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

ON THE EXPLOITATION OF UNCERTAINTY TO IMPROVE MAXIMUM EXPECTED VALUE ESTIMATE 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. Marcello Restelli

2019 – XXXI cycle

Abstract

ABSTRACT goes here.

Summary

SUMMARY goes here.

Contents

Glossary **XIII**

1 Introduction **1**
1.1 Perception and interaction 1
1.2 Learn how to act with Reinforcement Learning 2
1.2.1 Uncertainty in Reinforcement Learning 2
1.2.2 Balancing exploration and exploitation 3
1.3 My research 3

2 Preliminaries **5**
2.1 Agent and environment 5
2.2 Markov Decision Processes 6
2.2.1 Value functions 7
2.3 Solving a MDP 8
2.3.1 Dynamic Programming 8
2.3.2 Reinforcement Learning 10

3 Maximum Expected Value estimation **11**
3.1 Problem definition 12
3.1.1 Related Works 12
3.2 Weighted Estimator 13
3.2.1 Generalization to Infinite Random Variables 14
3.3 Maximum Expected Value estimation in Reinforcement Learning . . . 16
3.3.1 Online 16
3.3.2 Batch 17
3.3.3 Deep Reinforcement Learning 18
3.4 Empirical results 19
3.4.1 Discrete States and Action Spaces 20
3.4.2 Continuous state spaces 25
3.4.3 Deep Reinforcement Learning Scenario 27

4 Exploration 31

5 Deep 33

6 Mushroom 35

7 Conclusion 37

Bibliography 39

List of Figures

1.1 Reinforcement Learning problem scheme 2

2.1 Reinforcement Learning problem scheme 5
2.2 Markov Decision Process 7

3.1 MSE for each setting. Results are averaged over 2000 experiments. . . 20
3.2 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. 21
3.3 Grid world results with the three reward functions averaged over 10000 experiments. Optimal policy is the black line. 23
3.4 Profit per year averaged over 100 experiments. 24
3.5 Mean bias obtained by ME, DE and WE_∞ with different sample sizes and bins (only for ME and DE). 26
3.6 Variance of the bias obtained by ME, DE and WE_∞ with different sample sizes and bins. 27
3.7 Average reward averaged on 100 experiments. 28
3.8 Average reward averaged on 10 experiments. 29

List of Tables

3.1 Average reward in continuous action MDP. 30

List of Algorithms

1 Iterative Policy Evaluation 9
2 Value Iteration 9
3 Weighted Q-learning 17
4 Weighted FQI (finite actions) 18
5 Weighted FQI_∞ (continuous actions) 18

Glossary

D

DL Deep Learning. 4

DRL Deep Reinforcement Learning. 4

M

MAB Multi-Armed Bandit. 3

MDP Markov Decision Process. 6, 7

ML Machine Learning. 2

R

RL Reinforcement Learning. 2–5

CHAPTER 1

Introduction

EVERYONE experiences the process of taking 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 might 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, during the centuries, a strong debate between scientists, biologists, and even philosophers.

1.1 Perception and interaction

We start from the assumption that, by definition, an individual perceive the environment around it and acts on it in order to achieve *goals* expressed by its will. In other words, all the actions made by an individual are done to satisfy its will to obtain something from the world it lives in. This task is naturally performed by humans, but it implies some challenging problems that are hard, or unfeasible, to solve. One of them comes 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 senses, but the process of information retrieval by senses and the mental processes to understand them, inevitably introduce a certain level of noise that distorts the original true information. On the other hand, the interaction with the world deals with the will of the individual to perform actions to change the environment around it, but this apparently simple operation involves complex biologic 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

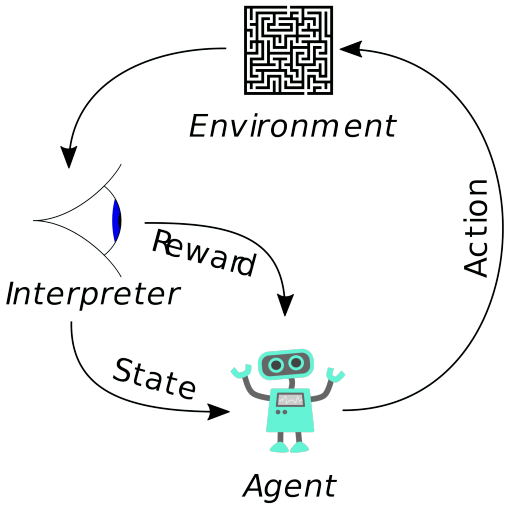


Figure 1.1: *The scheme of a RL model.*

result in performing actions without being sure of what it wants. It is arguable that discussing about the concept of true information and the concept of will requires strong theoretical considerations since they are both hardly definable 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 all this thesis, we lighten the definition of them to the one provided by common sense.

1.2 Learn how to act with Reinforcement Learning

Reinforcement Learning (RL) [20] is a subfield of Machine Learning (ML) which 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 providing the agent with the perception of its *state* in the environment and making it learn the appropriate *action* to perform, where an action can be seen as an atomic operation that brings the agent from a state to another one. The objective function represents a measure of how well the agent is accomplishing its task in the environment and it 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 which decides whether a certain behavior has to be reinforced (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 the uncertainty. In fact, initially, the agent is not provided with the knowledge of how the environment will react to its action, 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 the desired 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 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 field of Multi-Armed Bandit (MAB), a particular case of the RL problem with a single state [11]. In this problem the goal is to find the sequence of optimal actions, 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, 6, 25], and several algorithms to address this problem have been proposed such as UCB1 [2] and Thompson sampling [21].

The RL setting complicates the MAB problem because of the presence of multiple states. This makes the exploration-exploitation dilemma less tractable in terms of complexity and computational feasibility. Indeed, the quality of the actions must now be evaluated for each state, contrarily 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 a RL algorithm depends on its ability of reducing the uncertainty of the agent in a computationally and data-efficient way. The RL literature contains lots of algorithms and methodologies proposed to make the agent learn a good policy aiming at efficiency; however, despite addressing the reduction of uncertainty via experience, only few of them explicitly exploit uncertainty to learn.

During my Ph.D., I studied ways to develop algorithms that exploit uncertainty since the explicit consideration of uncertainty has been shown to be often helpful in order to improve the performance and efficiency of learning. One of the most common technique to explore is known as ε-greedy and consists in performing, at each state, a random action with probability ε and the action considered to be the best one with probability $1-\varepsilon$. This exploratory policy does not consider the uncertainty of the agent

and simply randomly moves it with the drawback of requiring a huge amount of experience to learn effective policies. This is shown especially in recent works on the field of Deep Reinforcement Learning (DRL) [13,24,26] which studies the application of Deep Learning (DL) models and methodologies to exploit their strong ability to generalize with the purpose to solve highly complex problems that were unfeasible before. Research on DRL, brought to the realization of groundbreaking works where authors have been able to reach the state-of-the-art in extremely complex games such as Go [16, 18] and chess [17].

