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

2019 – XXXI cycle

Abstract

ABSTRACT goes here.

Summary

SUMMARY goes here.

Contents

Glossary XIII

I Starting Point 1

1 Introduction 3
- 1.1 Perception and interaction 3
- 1.2 Learn how to act with Reinforcement Learning 4
  - 1.2.1 Uncertainty in Reinforcement Learning 4
  - 1.2.2 Balancing exploration and exploitation 5
- 1.3 My research 5

2 Preliminaries 9
- 2.1 Agent and environment 9
- 2.2 Markov Decision Processes 10
  - 2.2.1 Value functions 11
- 2.3 Solving a MDP 12
  - 2.3.1 Dynamic Programming 12
  - 2.3.2 Reinforcement Learning 14

II Bellman Update 19

3 Maximum Expected Value estimation 21
- 3.1 Problem definition 22
  - 3.1.1 Related Works 22
- 3.2 Weighted Estimator 23
  - 3.2.1 Generalization to Infinite Random Variables 24
- 3.3 Analysis of Weighted Estimator 26
  - 3.3.1 Bias 26
  - 3.3.2 Variance 28
- 3.4 Maximum Expected Value estimation in Reinforcement Learning 29

3.4.1 Online 29
3.4.2 Batch 30
3.4.3 Deep Reinforcement Learning 31
3.5 Empirical results 32
3.5.1 Discrete States and Action Spaces 32
3.5.2 Continuous state spaces 37
3.5.3 Deep Reinforcement Learning Scenario 39

4 Exploiting uncertainty of the Bellman operator components to deal with highly stochastic problems 41
4.1 Preliminaries 42
4.2 The Proposed Method 42
4.2.1 Decomposition of the TD error 43
4.2.2 Analysis of the decomposed update 43
4.2.3 Variance dependent learning rate 44
4.2.4 Discussion on convergence 45
4.3 Experimental Results 46
4.3.1 Noisy Grid World 47
4.3.2 Double Chain 49
4.3.3 Grid World with Holes 50
4.3.4 On-policy learning 52

III Uncertainty-Driven Exploration 53

5 Thompson Sampling Based Algorithms for Exploration in Reinforcement Learning 55
5.1 Related work 56
5.2 Thompson Sampling in value-based Reinforcement Learning 57
5.3 Efficient uncertainty estimation 58
5.3.1 Online estimation 58
5.3.2 Bootstrapping 61
5.4 Experiments 62
5.4.1 Discrete state space 62
5.5 Other results 63
5.5.1 Continuous state space 63
5.5.2 Deep Reinforcement Learning 64

6 Exploration Driven by an Optimistic Bellman Equation 67
6.1 Learning Value Function Ensembles with Optimistic Estimate Selection 68
6.1.1 An Optimistic Bellman Equation for Action-Value Function Ensembles 68
6.1.2 Optimistic Value Function Estimators 72
6.2 Experimental Evaluation 74
6.2.1 Results 75

IV Final Considerations 77

7 Mushroom 79

8 Conclusion 81

Bibliography 83

List of Figures

1.1 Reinforcement Learning problem scheme 4

2.1 Reinforcement Learning problem scheme 9
2.2 Markov Decision Process 11
2.3 DQN network scheme 18

3.1 Bias analysis in WE 26
3.2 Absolute bias analysis in WE 26
3.3 Variance analysis in WE 28
3.4 MSE analysis in WE 28
3.5 Boostrapped DQN network 31
3.6 Internet ads results 33
3.7 Sponsored search auctions results 34
3.8 Grid world results 35
3.9 Forex results 36
3.10 Bias in pricing problem 37
3.11 Variance in pricing problem 38
3.12 Swing-up pendulum results 39
3.13 Acrobot results 40

4.1 Noisy grid world algorithms comparison 47
4.2 Noisy grid world RQ-Learning variants comparison - 1 48
4.3 Noisy grid world RQ-Learning variants comparison - 2 49
4.4 Structure of the double-chain problem. 49
4.5 Double chain problem results 50
4.6 Learning rate adaptation in double chain problem 50
4.7 Policy in double chain problem 51
4.8 Grid world with holes algorithms comparison - 1 51
4.9 Structure of the Grid World with Holes problem. 52
4.10 Grid world with holes algorithms comparison - 2 52

5.1 Taxi problem results 63
5.2 Mountain car and acrobot problems results 64
5.3 Taxi with different upper bounds results - 1 64
5.4 Taxi with different upper bounds results - 2 65
5.5 Taxi with different upper bounds results - 3 65
5.6 Pong and Breakout problems results 65

List of Algorithms

1 Iterative Policy Evaluation 13
2 Value Iteration 13
3 Weighted Q-learning 29
4 Weighted FQI (finite actions) 30
5 Weighted FQI$_\infty$ (continuous actions) 30
6 Standard mean and variance update 58
7 Mean and variance update using momentums 58
8 SARSA with online variance update 59
9 SARSA with online variance update and Hoeffding upper bound . . . 59
10 SARSA with function approximation with variance update 60
11 Bootstrapped DQN with Thompson Sampling 62
12 Optimistic Deep Q-Network 74

Glossary

B

BDQN	Bootstrapped Deep Q-Network. 16, 29, 30, 55, 59, 60, 62, 65, 70–74
BE	Bellman Operator. 66, 69, 70
BQL	Bootstrapped Q-Learning. 73

C

CDF	Cumulative Density Function. 20, 55, 56
CLT	Central Limit Theorem. 22, 23, 56
CTR	Click-Through Rate. 30–32

D

DDQN	Double Deep Q-Network. 16, 29, 37, 59
DE	Double Estimator. 19–21, 24–32, 35, 36
DFQI	Double Fitted Q-Iteration. 28, 36, 37
DL	Deep Learning. 6, 15
DP	Dynamic Programming. 10, 12, 14
DQL	Double Q-Learning. 21, 27, 32–34
DQN	Deep Q-Network. 15, 16, 29, 30, 37, 59, 65, 70, 72, 73
DRL	Deep Reinforcement Learning. 6, 15, 16, 29, 55, 59, 62
DWE	Distribution-Aware Weighted Estimator. 21, 24–26

F

FQI	Fitted Q-Iteration. 15, 28, 36, 37

G

GP	Gaussian Process. 15, 23, 24, 29, 35, 37
GPU	Graphic Processing Units. 15

I	
IM	Intrinsic Motivation. 65, 66, 68, 69
K	
KL	Kullback-Leibler. 67
M	
MAB	Multi-Armed Bandit. 5, 30, 31, 35, 36, 53
MC	Monte Carlo. 14
MDP	Markov Decision Process. 8–10, 12–15, 27, 32, 34, 37, 40, 43, 44, 48–50, 53–60, 73
ME	Maximum Estimator. 19–21, 24–32, 35, 36, 40
MEV	Maximum Expected Value. 19–21, 25, 31, 35, 39, 40
ML	Machine Learning. 4, 66
MSE	Mean Squared Error. 26, 27, 31
N	
NFQI	Neural Fitted Q-Iteration. 15
O	
OBE	Optimistic Bellman Operator. 66, 68–71, 74
ODQN	Optimistic Deep Q-Network. 70–74
OFU	Optimism in Face of Uncertainty. 53, 54, 66
OQL	Optimistic Q-Learning. 70–73
P	
PDF	Probability Density Function. 20–22, 55, 56
PSRL	Posterior Sampling for Reinforcement Learning. 55
Q	
QL	Q-Learning. 6, 14, 15, 20, 21, 27, 28, 32–34, 73
R	
RL	Reinforcement Learning. 4–7, 12, 14, 15, 19, 20, 30, 36, 37, 39, 40, 53–55, 61, 65, 66, 68

RLSVI	Randomized Least Squares Value Iteration. 55

T

TD	Temporal-Difference. 14, 15, 41
TS	Thompson Sampling. 53–55, 59–62

W

WDQN	Weighted Deep Q-Network. 30, 37, 38
WE	Weighted Estimator. 19–33, 35, 36
WFQI	Weighted Fitted Q-Iteration. 15, 29, 36, 37
WQL	Weighted Q-Learning. 27, 32–34

Part I

Starting Point

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

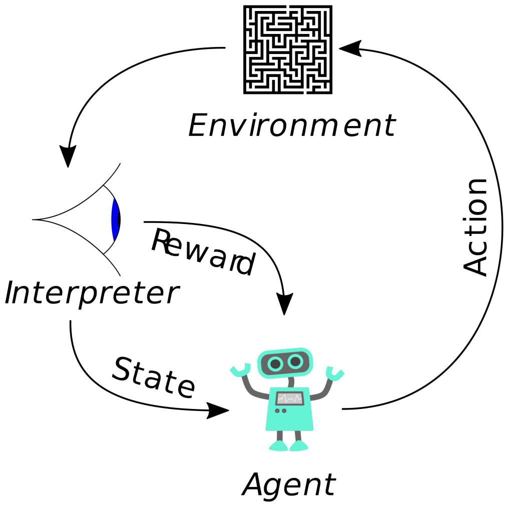


Figure 1.1: *The scheme of a RL model.*

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) [74] 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 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 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 [44]. 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 [1,17,82], and several algorithms to address this problem have been proposed such as UCB1 [4] and Thompson sampling [78].

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) [49,81,84] 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 [67,69] and chess [68].

