment following a specified policy, e.g. a $\varepsilon$-greedy policy. An instance of the `Core` class is built providing an agent and an environment object. It defines the `learn` and the `evaluate` methods, which implement the interaction of the agent on the environment.

The `evaluate` method runs the current policy on the environment and collects the dataset. The `learn` method also feeds the collected dataset to the learning algorithm of the agent, when needed. Both these methods can be run for a fixed number of episodes or steps; moreover, `learn` allows to specify the frequency, in terms of episodes or steps, of the call to the learning algorithm. Thanks to this choice, it is possible to implement online and batch algorithms transparently. The `Core` can also run a list of callback functions after each learning step.
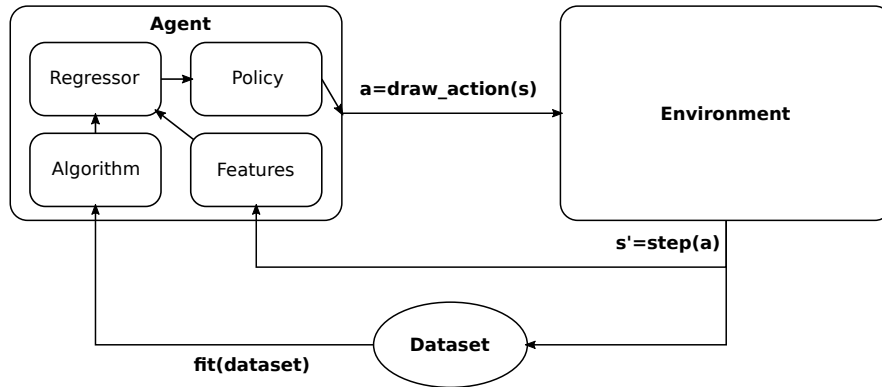
**Figure A.1:** *Interaction between the agent and the environment when the* `learn` *method of the* `Core` *class is called. A dataset, collected during this interaction, is used to update the approximator (e.g. policy, Q-function).*

```
# Online learning running for $30$ episodes
core.learn(n_steps=30, n_steps_per_fit=1)
# Batch learning over a dataset of 30 episodes
core.learn(n_episodes=30, n_episodes_per_fit=30)
# Online learning for 300 episodes, learning every 50 step
core.learn(n_episodes=300, n_steps_per_fit=50)
```

To simplify the implementation of generic RL algorithms, and TD algorithms in particular, Mushroom offers a high-level interface for the function regressors. This interface supports the use of ensemble regressors that are created specifying the number of models in the ensemble. Moreover, it manages the regressor of the *Q*-function in the case of discrete action spaces: it transparently deals with the creation of a different regressor for each action or a single regressor with a different output for each action. The user should not care about the low-level implementation of these regressors, since everything is managed by the `Regressor` interface, which also supports generic function approximator. Furthermore, both parametric and non-parametric are transparently managed by the interface.

```
# $Q$-function regressor with $5$ different linear approximator
approximator = Regressor(LinearApproximator, n_actions=5, output_shape=(1,),
    ...)
# $Q$-function regressor with a single linear approximator with 5 outputs
approximator = Regressor(LinearApproximator, n_actions=5, output_shape=(5,),
    ...)
# Generic linear approximator
approximator = Regressor(LinearApproximator, output_shape=(1,), ...)
```

The same logic used to implement the `Regressor` interface, has been used in the `Features` interface that has the purpose to transparently manage a generic set of basis functions, sparse features (e.g. tiles) and GPU-accelerated basis functions implemented with Pytorch [62].