The extraordinariness of these results is comparable to the amount of experience required by these algorithms to work. For instance, in [13] the experiments are performed using 50M samples corresponding to three days of computation and weeks of human play. This work does not address the problem of data-efficiency aiming more to other goals (e.g. maximizing the cumulative reward) and for this reason the previously described exploration policy of ε-greedy is used. However, data-efficiency can be pursued exploiting uncertainty in order to balance the knowledge of the agent of already known states and unknown ones, for instance letting it explore unknown states with higher probability. Moreover, the exploitation of uncertainty can also help to improve other aspect involved in learning algorithms which will explained more in details during the thesis.

My Ph.D. research brought to the publication of four conference papers, most of them focused on the previously described topic. I also developed, together with a colleague of mine, a RL Python library which had the initial purpose to facilitate my research, but which has become larger and larger allowing now to do RL research for general purposes. Other works are still ongoing and others have not been accepted for publications, still I think they worth to be mentioned in this thesis anyway. The whole document is composed of seven chapters, with this introduction being the first of them:

- **second** chapter resumes the main concepts of RL starting from the fundamental theory behind it and then giving a description of several methodologies related to this thesis. This chapter has the purpose to provide a general, but useful, overview of what is necessary to understand the following chapters;
- **third** chapter describes three publications I made about ways to exploit uncertainty in the context of Value-Based RL and more in particular in the famous algorithm of Q-Learning;
- **fourth** chapter deals with the description of novel algorithms that exploit uncertainty in order to improve exploration;
- **fifth** chapter extends the previous work to the DRL framework;
- **sixth** chapter provides a description of the RL Python library I developed;
- **seventh** chapter concludes the thesis resuming the previous chapters and providing my considerations about the research I made and the one I think will be interesting to pursue in the following years, by me or someone else!

CHAPTER 2

Preliminaries

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

2.1 Agent and environment

The interaction of an agent inside an environment can be seen as the execution of actions to move itself and observing 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_t of the environment, executes an

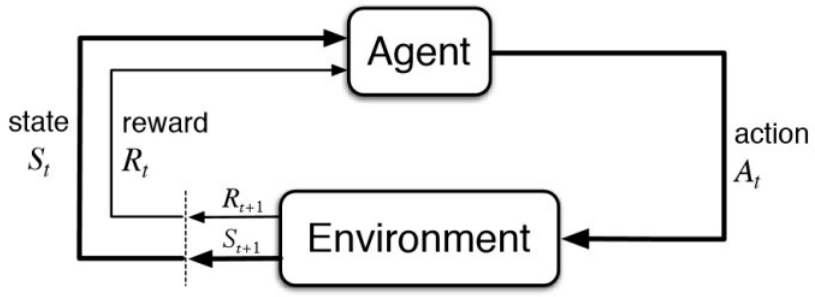


Figure 2.1: *The scheme of a RL model.*

action A_t and observes the new representation of the environment S_{t+1}. The problems about observation and interaction discussed in Chapter 1 are simplified by an explicit selection of data to observe from the environment and of the executable actions. In this way, only the relevant aspect of the sensory data acquired from the environment the agent are used. Together with S_{t+1}, the agent also sees a return R_t which is not given by the environment, but is a measure considered by the agent to evaluate the convenience of the consequences of the actions it executes. The total number of time steps 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 can terminate before reaching the horizon, which happens when the agent reaches special situations called *absorbing* states. These states are usually desirable or catastrophic states when the interaction of the agent with the environment is no more useful or impossible. The set of steps between the start of the interaction to the end is called *episode*.

The interaction of the agent with the environment is performed with the purpose 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* and is a function usually returning a real scalar value given the observation of the current state of the agent. The 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 to give different importance to rewards w.r.t. the time step they have been collected. If $\gamma = 1$ the equation reduces to 2.1, whereas the smaller it becomes the less the agent cares about rewards far in time.

2.2 Markov Decision Processes

The mathematical framework to study the interaction of the agent with the environment is provided by the theory behind Markov Decision Processes (MDPs). A MDP is defined as a 6-tuple where $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma, \mu >$:

- $\mathcal{S}$ is the set of states where the agent can be in the environment;

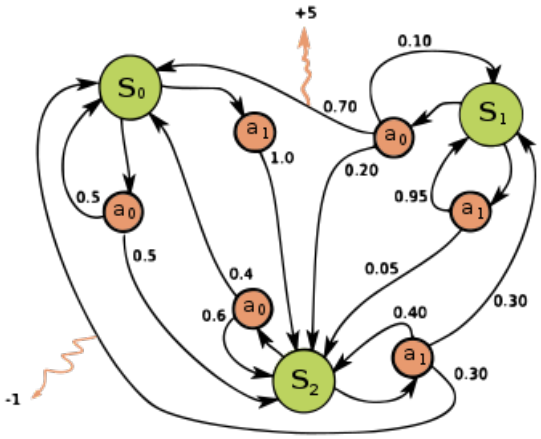


Figure 2.2: ...

- $\mathcal{A}$ is the set of actions that the agent can execute in the environment;
- $\mathcal{R}$ is the set of rewards obtainable by the agent;
- $\mathcal{T}$ is the *transition function* consisting in the probability of reaching a state s' executing action a in state s: $\mathcal{T}(s, a) = P(s'|s, a)$;
- γ is the discount factor;
- μ is the probability of each state to be the initial one: $\mu(s) = P(s_0 = s)$.

A MDP is called *finite*, or *discrete*, if the set of states $\mathcal{S}$ and 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 the MDPs:

- **stationarity:** the transition function $\mathcal{T}$ does not change over time;
- **Markovian assumption:** the transition and reward function depend only on the current time step and not on the previous ones.

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

2.2.1 Value functions

Recalling that the goal of the agent is to maximize the cumulative (discounted) reward obtained during an episode, a MDP is considered *solved* when the agent learns the actions to perform in each state which maximizes this measure. The function defining the probability of executing action a in a state s is called *policy*: $\pi(s) = P(a|s)$. An *optimal* policy π^* is the one which, when followed, allows the agent to solve the MDP. Considering the stochasticity in the transition function $\mathcal{T}$ and in the policy π, the expected value of the cumulative discounted reward obtainable following π is called *state value function*:

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

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:

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

And subsequently the optimal policy maximizes also the action value function of 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 a 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 case to the hardest ones.