The extraordinariness of these results is comparable to the amount of experience required by these algorithms to work. For instance, in [49] 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 divided in four parts, with this introduction included in the first of them:

- **first** part includes this chapter and Chapter 2 which introduces the main concepts of RL starting from the fundamental theory behind it and then giving a description of several methodologies related to this thesis with the purpose to provide a general, but useful, overview of what is necessary to understand the following chapters;

- **second** part includes the description of 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. In particular, Chapter 3 describes a novel way to address the problem of overestimation of the Maximum Expected Value in Q-Learning and Chapter 4 describes a novel way to deal with uncertainty in the estimate of the components of the Bellman Operator;

- **third** part includes two works about the exploitation of uncertainty to drive exploration. In this part, Chapter 5 describes a set of algorithms for the estimate of uncertainty of action-value functions and discusses the application of an exploration policy inspired by Thompson Sampling. Chapter 6 introduces a novel variant of the Bellman Operator which 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;

- **fourth** and last part 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. This hasty explanation offers a very high level definition of it, then a more formal way to model the problem is required to properly analyze it. To this end, 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

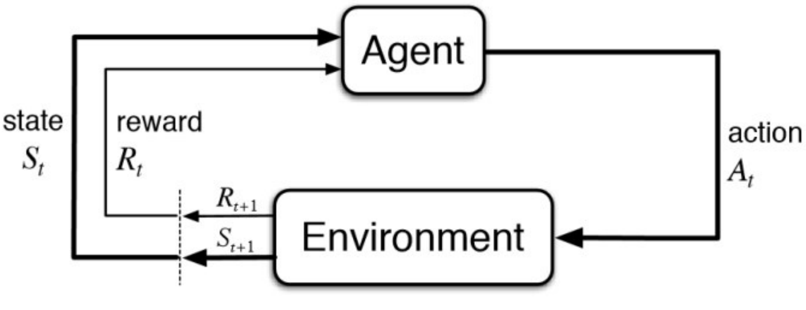


Figure 2.1: *The scheme of a RL model.*

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 aspects of the sensory data acquired from the environment by 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: $t_i, \forall i \in [0, 1, 2, \dots, H)$;
- infinite time horizon: $t_i, \forall i \in [0, 1, 2, \dots, \infty)$.

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. In all the following, we consider a *transition* as a 4-tuple $\langle s, a, r, s' \rangle$ where s is the state where action a is executed arriving at state s' and obtaining a reward r.

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 $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma, \mu \rangle$:

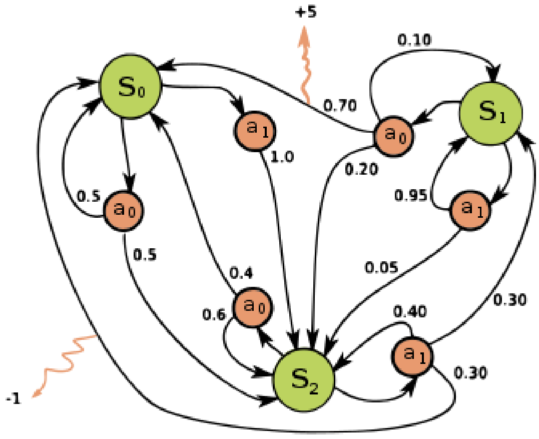


Figure 2.2: *An example of a MDP graph representing states S_i, actions a_i, transition probabilities (numbers over each edge) and rewards (pink arrows).*

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

- **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*:

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

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:

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

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) [12, 13]. The main concept in the research on DP is the optimal Bellman equation, defined as

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}[r_t + \gamma V^*(s')] \\ &= \max_a \sum_{s'} p(s'|s, a)[r + \gamma V^*(s')] \end{aligned} \tag{2.8}$$

for state value function, and

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}[r_t + \gamma \max_{a'} Q^*(s', a')] \\ &= \sum_{s'} p(s'|s, a)[r + \gamma \max_{a'} Q^*(s', a')] \end{aligned} \tag{2.9}$$

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 $\pi$ to evaluate, a small threshold $\theta$ 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:

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

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

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

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:

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

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

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

- **model-based:** the samples are used to learn a model of the environment which is subsequently used to learn the policy;
- **model-free:** the samples are used to directly compute the policy, for instance by means of value-function approximation.

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 policy.

This taxonomy splits the RL literature in two sharply different fields where value-based is mostly used for discrete MDPs whilst policy-based is mostly used in continuous MDPs, in particular in robotics.

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 balancing between exploratory actions and exploitative actions, a fundamental issue of RL known as *exploration-exploitation dilemma*. The work done in this thesis is mainly focused on model-free value-based RL in MDP with a finite number of actions, thus all the following analysis is only inherent to it.

Exploration policies In a hypothetical setting in which the agent is provided the knowledge of the optimal policy, 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 knowledge it needs to learn an optimal policy. The most trivial exploratory policy is called ε-greedy and is computed as

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

Equation 2.13 results in choosing, at each step, between a greedy or a random action according to the value of ε: 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 literature allowing to reach good results with minimum computational demand, but at the cost of a bad sample-efficiency.

One of the drawbacks of ε-greedy is that it makes no use of the information available to the agent. The *Boltzmann* policy, also known as *Softmax* policy, makes use of 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 τ 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 knowledge acquired by the agent in order to drive exploration in a smarter way than ε-greedy. A more recent strategy based on Boltzmann policy exploits the mellowmax operator *mm* to compute the Maximum Entropy Mellowmax policy [2]

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

where β is a value for which

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

and

$$mm_\omega Q(s) = \frac{\log(\frac{\sum_{i=1}^{|\mathcal{A}|} e^{\omega Q(s,a_i)}}{|\mathcal{A}|})}{\omega}. \tag{2.17}$$

The *mm* operator 2.17 is a softmax operator with the desirable properties of the Boltzmann operator 2.14, e.g. differentiability, but differently from the Boltzmann it is also a contractive operator, thus being an interesting operator for computing the target of the Bellman equation.

Temporal Difference Learning

Given a policy, RL is able to evaluate it from raw experience differently from DP which needs to know the complete model of the MDP. A well-known class of methods to do this is called Monte Carlo (MC) [61] and consists in collecting samples for an entire episode and then updating the considered value functions using the discounted cumulative reward obtained during the run:

$$V(s) \leftarrow V(s) + \alpha[G - V(s)] \tag{2.18}$$

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

To keep the desirable property of MC of being able to estimate value-functions from raw experience without losing the good property of DP of being 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 one of the most used methods in RL. The simplest TD method is known as TD(0) and updates the value-function after each step with

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

The success of TD over DP and MC is motivated by this simple update formula since the algorithm is able to take the advantages of both techniques with guarantees of convergence to the optimal value-function.

TD methods can be used for control if, instead of computing the state-value function, the action-value function is computed. This class of methods is split in two subclasses according to which policy is followed to compute the target of the TD update:

- **on-policy:** following the policy which 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 π its purpose is to estimate the action-value function $Q_\pi(s, a)$ for each (s, a) tuple via the update formula:

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

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

Q-Learning The corresponding off-policy TD algorithm for control w.r.t. SARSA is Q-Learning (QL) [85]. Contrarily to SARSA, Q-Learning (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

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

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, SARSA will learn a policy which let the agent stay far to undesired state and reach the goal safely but slowly; while 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. robotics, where the cost of reaching catastrophic situations is very high and is better to be conservative in order 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 [27]. 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

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

Under some conditions, the algorithm is proved to converge to the optimal Q in a finite number of steps. FQI has been firstly presented using extra-trees regression [32], but has been proved to work well with other kind of regression, e.g. neural network in the Neural Fitted Q-Iteration (NFQI) algorithm [60] or Gaussian Process (GP) regression [59] in the Weighted Fitted Q-Iteration (WFQI) algorithm [23].

Approximated Temporal Difference Learning

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 to 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 the learning with shallow RL techniques making them computationally impractical. The recent coming of powerful computational resources, in particular the use of Graphic 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 which showed the potential of DRL and started the research about it is the Deep Q-Network (DQN) algorithm [49]. It consists in the use of a deep neural network and 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

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

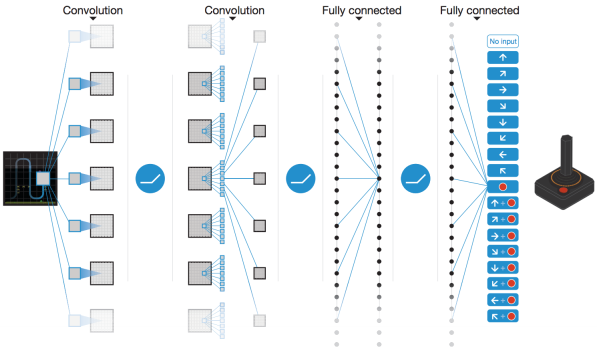


Figure 2.3: *Synthetic scheme of a deep neural network which takes the input of the screen of a videogame and output the action to perform in that state.*

and learning it in a supervised way.

The algorithm shows outstanding performance on the previously almost intractable problems of Atari games [10], a set of more than 50 old but challenging videogames, 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 consisting of using raw pixel frame as input of a convolutional network and applying a simple gradient descent step to update the network. In this way the complexity of the input, that made the solving of these problems intractable in the past, is addressed with the power of the deep convolutional network to extract highly abstract representation of it sensibly reducing its dimensionality and making it suitable for approximating the action-value function effectively. Despite its good properties, DQN has some issues about the stability of learning and the sample-efficiency that are common problems among DRL techniques. Indeed, the number of samples required by DQN to learn are in the order of millions and correspond to more than month of human playing; moreover in some games the algorithm shows very unstable performance even bringing to completely forgetting the good policy it learned after many steps. For these reasons, without forgetting the promising line of research that DQN revealed, the following literature focused on many variants of DQN to improve some of the issues it has. For instance, Double Deep Q-Network (DDQN) [36] has been proposed to improve the estimate of the action-value function whereas Bootstrapped Deep Q-Network (BDQN) [52] to improve exploration capabilities; both these algorithms will be considered and explained further in the thesis respectively in Chapter 3 and 5.

Part II

Bellman Update

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 [79].

The most used approach to this estimation problem is the Maximum Estimator (ME) which simply takes the maximum estimated utility. As proved in [71], this estimate is positively biased and, if used in iterative algorithms, can increase the approximation error step-by step [79]. More effective estimators have been proposed in the recent years. The Double Estimator (DE) [80] 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 [80] 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) [24] 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 a finite set of random variables, thus, in a subsequent work, we extended the study to problems with a set of infinite random variables and proposed an extension of WE [23] 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 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 [71], this estimator has a positive bias that may cause issues in applications of ME, such as in the RL algorithm of QL 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

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

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 [79] and theoretically analyzed in [80]. 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 (DQL) [79] 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 [87] 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 (DWE):

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

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 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, 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:

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

where 3.8-3.9 follow from the independence assumption. The term $\prod_{\mathcal{Z}} F^S_{\hat{\mu}_y}(x)^{dy} = P\left(\bigwedge_{y \in \mathcal{Z}} \hat{\mu}_y(S) \leq x\right)$ 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: $\prod_{\mathcal{Z}} F^S_{\hat{\mu}_y}(x)^{dy} = \exp\left(\int_{\mathcal{Z}} \ln F^S_{\hat{\mu}_y}(x) dy\right)$ [35].

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 $f^S_{\hat{\mu}_z}$ (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 $f^S_{\hat{\mu}_z}$ with a normal distribution $\tilde{f}^S_{\hat{\mu}_z} = \mathcal{N}\left(\hat{\mu}_z, \hat{\sigma}^2_{\hat{\mu}_z}\right)$.

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

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

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 $\epsilon \sim \mathcal{N}(0, \sigma^2_n)$.

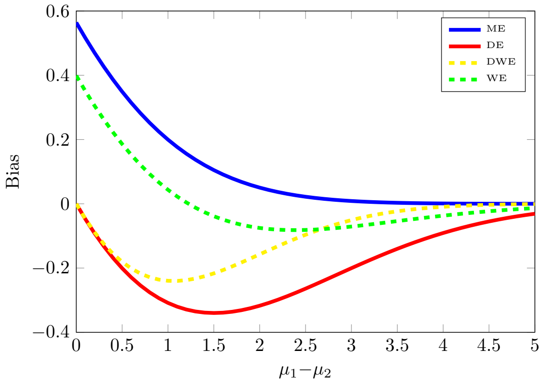


Figure 3.1: *Comparison of the bias of the different estimators varying the difference of the means*

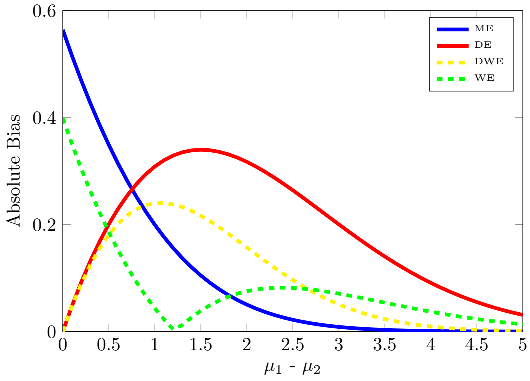


Figure 3.2: *Comparison of the absolute bias of the different estimators varying the difference of the means.*

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) [59]. 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 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 [80]. For what concerns the direction of the bias, ME is positively biased, while DE is negatively biased. If we look at the absolute bias, there is no clear winner. For instance, when all the random variables are identically distributed, DE is unbiased, while the same setting represents a worst case for ME. On the other hand, when the maximum expected value is sufficiently larger than the expected values of the other variables, the absolute bias of ME can be significantly smaller than the one of DE. The bias of ME is bounded by:

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

For the bias of DE, [80] 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:*

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

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:*

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

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 of their expected values. Both variables have 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 a clear winner. As previously mentioned, when the variables have the same mean, both DE and DWE are unbiased, while it represents a worst case for the bias of ME and WE. It follows that, when the difference of 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 minimum 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 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 variable whose expected value is significantly larger (w.r.t. the variances) than all the expected values of all the other variables. In all the other cases, WE is less biased.

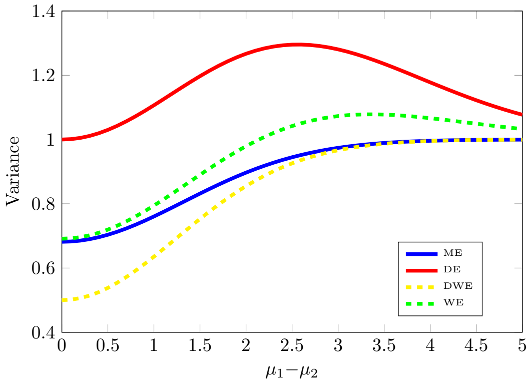


Figure 3.3: *Comparison of the variance of the different estimators varying the difference of the means.*

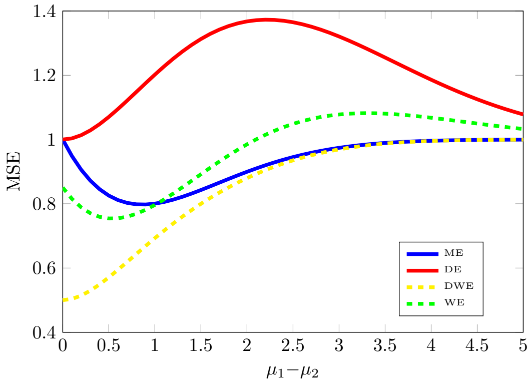


Figure 3.4: *Comparison of the MSE of the different estimators varying the difference of the means.*

3.3.2 Variance

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

[80] 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: $\mathrm{Var}\left(\hat{\mu}_*^{ME}\right) \leq \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}$, $\mathrm{Var}\left(\hat{\mu}_*^{DE}\right) \leq \sum_{i=1}^{M} \frac{\sigma_i^2}{|S_i|}$. 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*

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

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*

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

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

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., $\varepsilon$-greedy)
    $s', r \leftarrow$ 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
```

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.

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 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.

3.4.1 Online

It is demonstrated that since QL 2.3.2 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^* [85]. 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 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 could fail. In the last years, different approaches have been proposed trying to overcome this issue [11, 45, 89]. In particular, the most successful one is the DQL algorithm [79] 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 [24]. WQL 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

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 $\mathcal{A} = \{a_1, \dots, a_M\}$
Train $\widehat{Q}_0^{\bar{a}}$ on $\mathcal{T}_0 = \{\langle s_i, r_i \rangle$ 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 $\mathcal{T}_t = \{\langle s_i, r_i \rangle$ 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
```

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'.

3.4.2 Batch

The FQI update 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 which replaces ME with DE is called Double Fitted Q-Iteration (DFQI), and the variant which we propose using WE is called *Weighted* FQI (WFQI) [23].

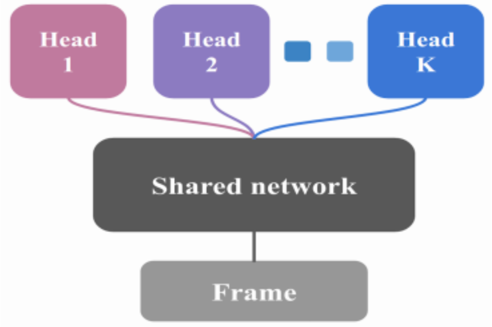


Figure 3.5: *Architecture of the neural network used in BDQN.*

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.

3.4.3 Deep Reinforcement Learning

We also analyzed the case of DRL where the overestimation problem caused by ME happens also in the DQN algorithm resulting in critical loss of performance in some problems due to instability. The DDQN algorithm [36] replaces ME with DE and shows considerable improvements and performance and stability. We worked on the replacement of ME with WE even though it 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.

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 BDQN [52] showed in Figure 3.5. This work follows the DQN algorithm described in [49] 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 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 only use the K-th head trained with WE for exploration. This way, the training of the $K-1$ heads do 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:

$$y_i^k = \begin{cases} r_i & \text{if episode terminates at episode } i+1 \\ r_i + \gamma \max_{a'} \hat{Q}^k(s'_i, a') & \text{otherwise} \end{cases} \tag{3.12}$$

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

$$y_i^K = \begin{cases} r_i & \text{if episode terminates at episode } i+1 \\ r_i + \gamma \sum_{j \in 1, \ldots, \#\mathcal{A}} w_i^j Q^K(s'_i, a_j; \theta_K^-) & \text{otherwise} \end{cases} \tag{3.13}$$

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 θ_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).

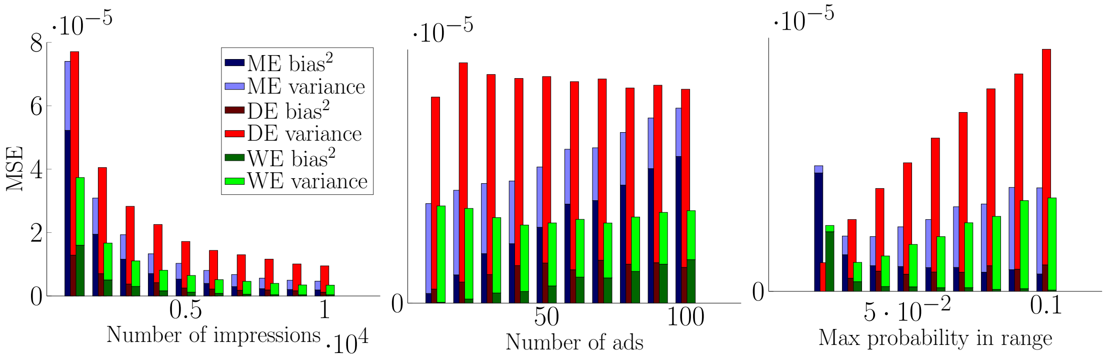
3.5 Empirical results

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

3.5.1 Discrete States and Action Spaces

Internet Ads

We consider this MAB problem as formulated in [80]. 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



(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.*

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 3.6, 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)) reasonably the 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 3.2). Figure 3.6(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 3.6(c) when all the ads share the same click rate (0.02), where DE is the best. However, it starts to have 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 [87]. 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

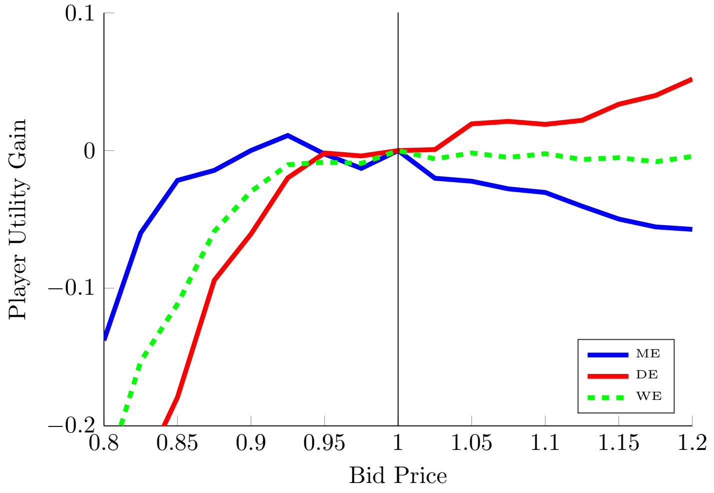


Figure 3.7: *Relative player 1 utility gain for different value of the bid defined as* $\frac{utility(b)}{utility(v)} - 1$*. Results are averaged over 2000 experiments.*

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 [4] in a learning phase consisting of 10000 rounds of exploration (i.e. impressions), as done in [87].