2.3.1 Dynamic Programming

When the transition function $\mathcal{T}$ and reward function $\mathcal{R}$ of a 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 which return would obtain, but constitutes an interesting scenario to start studying the problem of solving a MDP. The theory behind the solving MDPs with full model available is known under the name of Dynamic Programming (DP) [4, 5]. The main concept in the research on DP is the optimal Bellman equation, defined as

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}[R_t + \gamma V^*(s') | S_t = s, A_t = a, S_{t+1} = s'] \\ &= \max_a \sum_{s'} p(s'|s,a)[r + \gamma V^*(s')] \end{aligned} \tag{2.8}$$

for state value function, and

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}[R_t + \gamma \max_{a'} Q^*(s',a') | S_t = s, A_t = a, S_{t+1} = s'] \\ &= \sum_{s'} p(s'|s,a)[r + \gamma \max_{a'} Q^*(s',a')] \end{aligned} \tag{2.9}$$

for action value function, for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$. The optimal Bellman equation 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 Bellman equation in form of an assignment repeated iteratively.

Algorithm 1 Iterative Policy Evaluation

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

Algorithm 2 Value Iteration

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

Policy Iteration

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

$$\begin{aligned} V_{t+1}(s) &= \mathbb{E}_\pi[R_t + \gamma V_t(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_s P(s'|s,a)[r + \gamma V_t(s')] \end{aligned} \tag{2.10}$$

for all $s \in \mathcal{S}$. It can be shown that the iterative application of the Bellman equation always converges to a single fixed point V_π.

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

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[R_t + \gamma V_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s'} P(s'|s,a)[r + \gamma V_\pi(s')]. \end{aligned} \tag{2.11}$$

Whenever $Q_\pi(s,a) > Q_\pi(s,\pi(s))$ it is convenient to update the policy such as $\pi(s) = a$. This procedure is called *policy improvement*. The process of alternating steps of iterative policy evaluation and policy improvement brings to the estimation of the optimal value functions and is resumed in an algorithm called *Policy Iteration* (Algorithm 2).

Value Iteration

The alternation of policy evaluation and policy improvement is a drawback of Policy Iteration which may slowdown the learning. Among other algorithms, the algorithm of *Value Iteration* addresses this problem stopping policy evaluation after only one 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:

$$\begin{aligned} V_{t+1}(s) &= \max_a \mathbb{E}[R_t + \gamma V_t(S_{t+1}) | S_t = s] \\ &= \max_a \sum_s P(s'|s,a)[r + \gamma V_t(s')] \end{aligned} \tag{2.12}$$

for all $s \in \mathcal{S}$. Desirably, Value Iteration maintains the properties of Policy Iteration about 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 model of the MDP is known. However, in most of real cases the full model is not available and the agent must move in the environment in order to understand it.

2.3.2 Reinforcement Learning

Online

Batch

Deep Reinforcement Learning

CHAPTER 3

Maximum Expected Value estimation

The computation of the Maximum Expected Value (MEV) is required in several applications. Indeed, almost any process of acting involves the optimization of an expected utility function. For example, in the daily 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 computation 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, on its turn, depends on the MEVs of the actions available in the reached states. Since errors propagate through all the state-action pairs, a bad estimator for the MEV negatively affects the speed of learning [22].

Generally, we use this estimation when we need to know not only which is the variable with the MEV from a set of variables, but we want to have also a good estimation of such expected value. The most used approach to this estimation problem is the Maximum Estimator (ME) which simply takes the maximum estimated utility. As proved in [19], this estimate is positively biased and, if used in iterative algorithms, can increase the approximation error step-by step [22]. More effective estimators have been proposed in the recent years. The Double Estimator (DE) [23] approximates the maximum by splitting the sample set into two disjoint sample sets. One of this set is used to pick the element with the maximum approximate value and its value is picked from the other set. This has to be done the opposite way 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 negative bias [23] which, in some applications, allows to overcome the problem of ME.

During my research, I analyzed this problem and proposed the Weighted Estima-

tor (WE) [8] which approximates the maximum value 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 finite random variables, thus, in a subsequent work, I extended the study to problems with infinite random variables and proposed an extension of WE [7] to address them. By exploiting the Central Limit Theorem (CLT) and the spatial correlation among variables, the probability distribution of each random variable is approximated as a normal distribution whose means and variances are estimated by means of Gaussian Processes (GPs) [15].

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 μ_i its mean, and with σ_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 if 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 to estimate 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 [19], this estimator has a positive bias that may cause issues in applications of ME, such as in the RL algorithm of Q-Learning where the overestimation of the state-action values due to the positive bias can cause an error that increases step by 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 [22] and theoretically analyzed in [23]. DE uses a sample set S retrieved by the true unknown distribution like ME, but splits it in 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 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

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

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 to be the maximum. Since these probabilities sum to one, the approximation given by DE results in a value that is lower than or equal to the maximal expected value. Even if the underestimation does not guarantee better estimation than the ME, it can be helpful to avoid an incremental approximation error in some learning problems. For instance, Double Q-Learning [22] is a variation of Q-Learning that exploits this technique to avoid the previously described issues due to overestimation. Double Q-Learning has been tested in some very noisy environments and succeeded to find better policies than Q-Learning. Another remarkable application of DE is presented in [28] 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 average of the variable that is estimated to be the one with the largest mean, the proposed WE estimates the MEV $\mu_*(X)$ computing a weighted mean of all the sample averages:

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

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

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

We know that the sample mean $\hat{\mu}_i(S)$ is a random variable whose expected value is μ_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 that is assumed to be unknown. In particular, if X_i is normally distributed, then, independently 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, by the 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, independently of the distribution of X_i. Leveraging on 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 μ_i, and the variance of the normal distribution $\tilde{f}_i$ tends to zero, so that the weights of the variables with expected value less than $\mu_*(X)$ go to zero, so that $\hat{\mu}_*^{WE} \to \mu_*(X)$.