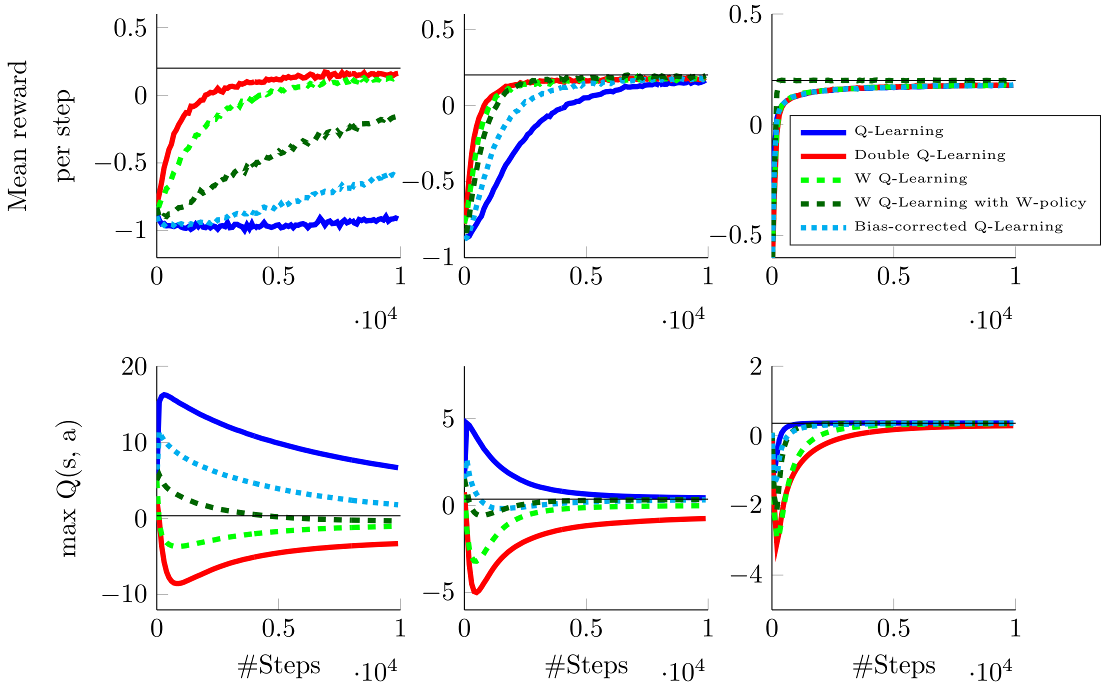
Figure 3.7 shows the utility gain of advertiser 1 when using ME, DE and WE.[1] The true bid price is highlighted with a black vertical bar. ME results to be only one not able to achieve incentive compatibility, since the utility has positive values before the true bid price. On the contrary with DE and WE the advertiser has no incentive to underbid, but there is an incentive to overbid using DE. Therefore WE is the only estimator 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 [79]. In this domain, we compare the three estimators together with the performance of an algorithm called Bias-corrected QL, a modified version of QL 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 [**?**, 45]. Moreover, we test WQL 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 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

---

[1] The debiasing algorithm proposed in [87] 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.



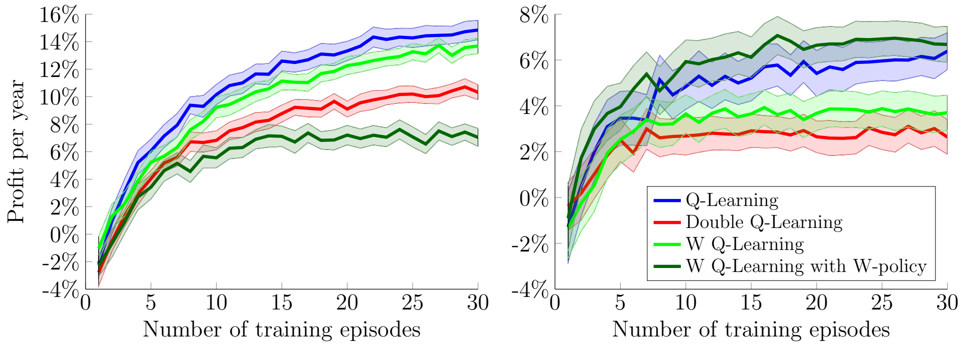
(a) Bernoulli.

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

(c) $\mathcal{N}(-1, 1)$.

Figure 3.8: *Grid world results with the three reward functions averaged over 10000 experiments. Optimal policy is the black line.*

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 3.8, 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 algorithm 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 the exploration faster than ε greedy due to exploiting of the good approximation of the Q-function computed by WQL. It is worth to point out that WQL 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,



(a) Training phase.

(b) Test phase.

Figure 3.9: *Profit per year averaged over 100 experiments.*

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 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 $s_{i=1\ldots7}(t) = \{-1, 0, 1\}$ 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 QL, 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 3.9 shows the profit per year, 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 ε-greedy policy is used for QL and DQL, while WQL uses both the ε-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 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. DQL performs worse than QL and WQL both in training phase and test phase. The reason is that in many states there is an action that is significantly better

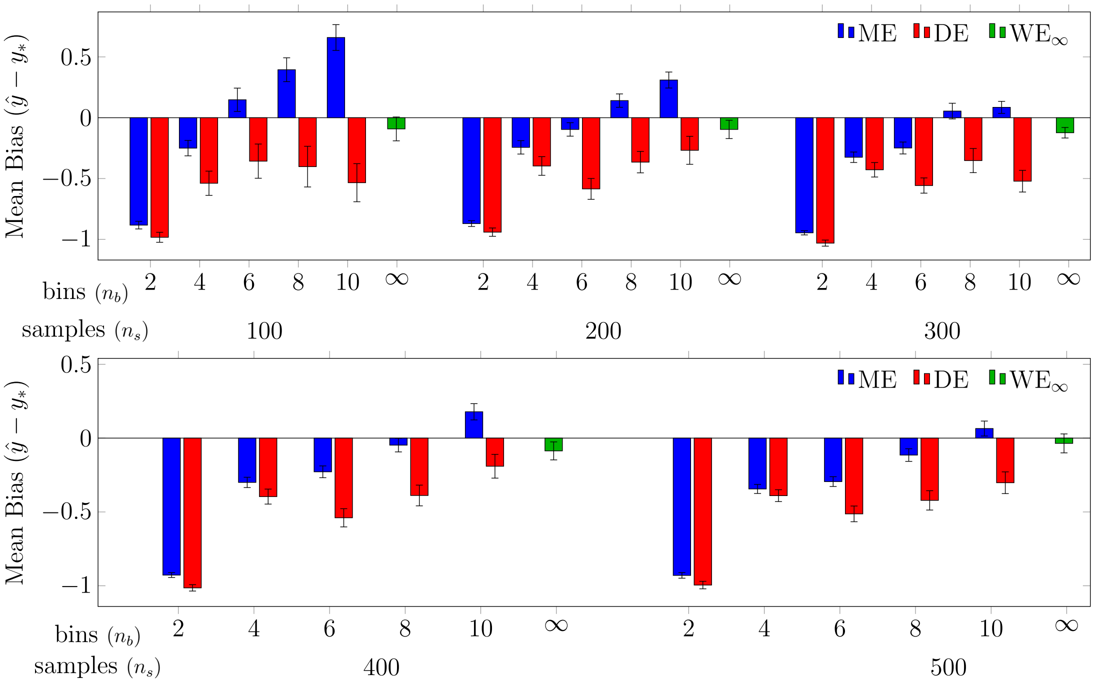


Figure 3.10: *Mean bias obtained by ME, DE and WE$_\infty$ with different sample sizes and bins (only for ME and DE).*

than the others, that represents the case where ME gives the best results, and the case where DE suffers the most.

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 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 3.10, the number n_b of

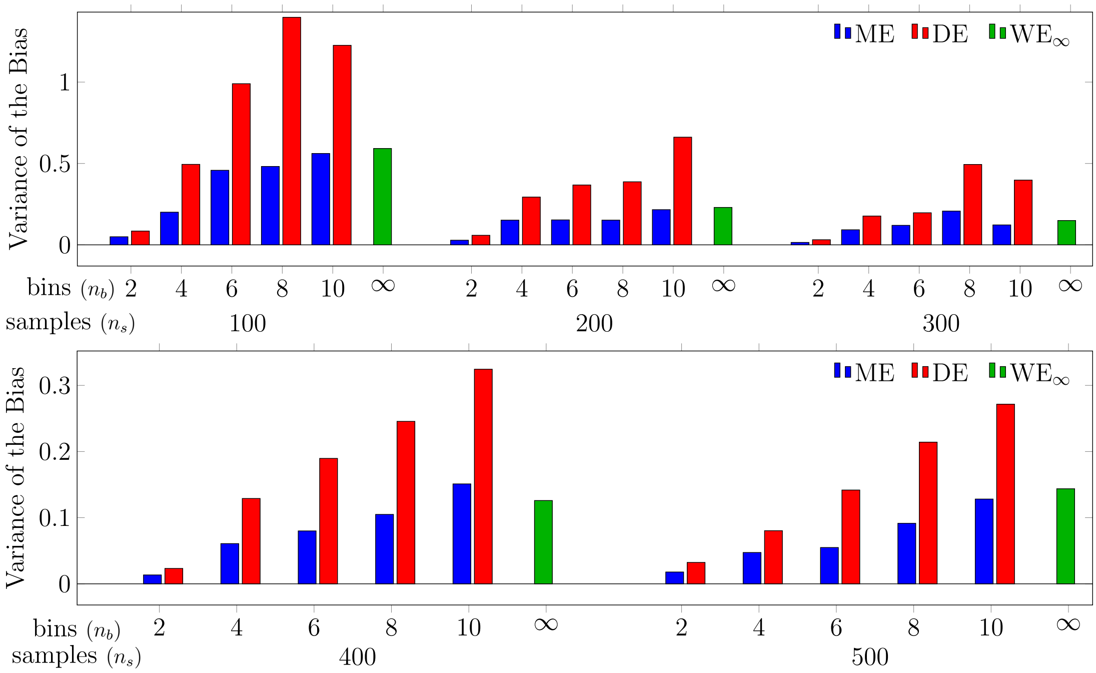
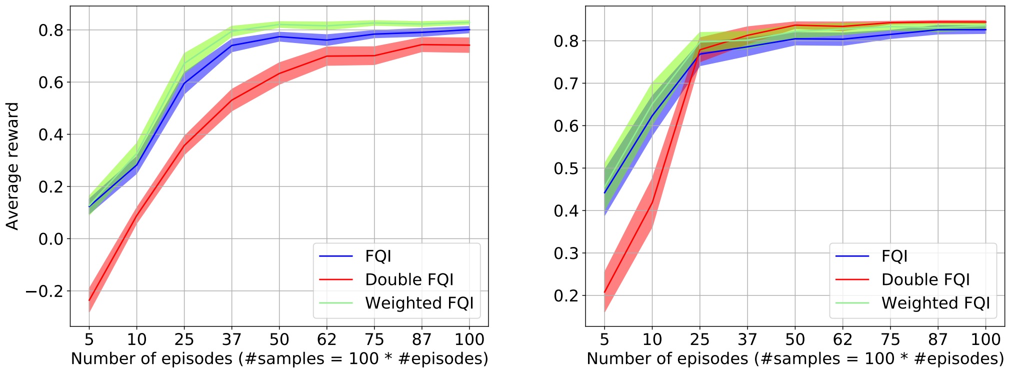


Figure 3.11: *Variance of the bias obtained by ME, DE and WE_∞ with different sample sizes and bins.*

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 3.11 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 because this pricing 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 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 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 [25]. 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 ($u \in [-5, 5]$)



(a) Discrete actions.

(b) Continuous actions.

Figure 3.12: *Swing-up pendulum performance averaged over* 100 *experiments.*

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. Figure 3.12 show that WFQI for discrete actions 4 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 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.

3.5.3 Deep Reinforcement Learning Scenario

Acrobot

We evaluate the performance of DQN, DDQN 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

---

[2] We use the Acrobot-v1 environment of the OpenAI Gym library [16].

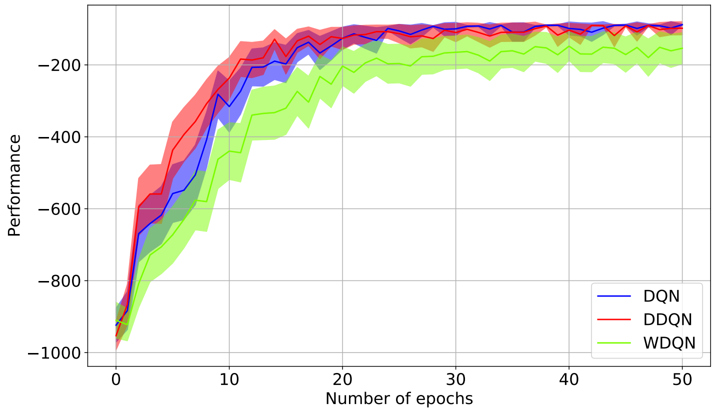


Figure 3.13: *Acrobot performance averaged on* 100 *experiments.*

Table 3.1: *Hyperparameters setting in the Acrobot experiment.*

Replay memory size	5000
Initial replay size	100
Agent history length	1
Target network update frequency	100
Masking probability p	1
Number of hidden layers	2
Number of neurons	80
Number of heads	10
Test samples	5000
Evaluation frequency	1000
Initial exploration rate	1
Final exploration rate	0.05
Final exploration frame	5000
Max no-op actions	0
Total number of steps	50000
Optimizer	Adam

the bot reaches the threshold height. The discount factor is $\gamma = 0.99$. The horizon is set to 1000. The hyperparameters of the training are specified in Table 3.1.

The results shown in Figure 3.13 are not satisfying in this problem and we still need to investigate the reasons of this. Moreover, we plan to evaluate the performance of WDQN in more complex problems such as Atari games.

CHAPTER 4

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 to the optimal action-value with infinite samples, in practice the presence of stochastic components leads to slow learning. Actually, the majority of real-world problems have significant sources of stochasticity: the environment could 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 high variance of the returns, the most successful RL algorithms are based on bootstrapping (e.g. Q-Learning [85]), that trades off the variance of the estimation with a consistent, but biased estimator. However, with a finite number of samples, the bias of the estimation could be significantly relevant when propagating the action-values to the next state and, recursively, to all the other states. Recent works tried to deal with this issue, in particular 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 bad performance; indeed most of the policies are not dependent on the accuracy of the action-values, but instead they 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 weighting the new samples according to the uncertainty of the 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 uncertainty and the propagation of information across the states of the MDP has been addressed in many works on RL [50, 77]. Some of them consist in tuning the meta-parameters of the learning process according to uncertainty, e.g. the dynamical changes of the environment [42, 65, 88].

The interesting aspect of this approach is that it is possible to use any MEV estimator and that it 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

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

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

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

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

$$\begin{aligned} \tilde{R}(s,a) &= \mathop{\mathbb{E}}_{x' \sim \mathcal{P}(s'|s,a)}\left[r(s,a,s')\right], \\ \tilde{Q}(s,a) &= \mathop{\mathbb{E}}_{x' \sim \mathcal{P}(s'|s,a)}\left[\max_{a'} Q^*(s',a')\right]. \end{aligned} \tag{4.3}$$

We can give an interpretation of these two functions: $\tilde{R}(s,a)$ is the expected immediate reward of the action a in the 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 transition and reward models, while $\tilde{Q}$ also depends 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 relations with the standard Q-Learning update. We call this method RQ-Learning as it decomposes the TD-Error in 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 in 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 the estimation of 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 the action a in the state s, which depends, recursively, on the other action-value functions. It is clear that the propagation of uncertain values only affects 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 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:

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

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 δ_t increases monotonically in the interval $[0,1]$, our method works by increasing the effective horizon 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 is close to 1 [7, 20, 30].

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 weight 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 have to make the assumption that the learning rate is independent from data. This assumption is needed in order to have a closed form for the variance of the estimator, and it works well in practice. We will assume also that the samples X_i are i.i.d., with mean μ and variance σ^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:

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

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

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 σ^2. It is possible to compute in an incremental way both the sample covariance, in the traditional way, and ω:

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

A weak assumption of this model is that the variables are identically distributed. While this could be true for the reward function, if the MDP is stationary, this is not true for the Q-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 temporal window in which the distribution is approximately stationary. Using this approach, we can compute the variance of the process in a given time window forgetting old values that can lead to a biased estimation of the current window variance. While 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 $\mathrm{Var}\left[\widetilde{X}_t\right]$. 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 η 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 speed of the decrease of 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 of the discount factor faster than the decrease of 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 η 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 [28, 85]:

$$\begin{aligned} &0 \leq \alpha < 1, \\ &\lim_{N \to \infty} \sum_{t=1}^{N} \alpha_t = \infty, \\ &\lim_{N \to \infty} \sum_{t=1}^{N} \alpha_t^2 < \infty. \end{aligned} \tag{4.16}$$

In order to give some preliminary results about the convergence of RQ-Learning and to motivate the choice of the formulas of learning rates α and δ, 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 σ 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, that 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:

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

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

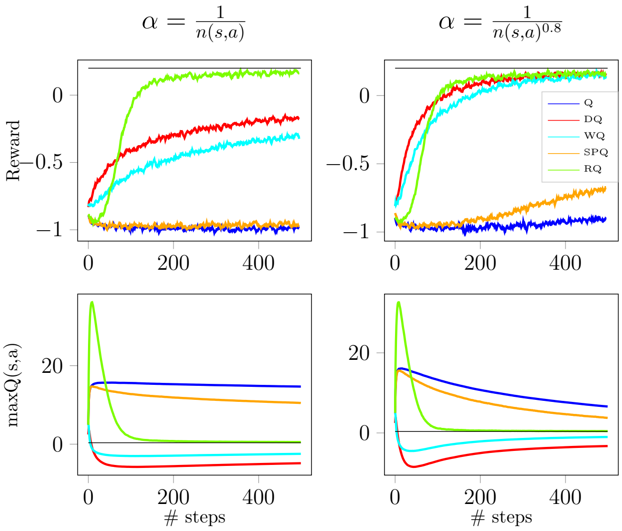
This analysis considers only the estimation of the variance of the sample mean. The analysis of the generic variance estimator is more complex, but we conjecture that convergence could be guaranteed under some mild conditions on η and σ.

4.3 Experimental Results

We compare RQ-Learning against Q-Learning [85], Double Q-Learning [79], Weighted Q-Learning [24] and Speedy Q-Learning[1] [33] 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 [79] 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 consists in exploiting data in the best way, in particular in the estimation of the action-value, giving higher relevancy 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.



(a) $\alpha = \dfrac{1}{n(s,a)}$ (b) $\alpha = \dfrac{1}{n(s,a)^{0.8}}$

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 weights 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]

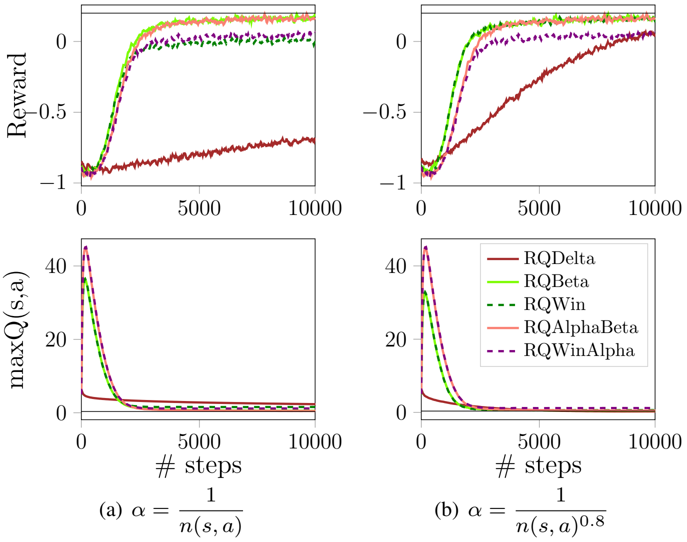
4.3.1 Noisy Grid World

This environment, proposed in [79] and [24], consists of a 3×3 grid with the initial position in the lower-left cell and the goal state in the upper-right cell. Each action performed in a non-goal state obtains a reward -12 and 10 with equal probability. In the goal state, every action obtains a reward of 5 and terminates the episode. The discount factor is $\gamma = 0.95$. The policy is ε-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 mean reward per step and the approximation of the maximum action-value in the initial state computed by the other algorithms and RQ-Learning with α as used by the other algorithms, but with the separated variance-dependent learning rate β 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 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.