3.2.1 Generalization to Infinite Random Variables

As far as we know, 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 variables in $\mathcal{Z}$ have some spatial correlation. Here, we consider $\mathcal{Z}$ to be a closed interval in $\mathbb{R}$ and that each variable $z \in \mathcal{Z}$ has unknown mean μ_z and variance σ_z^2. Given a set of samples S we assume to have an estimate $\hat{\mu}_z(S)$ of the expected value μ_z for any variable $z \in \mathcal{Z}$ (in the next section we will discuss the spatial assumption and we will explain how to obtain this estimate). As a result, the weighted sum of equation 3.4 generalizes to an integral over the 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*, that 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:

$$\begin{aligned}
\mathfrak{f}_z^*(S) &= f\left(\hat{\mu}_z(S) = \sup_{y\in\mathcal{Z}} \hat{\mu}_y(S)\right) \\
&= \int_{-\infty}^{\infty} f(\hat{\mu}_z(S)=x)\; P\Big(\hat{\mu}_y(S) \le x,\; \forall y \in \mathcal{Z}\setminus\{z\}\Big)\mathrm{d}x \\
&= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\; P\left(\bigwedge_{y\in\mathcal{Z}\setminus\{z\}} \hat{\mu}_y(S) \le x\right)\mathrm{d}x \qquad (3.8)\\
&= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\; \frac{P\left(\bigwedge_{y\in\mathcal{Z}} \hat{\mu}_y(S) \le x\right)}{P\left(\hat{\mu}_z(S)\le x\right)}\mathrm{d}x \qquad (3.9)\\
&= \int_{-\infty}^{\infty} f_{\hat{\mu}_z}^S(x)\; \frac{\prod_{\mathcal{Z}} F_{\hat{\mu}_y}^S(x)^{dy}}{F_{\hat{\mu}_z}^S(x)}\mathrm{d}x
\end{aligned}$$

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 defined in the geometric calculus (that is the generalization of the product operator to continuous supports) and can be related to the classical calculus through the following relation: $\prod_{\mathcal{Z}} F_{\hat{\mu}_y}^S(x)^{dy} = \exp\left(\int_{\mathcal{Z}} \ln F_{\hat{\mu}_y}^S(x) dy\right)$ [9].

Spatially Correlated Variables

The issues that remain to be addressed are I) the computation of the empirical mean $\hat{\mu}_z(S)$ and II) 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 have 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.

In order to face the second issue, we need to restrict the regression class to 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* $\big($i.e. $\hat{\mu}_z, \hat{\sigma}^2_{\hat{\mu}_z} \leftarrow g(z)\big)$. 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 $\tilde{f}_{\hat{\mu}_z}^S = \mathcal{N}\left(\hat{\mu}_z, \hat{\sigma}^2_{\hat{\mu}_z}\right)$.

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

$$\hat{\mu}_*^{\mathrm{WE}}(S) = \int_{\mathcal{Z}}\int_{-\infty}^{\infty} \frac{\hat{\mu}_z(S)\tilde{f}_{\hat{\mu}_z}^S(x)}{\tilde{F}_{\hat{\mu}_z}^S(x)} e^{\int_{\mathcal{Z}} \ln \tilde{F}_{\hat{\mu}_y}^S(x)\mathrm{d}y}\mathrm{d}x\mathrm{d}z. \qquad (3.10)$$

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 process

mean and variance. Consider to have 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 σ_n^2, the GP approximation for a certain variable z^* is $q_* \sim \mathcal{N}\left(\hat{\mu}_{z^*}, \hat{\sigma}^2_{\hat{\mu}_{z_*}} + \sigma_n^2 I\right)$ where:

$$\begin{aligned} \hat{\mu}_{z_*} &= \mathbf{k}_*^T \left(K + \sigma_n^2 I\right)^{-1} \mathbf{q}, \\ \hat{\sigma}^2_{\hat{\mu}_{z_*}} &= \mathrm{Cov}\left(\mu_{z_*}\right) = k(z_*, z_*) - \mathbf{k}_*^{\mathrm{T}} \left(K + \sigma_n^2 I\right)^{-1} \mathbf{k}_*, \end{aligned} \tag{3.11}$$

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 3.11, 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 ($\hat{\sigma}^2_{\hat{\mu}_{z_*}}$) and the variance of the noise (σ_n^2) [15]. As a result, by only considering the mean contribute, we approximate the distribution of the sample mean by $\tilde{f}^S_{\hat{\mu}_z} = \mathcal{N}\left(\hat{\mu}_z, \hat{\sigma}^2_{\hat{\mu}_z}\right)$ as defined in equations 3.11.

3.3 Maximum Expected Value estimation in Reinforcement Learning

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

3.3.1 Online

A famous online algorithm is Q-learning [27], an off-policy algorithm that updates the action value function Q at each step using the following formula:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right), \tag{3.12}$$

where $\alpha_t(s_t, a_t)$ is a learning rate, γ is a discount factor and r_t is the immediate reward obtained by taking action a_t in state s_t. It is demonstrated that since Q-learning is a stochastic approximation algorithm, under the assumption that each state-action pair is visited infinitely often and the step sequence satisfies certain conditions, Q_k converges to Q^* [27]. However, under particular conditions, such as a wrong tuning of parameters and noisy environments, the Q-function could converge to Q^* too much slowly. One of the main reasons is that the Q-learning uses the ME to estimate the current maximum Q-value of the next state s_{t+1}. 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 could fail. In the last years, different approaches have been proposed trying to overcome this issue [3, 12, 29]. In particular, the most successful one is the Double Q-Learning algorithm [22] which replaces the ME used in Q-Learning with DE. The underestimation of the Q-function performed by Double Q-Learning allows to learn a good policy in very noisy environments where Q-Learning fails.

Algorithm 3 Weighted Q-learning

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

Weighted Q-Learning

Recently, the replacement of ME with WE has been proposed in the *Weighted Q-Learning* algorithm [8]. Weighted Q-Learning maintains an estimate of the mean value of the Q-function and its variance in order to compute the weights of WE (Algorithm 3). 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 showed that the variance can be computed incrementally at each step t with:

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

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'.

3.3.2 Batch

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

$$\mathcal{D}_t = \left\{ \langle (s_i, a_i), r_i + \gamma \max_{a'} Q_{t-1}(s_i', a') \rangle \right\}_{1 \leq i \leq N}. \tag{3.13}$$

The FQI update, similarly to the Q-Learning update (see equation **??**), requires the computation of ME which causes the same overestimation problem of Q-Learning. Intuitively, the replacement of ME with DE or WE can help to solve this issue also in

Algorithm 4 Weighted FQI (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 5 Weighted FQI$_\infty$ (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

FQI. The FQI variant which replaces ME with DE is called Double Fitted Q-Iteration (DFQI), and the variant which uses WE is called Weighted Fitted Q-Iteration (WFQI) [7].

Weighted Fitted Q-Iteration WFQI uses GP regression in order to compute the mean Q-value and its variance in continuous state spaces (Algorithm 4). The interesting aspect of WFQI is that it can handle infinite action spaces too, as explained in Section 3.2.1 and showed in Algorithm 5. At the best of our knowledge, this makes it the only value-based algorithm able to deal with infinite action spaces.

3.3.3 Deep Reinforcement Learning

In the last few years, value-based RL algorithms exploiting deep neural networks for Q-function approximation proved to be a very powerful way to solve complex highly dimensional MDPs. The most famous algorithm is the Deep Q-Learning algorithm [13], more known as Deep Q-Network (DQN) algorithm, where the Q-function is approximated with a deep neural network in an online setting. DQN consists of a neural network to be trained online and another one that builds the target of the previous one. The target network is used for stability reasons, and it is updated with the weights of the online network every time a specified number of samples have been collected. The algorithm updates the online network using minibatch of samples collected using a ε-greedy policy, and stored in a replay memory, minimizing a loss function between the

current estimate of the target network and the following target:

$$\hat{Q} = \begin{cases} r_t & \text{if } s_{t+1} \text{ is an absorbing state} \\ r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases}.$$

where θ^- are the parameters of the target network. The overestimation problem caused by ME happens also in DQN and results in critical loss of performance in some problems due to instability. The Double Deep Q-Network (DDQN) algorithm [10] replaces ME with DE and shows considerable improvements and performance and stability.

Weighted Deep Q-Network The replacement of ME with WE is not straightforward in this case due to difficulties in computing the variance of the approximation. The GP regression, used in WFQI, is commonly not used in DRL and a deep neural network adapts better in important DRL problems, such as the well known Atari domain. The estimation of mean and variance with a neural network is possible, but the computational complexity increases with the number of parameters such that it becomes unfeasible in deep neural networks. 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) [14]. This work follows the DQN algorithm described in [13] with few, but important changes. The output of the neural network is split in K heads that share the same first hidden layers. To perform bootstrapping, a binary mask $w_1, \dots, w_K \in 0, 1$ 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 follow 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.

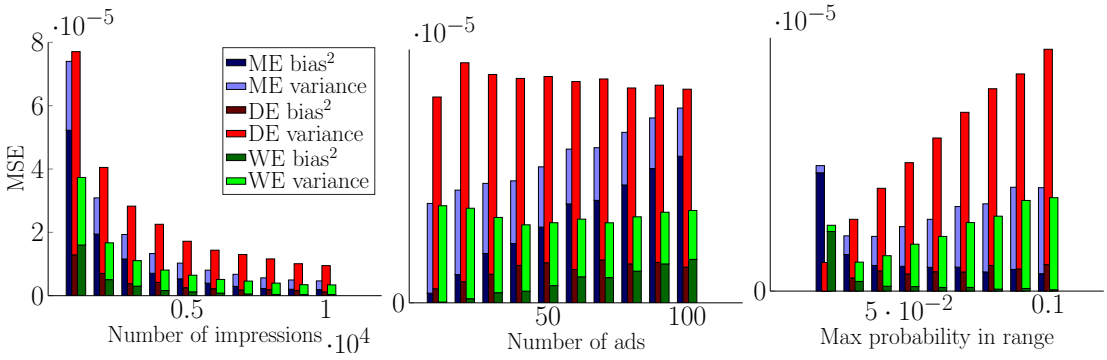
We propose to use the architecture of BDQN replacing the DE with WE and to approximate the weights of its update formula using the output of each head. The resulting update formula is:

$$Q^k_{t+1} \leftarrow Q^k_t + r_t + \gamma \sum_{i \in 1, \dots, \#\mathcal{A}} w^i_t Q^k_t(s_{t+1}, a_i; \theta^-_k), \tag{3.14}$$

where w^i_t is the weight of WE for action i at time t and θ^-_k are the parameters of the target network of the head k. Since the Q-values of the next state are computed using the target network, we propose to compute the weights using the online network in order to emulate the desirable behavior of DE. Note that the weights vector $\mathbf{w}_t = \langle w^1_t, w^2_t, \dots, w^K_t \rangle$ is the same for all heads. 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).

3.4 Empirical results

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



(a) Increasing number of impressions. (b) Increasing number of ads. (c) Increasing value of maximum CTR.

Figure 3.1: *MSE for each setting. Results are averaged over 2000 experiments.*

3.4.1 Discrete States and Action Spaces

Internet Ads

We consider this MAB problem as formulated in [23]. 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 fixed at 0.02.

In Figure 3.1, we show the $MSE = bias^2 + variance$ for the three settings comparing the results obtained by each estimator. In the first setting (Figure 3.1(a)) reasonably the Mean Squared Error (MSE) decreases for all 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 lack of samples (as showed also in Figure **??**). Figure 3.1(b) shows that an increasing number of actions has a negative effect on ME and a positive effect on the 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 3.1(c) when all the ads share the same click rate (0.02), where DE is

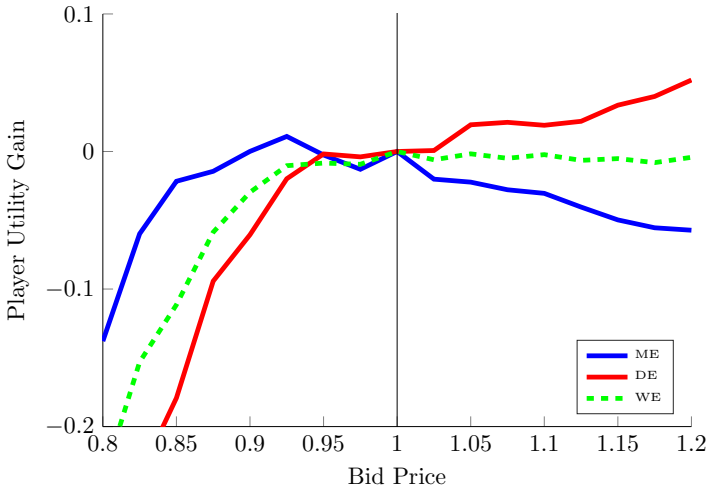


Figure 3.2: *Relative player 1 utility gain for different value of the bid defined as* $\frac{utility(b)}{utility(v)} - 1$. *Results are averaged over 2000 experiments.*

the best. However, it starts to have large variance as soon as the range of probabilities increases (Figure **??**). 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 [28]. 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 ρ of the ads. CTRs are generally unknown, therefore the search engine should use data to estimate which is the best ad (i.e., the one that maximizes $b \cdot \rho$) and the payment in case of click; reasonably, wrong estimations may significantly harm the revenue. On the other hand, the 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 that the advertisers maximize their utility by truthfully bidding $b_i = v_i$. Incentive compatibility may not occur when the estimate 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 per click value 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 are: $\rho = \{0.15, 0.11, 0.1, 0.05, 0.01\}$. CTRs are estimated from data collected using the UCB1 algorithm [2] in a learning phase consisting of 10000 rounds of exploration (i.e. impressions), as done in [28].

Figure 3.2 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,

[1]The debiasing algorithm proposed in [28] is a cross validation approach, but differs from the estimators considered in this paper. It averages the values used for selection and the values used for estimation, thus being a hybrid of DE and ME.