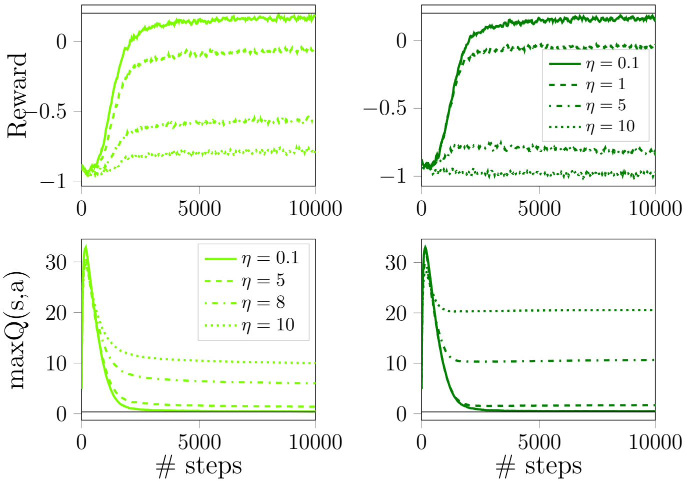
(a) $\alpha = \dfrac{1}{n(s,a)}$ (b) $\alpha = \dfrac{1}{n(s,a)^{0.8}}$

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 mean reward per step, even with a diverging estimate of the action-value function. This is an empirical evidence of our conjecture: the bias of the estimation is not necessarily correlated to the performance. Moreover, as expected, the performance of RQ-Learning is not affected by the exponent used in the learning rate. The other algorithms achieve the optimal performance only in the setting with the higher learning rate, confirming the advantage of giving more importance to 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 variance with a window of length 50 and $\eta = 0.5$. “RQAlphaBeta” uses a variance-dependent learning rate also for α with $\eta = 100$ and β with $\eta = 1$; “RQWinAlpha” is the same configuration of the previous one, but uses a windowed β with $\eta = 0.5$. Note that η has a larger value in configurations without windowed variance estimation because such configurations are likely to overestimate the current variance of the process. “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 with 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 of Figure 4.1 and 4.2 for RQ-Learning with different values of η to give a sense of how this parameter influences the learning process. It is clear how lower values of η (i.e. a higher learning rate) let the results converge faster, confirming that a high value of the learning rate helps to speedup 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* η *and* $k = 0.8$. *Results are averaged over* 10000 *experiments.*

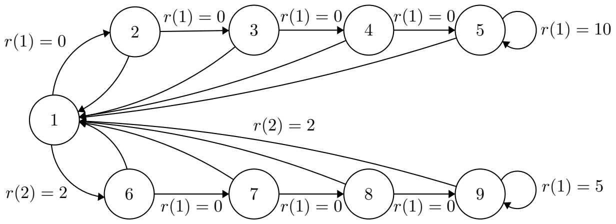
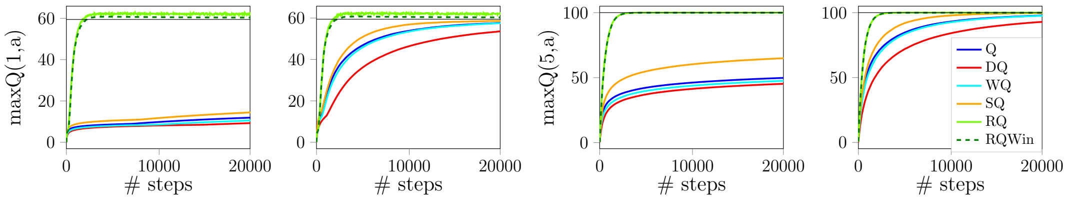


Figure 4.4: *Structure of the double-chain problem.*

4.3.2 Double Chain

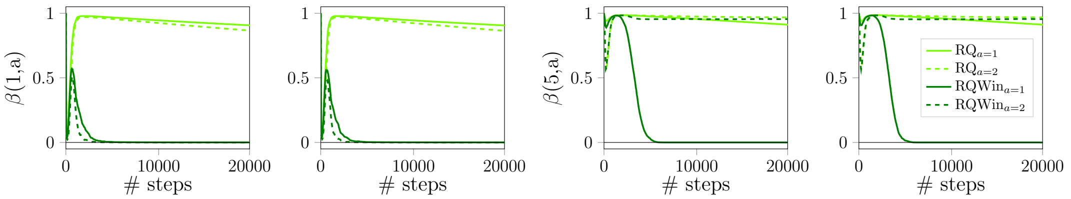
This is a problem proposed in [57], which consists in a Markov chain with two branches (Figure 4.4). In state 1, action 1 yields a reward of 0 and moves the agent in state 2; action 2 yields a reward of 2 and moves the agent in state 6. In all other states, action 2 moves the agent in state 1 and returns a reward of 2; action 1 moves the agent in the next state of the chain returning a reward of 0. In states 5 and 9, action 1 yields a reward of respectively 10 and 5. 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,



(a) $\alpha = \dfrac{1}{n(s,a)}$ (b) $\alpha = \dfrac{1}{n(s,a)^{0.51}}$ (c) $\alpha = \dfrac{1}{n(s,a)}$ (d) $\alpha = \dfrac{1}{n(s,a)^{0.51}}$

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.*



(a) $\alpha = \dfrac{1}{n(s,a)}$ (b) $\alpha = \dfrac{1}{n(s,a)^{0.51}}$ (c) $\alpha = \dfrac{1}{n(s,a)}$ (d) $\alpha = \dfrac{1}{n(s,a)^{0.51}}$

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.*

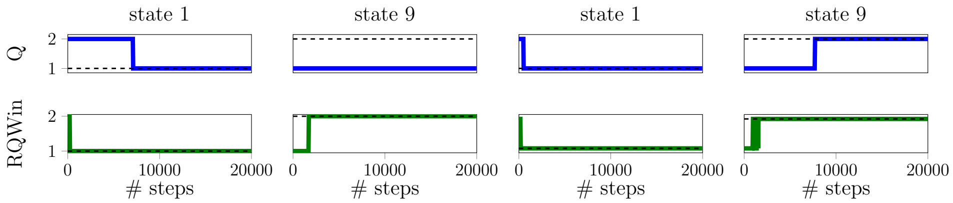
together with the stochasticity of the transitions, causes the oscillation of the estimate and slow convergence rate. This behavior is highlighted in Figure 4.6 where we show the learning rates of the action-value in the considered states. While initially the learning rates are similar, the windowed 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 cases, increasing η can speedup the decreasing of the learning rate.

In this experiment, RQ-Learning does not only approximate the value function very well, but it 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 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 with the others. This empirical result confirms our conjecture.

4.3.3 Grid World with Holes

This environment consists in a 5×5 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



(a) $\alpha = \dfrac{1}{n(s,a)}$

(b) $\alpha = \dfrac{1}{n(s,a)^{0.51}}$

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.*

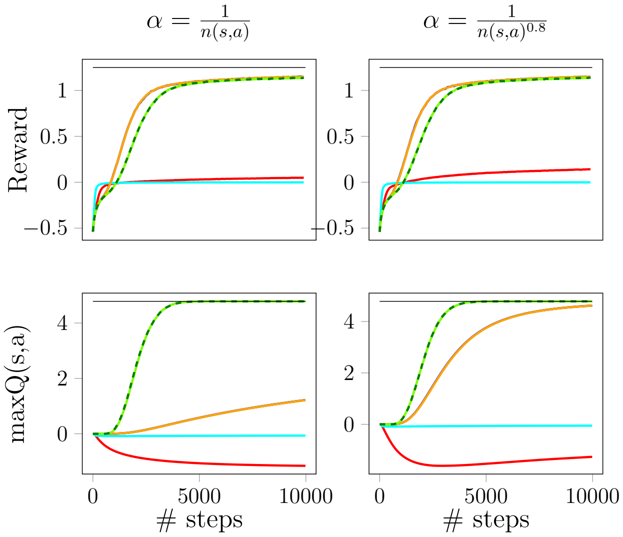


Figure 4.8: *Mean reward per step (top) and maximum action-value estimate in the Grid World with Holes problem in the initial state (bottom) of all the other algorithms and of the best setting of RQ-Learning for this experiment. Results are averaged over* 10000 *experiments.*

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 of 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 negative reward with probability $\frac{\varepsilon}{2}$ due to the ε-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.

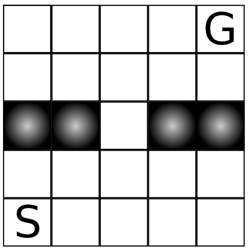


Figure 4.9: *Structure of the Grid World with Holes problem.*

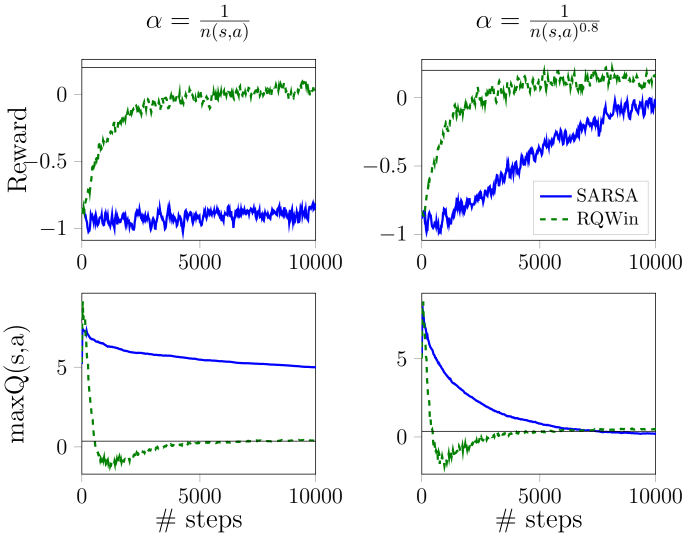


Figure 4.10: *Mean reward per step (top) and maximum action-value estimate in the Noisy Grid World problem in the initial state (bottom) of SARSA and of the on-policy windowed version of RQ-Learning for this experiment. Results are averaged over* 1000 *experiments.*

4.3.4 On-policy learning

As we have 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 current policy action-value function, not the optimal one. Indeed, by looking at the mean 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

CHAPTER 5

Thompson Sampling Based Algorithms for Exploration in Reinforcement Learning

A desirable property of sampling strategies is to have theoretical proves of their efficiency in terms of the number of collected samples. Among others, the principle of Optimism in Face of Uncertainty (OFU) [44] has been well studied in literature where large evidence of its efficiency is given [39–41]. OFU states that actions with statistically uncertain values are to be favored (e.g., through an exploration bonus) in the action-selection process compared to the more certain ones in order to improve the knowledge of the environment. This optimistic sampling strategy speedups the learning of the action values and efficiently exploits them 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) [78]. 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 [18, 34, 46, 66]. 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 in order to make optimism-based strategies feasible.

One widely used approach to model uncertainty is discussed in the literature about *intrinsically motivated* RL [19, 62] 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 [9, 76]. Other techniques to represent the interest of a state have been proposed, for instance, relying on the amount of knowledge a deep learning model has of a specific state [56] or considering the representativeness of a state w.r.t. the MDP [14]. 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 where a probability distribution is used to model the uncertainty over action value functions in each state. One of the first works following this approach is Bayesian Q-Learning [22] which proposes to estimate the action values 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 allowing to pursue OFU using TS.

We present several methodologies to efficiently compute the uncertainty of action values in online RL problems and to use TS strategy. We propose two main techniques to accomplish these tasks, one of them 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 where the exploration is a critical aspect to empirically show how TS can outperform other sampling strategies in terms of performance and sample-complexity independently from the dimensionality of the problem.

5.1 Related work

The convergence of many RL algorithms is guaranteed when the exploration policy allows visiting each state for an infinite number of times. Common sampling strategies like ε-greedy or Boltzmann guarantee the convergence property, but they can suffer from unfeasible learning time. For this reason, finite-time bounds for convergence are a desirable property of exploration policies to understand the theoretical guarantees about the feasibility of a learning algorithm based on them. The problem of finding finite-time bounds is addressed in several works that propose nearly-optimal sampling strategies with polynomial bounds for 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 [5] and UCRL2 [39] guarantee logarithmic regret bounds 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 [4] algorithm used in MABs. A similar approach is proposed in Delayed Q-Learning [72] where the 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 [41] algorithm and the later R-MAX [15] algorithm. However, despite the desirable theoretical properties, these algorithms usually do not work well in practice, and most importantly they do not scale well w.r.t. the dimensionality of the problem.

A different approach to pursue 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. The model-based ones propose to start from a prior distribution over

MDPs and compute the respective posterior distribution as newer samples are collected from the MDP [21,43]. Following this approach, the Posterior Sampling for Reinforcement Learning (PSRL) [73] 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 convergence to the optimal policy for a stationary process with discrete states. An extension of this work which shows a better upper bound on expected regret for PSRL is presented in [53]. 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 using randomized value functions sampled by probability distributions. The first algorithm based on a Bayesian approach is Bayesian Q-Learning [22]. 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 [24] algorithm 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) [54] and its extension to the DRL setting BDQN [52], which we consider later in this work.

5.2 Thompson Sampling in value-based Reinforcement Learning

We want to use glsts 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 respectively the mean and the variance of these distributions. The PDF and CDF of the approximated 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 of each action a_i to be the optimal one in a given state s can be computed as:

$$P\left(\tilde{Q}_i(s) = \max_j \tilde{Q}_j(s)\right) = P\left(\tilde{Q}_i(s) \geq \tilde{Q}_j(s), \forall j \neq i\right) = \int_{-\infty}^{+\infty} \tilde{f}_i^s(x) \prod_{j \neq i} \tilde{F}_j^s(x) dx. \tag{5.1}$$

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 sampling an action from the discrete density function resulting from them. Since we are only interested 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 sampled maximum 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 [22]).

Algorithm 6 Standard mean and variance update

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

Algorithm 7 Mean and variance update using momentums

```
Inputs: current mean $\overline{X}$ , current squared mean $\overline{X^2}$, current variance $S^2$, new sample $x$, learning rate $\alpha$
$\Delta_{\overline{X}} \leftarrow x - \overline{X}$
$\Delta_{\overline{X^2}} \leftarrow x^2 - \overline{X^2}$
$\overline{X} \leftarrow \overline{X} + \alpha \Delta_{\overline{X}}$
$\overline{X^2} \leftarrow \overline{X^2} + \alpha \Delta_{\overline{X^2}}$
$S^2 \leftarrow \overline{X^2} - \overline{X}^2$
```

The approximation of the unknown PDFs and CDFs is done relying on the CLT, as proposed in [**?**], 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 from 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 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 to deal with this problem. The former consists in updating the uncertainty in an online fashion and is specifically proposed to be used in discrete MDPs; nevertheless, we discuss its extension to continuous state spaces, that 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 6). An alternative method, which is more suitable for computational reasons but is slightly more biased than the previous one, is to estimate the variance as the difference between the second order momentum and the square of the first order one (Algorithm 7). The previously described procedures can be easily integrated into the update 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 have to normalize the variance of the process $\sigma^2_{process}$ by the number of effective samples that is limited by the learning rate.

Algorithm 8 SARSA with online variance update

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

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

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

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 8. 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.

Algorithm 9 SARSA with online variance update and Hoeffding upper bound

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

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

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

   where $N = \lceil n_{eff} \rceil$ and β 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.

Algorithm 10 SARSA with function approximation with variance update

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

2. Another upper bound is given by the Hoeffding bound [37]. 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 $\varepsilon > 0$:

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

   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:

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

   From these considerations, we can derive the variance estimation method shown in Algorithm 9 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 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, delays too much the exploitation phase. A way to overcome this issue, is to sum the variance with a term, similar to the Hoeffding bound, that uses a parameter which allows to tune the amount of exploration in the first stages of learning:

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

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

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

The previous techniques cannot be applied in MDPs with **continuous state spaces** since a tabular representation of the variance is not suitable. In these problems, the variance of the process can be approximated using an adaptation of the procedures in Algorithm 6 and Algorithm 7. While the latter is immediate since it simply requires to estimate 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 10. In this case, the problem we are left with is the estimation 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 hash-function [76].

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 in training different regressors of the target function with datasets obtained through sampling with replacement from the original dataset [31]. The approximation of the variance of the estimate can be directly computed as the variance of the target function estimates provided by each regressor. To implement our idea, we consider the BDQN algorithm [52], a variant of the famous DQN algorithm [49], 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 [53] that, 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 both using multiple neural networks, or splitting the output of only one neural network in multiple heads, each of which representing the $\tilde{Q}_k$ estimate of the real action value function Q. Bootstrapping is performed diversifying the heads with the random initializations of their weights and assigning a randomly generated binary mask $m_1, \dots, m_K \in [0, 1]$ to each sample to indicate which head should be trained with it. Moreover, the update of DDQN [81] is used in place of the standard DQN update to avoid the overestimation issue. Finally, the exploration policy consists in 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 11 shows the pseudocode of TS via bootstrapping.

Bootstrapped Q-Learning We slightly modify BDQN to adapt it for 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

---

[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 11 Bootstrapped DQN with Thompson Sampling

```
Inputs: action value function network $Q$ with $K$ outputs $Q_{k=1}^{K}$. Masking distribution $M$. Replay Buffer $B$ storing experience for training.
repeat
    Obtain state $s_t$ from the environment
    Sample action values $a_t^i$ from a randomly chosen head $Q_k$ for each action $i$
    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 $s_{t+1}$
    Sample bootstrap mask $m_t \sim M$
    Add $(s_t, a_t, r_t, s_{t+1}, m_t)$ to replay buffer $B$
until End condition is satisfied
```

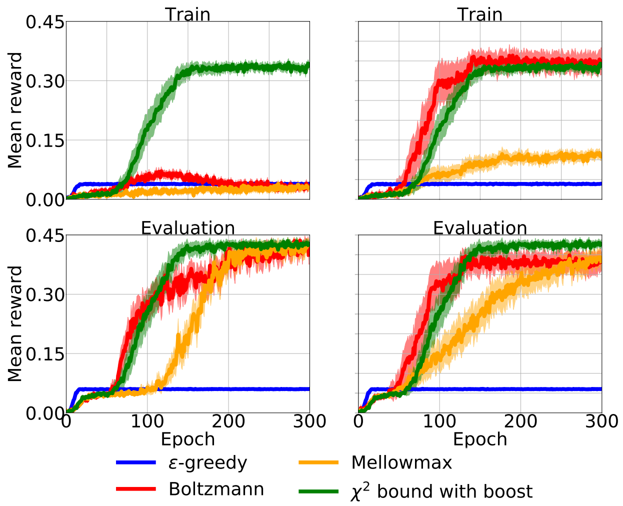
rule 3.1.1. To perform bootstrapping, the action value functions are initialized randomly and masking is used as in BDQN.

5.4 Experiments

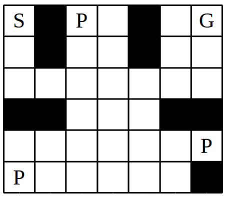
Our focus on the empirical evaluation of TS over other exploration strategies is twofold since we want to show how TS can improve and/or stabilize learning in MDPs where exploration is a key issue, but we want also to evince how it can robustly deal with generic MDPs.

5.4.1 Discrete state space

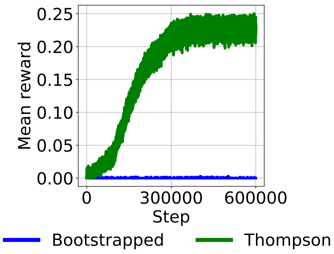
The **Taxi** problem consists of a discrete MDP, similar to a grid world, where the agent has to collect passengers before going to the goal position [3] (Figure 5.1(b)). The need for exploration is intuitively critical in this problem considering the unknown location of the passengers and the sparse 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. For the online case, we compare the performance of χ^2 upper bound with boost against the classic exploration strategies ε-greedy and Boltzmann, together with a recently proposed variant of the Boltzmann called mellowmax [3]. For the bootstrapped case, we consider tabular approximation and Double Q-Learning **??** as the learning algorithm, opposed to BDQN which instead uses a deep neural network and Deep Double Q-Learning [81]; thus, we refer to this approach as *Bootstrapped Q-Learning*. Figure 5.1(a) shows that TS reaches better results in less time w.r.t. the other strategies. In particular, it shows how ε-greedy is completely ineffective in this problem, while Boltzmann and mellowmax are able to learn good policies, but still more slowly than TS. Thus, we can conclude that here TS shows the best balancing 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 good as in the online variance case.