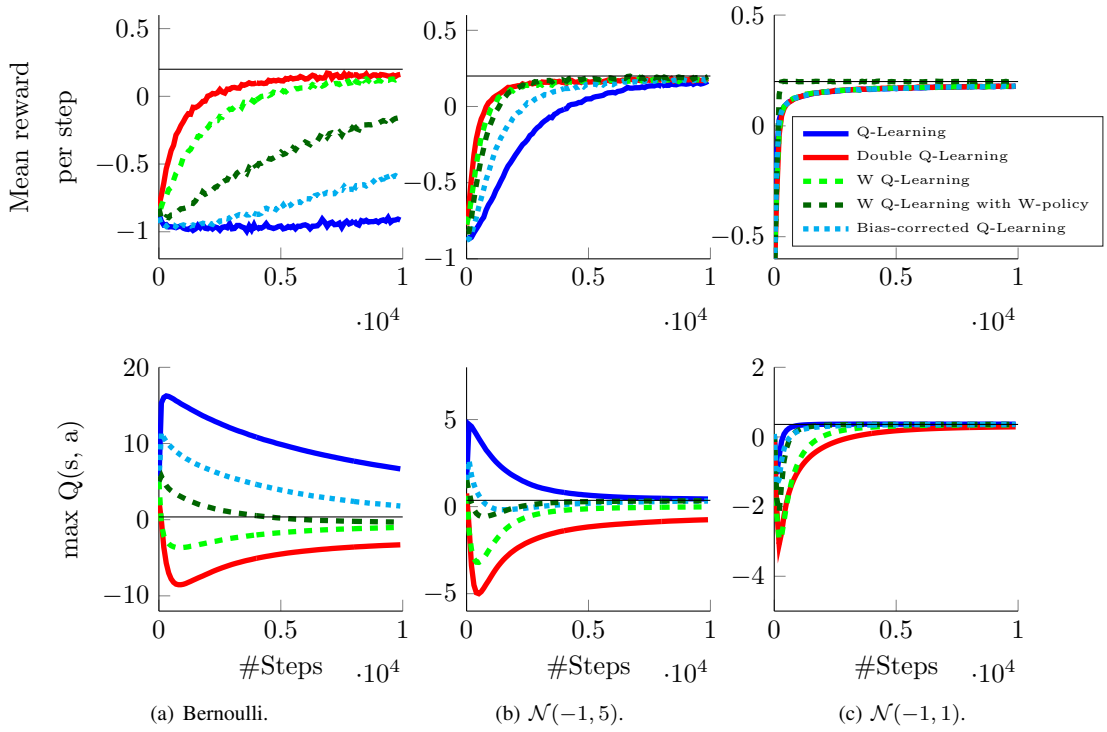
but there is an incentive to overbid using DE. Therefore WE is the only estimator which succeeds to achieve incentive compatibility.

Grid World

This simple MDP consists of a 3×3 grid world where the start state in the lower-left cell and the goal state in the upper-right cell [22]. In this domain, we compare the three estimators together with the performance of an algorithm called Bias-corrected Q-Learning, a modified version of Q-learning that, 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 [**?**, 12]. Moreover, we test Weighted Q-Learning also using a different policy, that we call *weighted policy*, which samples the action to perform in a state from the probability distribution of the weights of WE. We use an ε-greedy policy with $\varepsilon = \frac{1}{\sqrt{n(s)}}$ where $n(s)$ is the number of times the state s has been visited. 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 Double Q-Learning 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 $\sigma = 5$, 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 maximizing the Q-value is $5\gamma^4 - \sum_{k=0}^{3} \gamma^k \approx 0.36$. In Figure 3.3, 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 3.3(a) and 3.3(b) show that, regardless of the bad approximation of the Q-function, the underestimation of Double Q-Learning allows to learn the best policy faster than other algorithm in these noisy settings. Bias-corrected Q-Learning estimates the Q-function better than Double Q-Learning, but performs worse than the other algorithms, except for Q-Learning, when the variance is large. Weighted Q-Learning 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 3.3(c)). These good results are explained considering that the weighted policy is able to reduce the exploration faster than ε greedy due to exploiting of the good approximation of the Q-function computed by Weighted Q-Learning. It is worth to point out that Weighted Q-Learning works well for both Gaussian and Bernoullian rewards, showing that WE is effective even 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 environment with acknowledged hardly predictable dynamics that complicate the estimate of the Q-values and, therefore, of the expected profit. The MDP we build is a simplified



(a) Bernoulli. (b) $\mathcal{N}(-1, 5)$. (c) $\mathcal{N}(-1, 1)$.

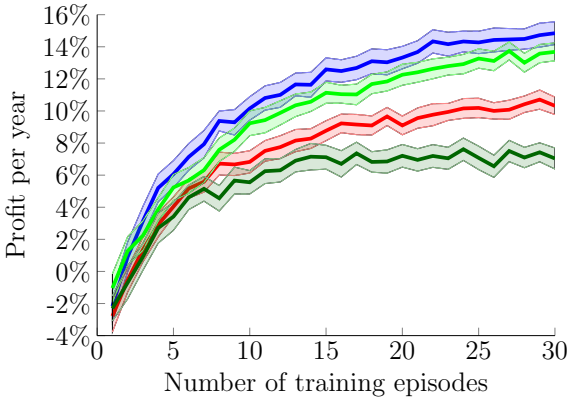
Figure 3.3: *Grid world results with the three reward functions averaged over 10000 experiments. Optimal policy is the black line.*

version of the real Forex market. In our Forex MDP the agent enters in the market always 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 from 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 $c(t-1)$:

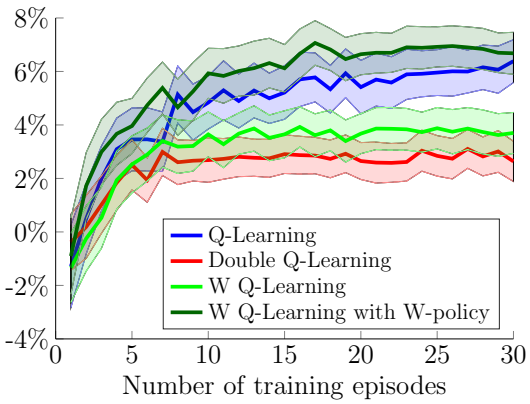
$$r(t) = a(t-1)(c(t) - c(t-1)) + 0.5 * spread|a(t) - a(t-1)|.$$

The same algorithms used in the grid world, except for Bias-Corrected Q-Learning, domain were trained using historical daily 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 learning rate $\alpha(s, a) = \frac{1}{n(s,a)}$, discount factor $\gamma = 0.8$ and $\varepsilon = \frac{1}{\sqrt{n(s)}}$.

Figure 3.7 shows the profit per year, w.r.t. the number of training episodes, of the four algorithms during the training phase (Figure 3.4(a)) and during a test phase where the learned policy is run with a greedy policy (Figure 3.4(b)). In the training phase an ε-greedy policy is used for Q-learning and Double Q-Learning, while Weighted Q-Learning uses both the ε-greedy policy and the weighted policy. During the training phase, Q-learning performs better than Double Q-learning and also than Weighted Q-



(a) Training phase.



(b) Test phase.

Figure 3.4: *Profit per year averaged over 100 experiments.*

learning. Interestingly, Weighted Q-learning with the weighted policy reaches the worst performance in the training phase, but it reaches the best performance in the test phase. This happens because more exploration is induced by the weighted policy w.r.t. the ε-greedy policy, resulting in bad performance during the training phase, but also in better estimates of the Q-values. Double Q-learning performs worse than Q-learning and Weighted Q-learning both in training phase and test phase. The reason is that in many states there is an action that is significantly better than the others, that represents the case where ME gives the best results, and the case where DE suffers the most.

3.4.2 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_∞) 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 be shown to the user ($p \in [0, 10]$). The reserve price τ , 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 τ_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 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 uniformly spaced bins. As shown in Figure 3.5, 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 for selecting the appropriate number of bins (not required by WE). WE is able to achieve the lowest or a 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 3.6 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 because this pricing problem is not iterated. From Figure 3.5 we can see that ME is prone to provide positive bias, while WE bias is almost always the smaller or stays between ME and DE. The reason for which the ME bias is not always positive, as stated by its theoretical property (for

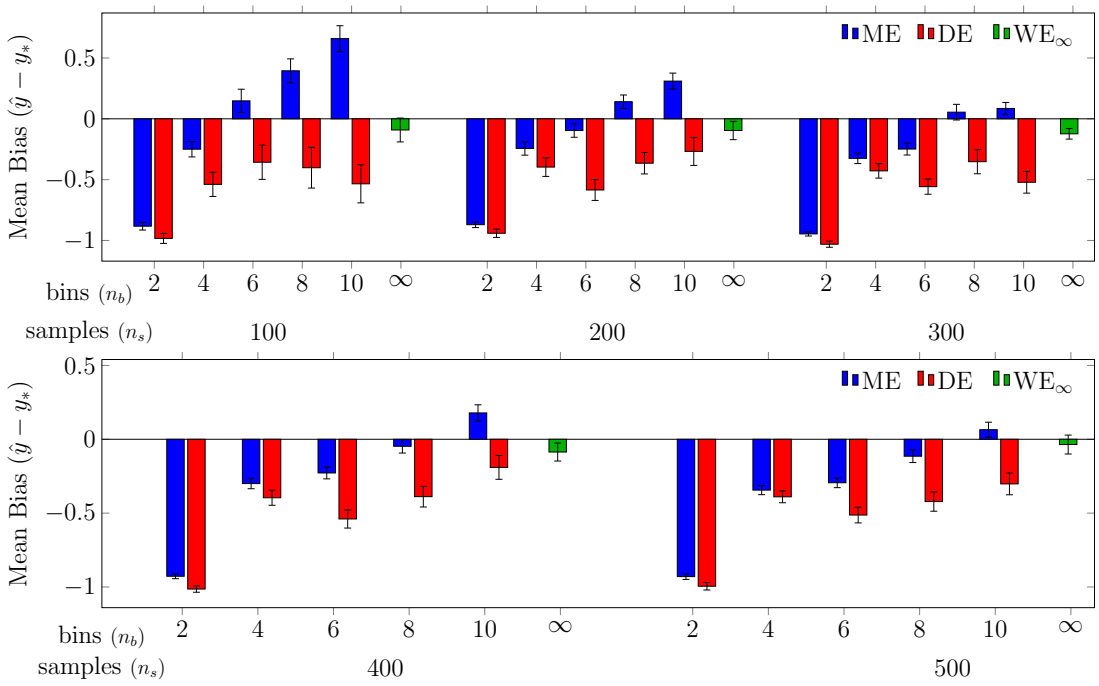


Figure 3.5: *Mean bias obtained by ME, DE and WE$_\infty$ with different sample sizes and bins (only for ME and DE).*

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 [**?**]. 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 θ is the angle and ω 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 $ml^2\dot{\omega} = -\mu\omega + mgl\sin(\theta) + u$, 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 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 $\theta_0 = \{\frac{2\pi k}{36} | k = \{0, 1, \ldots, 35\}\}$. 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. Results show that WFQI for discrete actions 4 and FQI are robust with respect to the number of episodes

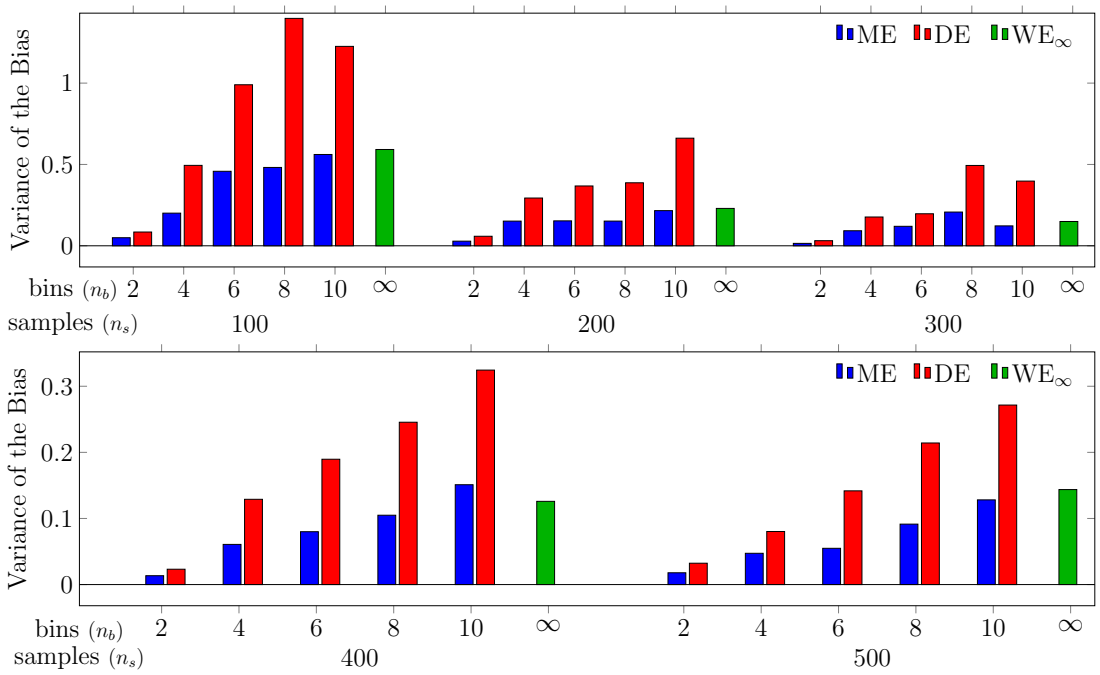


Figure 3.6: *Variance of the bias obtained by ME, DE and WE$_\infty$ with different sample sizes and bins.*

and WFQI reaches the highest average reward in each case (with statistical confidence obtained over 100 runs and level 95%). DFQI performance is reasonably poor with few examples since it uses a half of the training set to train each regressor. In the latter setting, the only algorithm that is able to directly handle continuous space is the WFQI defined in Algorithm 5. The other algorithms use a GP with 100 actions to approximate the maximum.

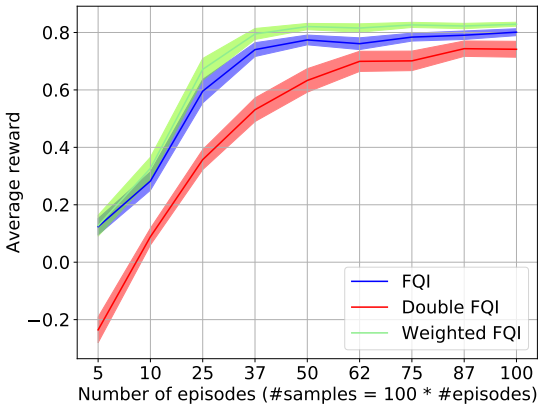
3.4.3 Deep Reinforcement Learning Scenario

Acrobot

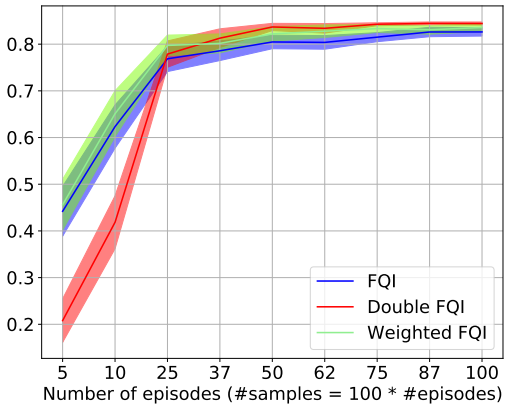
We evaluate the performance of BDQN with ME and DE and WDQN on the RL problem of Acrobot. This is a well-known problem consisting in swinging up a two-link robot over a certain threshold. The state space and the dynamics of the problem make it a complex MDP to be solved[2]. The reward of the MDP is -1 at each step and 0 when the arm of the bot reaches the threshold height. The discount factor is $\gamma = 0.99$. The horizon is set to 200. The training phase policy is the Bootstrapped policy described in Section 3.3.3, while the evaluation policy computes the best action through ensemble voting. The hyperparameters of DQN are the same used in the Atari experiments in [14], except for the ones specified in Table 3.1. The policy used is the Bootstrapped policy used in [14], where at the beginning of an episode a head is chosen randomly and the greedy policy derived by that head is followed. In our setting, we choose a random head at each step instead at each episode in order to favor exploration.

Figure 3.8 shows that all algorithms converges to the same performance, but WDQN achieves it considerably faster than the others. Moreover, the score obtained by WDQN

[2]We use the Acrobot-v1 environment of the OpenAI Gym library [**?**].



(a) Discrete actions.



(b) Continuous actions.

Figure 3.7: *Average reward averaged on 100 experiments.*

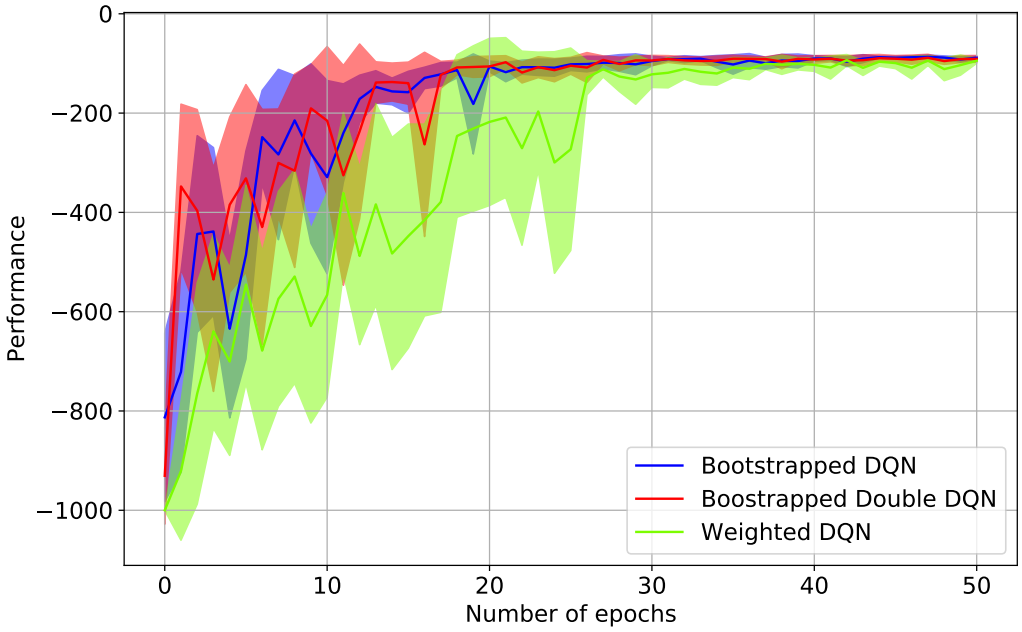


Figure 3.8: *Average reward averaged on 10 experiments.*

is more stable during the training epochs, showing that WE helps also to stabilize the learning which is an important issue in the DRL scenario.

Table 3.1: *Average reward in continuous action MDP.*

Replay memory size	50000
Initial replay size	5000
Agent history length	1
Target network update frequency	100
Masking probability p	$^2/_3$
Number of hidden layers	2
Number of neurons	80
Number of heads	10
Test samples	5000
Evaluation frequency	5000
Max no-op actions	0
Total number of steps	250000
Optimizer	Adam

CHAPTER 4

Exploration

CHAPTER 5

Deep

CHAPTER 6

Mushroom

CHAPTER 7

Conclusion

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

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

Bibliography

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

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

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

[26] 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.

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

[28] 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.

[29] 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.