(a) Online variance



(b) Taxi grid



(c) Bootstrapped variance

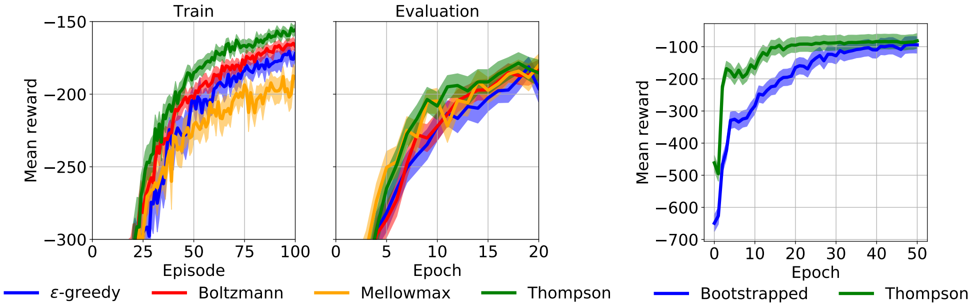
Figure 5.1: *Results on the* ***Taxi*** *problem. Figure 5.1(a) shows the performance of the online variance estimation in terms of mean reward per step. The plots on the left show results using the setting of the policies parameters that is optimal in evaluation; the ones on the right consider parameters 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 mean reward per step.*

5.5 Other results

We conduct further studies on the performance of the different algorithms described in Section 5.3 on the Taxi problem. Firstly, Figure 5.3 shows how the upper bounds to the variance can help learning and also how χ^2 upper bound with boost learns the optimal policy faster. In Figure 5.4 is shown 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 to speedup the learning, provided that it is set to a sufficiently high value in order to avoid suboptimal performance. Eventually, in Figure 5.5 we show the improvement given by the χ^2 upper bound to the Hoeffding bound with $c = 2$.

5.5.1 Continuous state space

We test the online variance estimation extension to continuous state spaces and bootstrapping on the well-known **Mountain Car** and **Acrobot** problems. The need for exploration is given by the sparse reward function, even if exploration is not highly critical as in the Taxi problem. 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 10) for the continuous case in this problem, while we consider the more complex Acrobot for bootstrapping. Figure 5.2 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 5.2(a) highlights the effectiveness of exploration driven by TS during



(a) Mountain Car

(b) Acrobot

Figure 5.2: *Figure 5.2(a) shows the mean cumulative reward in train and evaluation for Mountain Car, while Figure 5.2(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.*

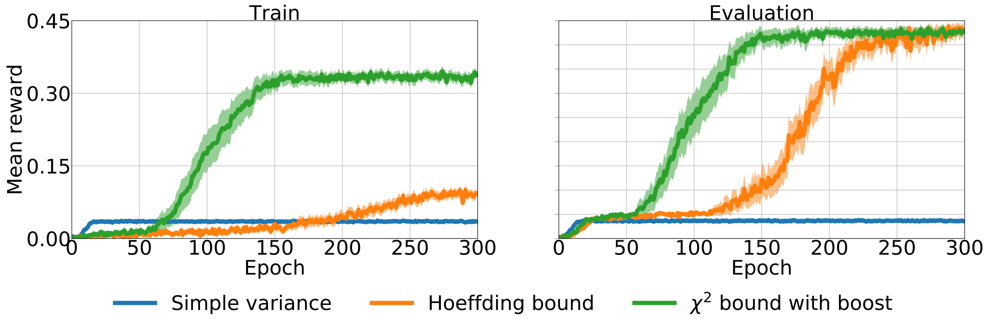


Figure 5.3: *Results of TS on Taxi with different upper bounds on variance.*

the training phase.

5.5.2 Deep Reinforcement Learning

We evaluate TS via bootstrapping in DRL considering the **Atari** games [8] 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 evaluating only these two games, that 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-parameters 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.

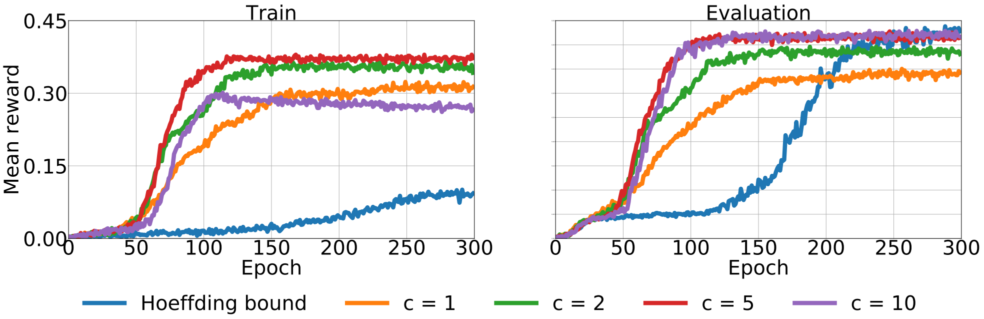


Figure 5.4: *Results of TS on Taxi using Hoeffding upper bound with different values of the c parameter.*

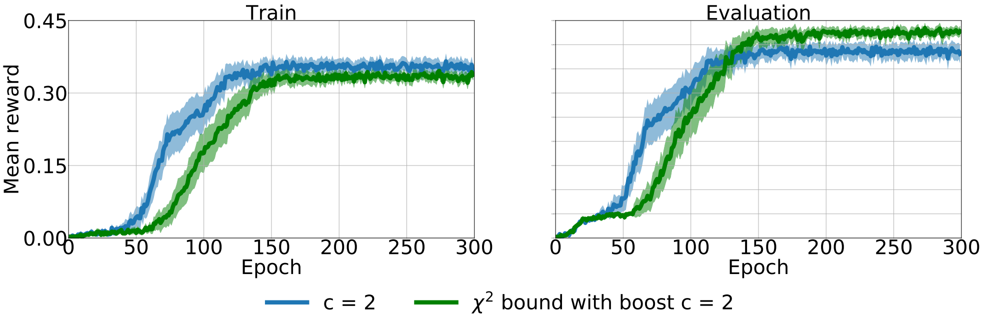
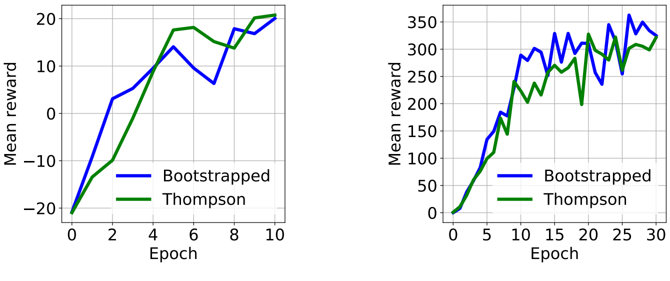


Figure 5.5: *Results of TS on Taxi using Hoeffding upper bound with $c = 2$ without and with χ^2 upper bound boost.*



(a) Pong (b) Breakout

Figure 5.6: *Mean cumulative reward in evaluation. An epoch is performed every 250000 steps.*

CHAPTER 6

Exploration Driven by an Optimistic Bellman Equation

The complexity of exploration in RL is, among other reasons, explained by the presence a sparse reward function; indeed before seeing the states yielding a sparse reward the agent does not necessarily have much information to base its decisions on. In this particular case, the attempt to address the *exploration/exploitation* trade-off (near)-optimally makes no sense since the agent has no information to reason about possible rewards that it has not yet observed. In this setting, classical exploration approaches such as ϵ-greedy may fail as the probability of reaching the positive reward can be low. A more effective strategy should take into account the underlying uncertainty and try to minimize it, in order to maximize the information gain. Bayesian approaches consider the uncertainty in a principled way but are often computational demanding [26,83]. Recently, computationally feasible algorithms inspired by Bayesian principles have been introduced such as Bayesian DQN [6] and BDQN [52] discussed in Chapter 5. However, to the best of our knowledge, there is no algorithm among these approximate techniques that is particularly suited for very sparse rewards in high-dimensional state space. Our hypothesis is that Bayesian methods are in general more focused on balancing between exploration and exploitation while they cannot achieve deep exploration. The broad category of algorithms based on Intrinsic Motivation (IM) [70], have less theoretical guarantees than Bayesian approaches, yet they have obtained impressive results for example in the challenging Montezuma's Revenge task [9]. IM algorithms define an additional *intrinsic* reward, which acts as an exploration bonus. Often, the additional reward is defined using heuristics, such as counting state visits and rewarding less visited states [55], or by *surprise* which is the error in predicting future states [56]. However, the drawback of IM techniques is their lack of a principled definition of the intrinsic reward for exploration.

The OFU-based techniques already discussed in Chapter 5 provide an optimistic estimation under uncertainty, encouraging in this way exploration of uncertain region. Optimism can be categorized in

- **optimistic initialization** of the action-value function approximator to an optimistic value, a method proposed first by [74] about which subsequently [29] proved convergence to a near-optimal solution;
- **confidence interval estimation**, such as IEQL+ [48] and UCRL [5], which directly estimates Q-value confidence intervals.

Considering this premise, we worked on the proposal of a novel Optimistic Bellman Operator (OBE). The OBE results in an optimistic Q-value estimate from an ensemble of 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 approximated Bayesian method [26, 83], we estimate the uncertainty with an ensemble [52]; like optimism-based methods [6, 15, 41, 44], we select optimistic estimates, and like IM, we propagate an implicit exploration bonus [63, 70, 86].

6.1 Learning Value Function Ensembles with Optimistic Estimate Selection

Ensemble methods [51] are a prevalent ML technique where 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 the uncertainty of the prediction. 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 the its uncertainty, in order to apply the OFU principle in action selection.

6.1.1 An Optimistic Bellman Equation for Action-Value Function Ensembles

The core of our work consists of a Bellman Operator (BE) for action-value ensembles which incorporates the information about the uncertainty provided by a Q-function ensemble. In more detail, we want to overestimate the action-value functions with the result of encouraging exploration. Thus, we propose an OBE which propagates an optimistic estimate of the action-value function. We want to emphasize that when all the Q-functions of the ensemble are identical, we assume that there is no uncertainty, and under this condition the OBE will behave exactly equivalently to the classic BE. The solution Q^* of OBE is the same of the classic BE. In other words, the OBE differs from the classic BE when it is not satisfied, and more precisely when approximation is introduced either by limited availability of samples and/or functional approximation. This makes sense, since when the perfect solution is available there is no need of optimism and exploration. The optimistic Bellman operator derived from the OBE, enjoys the properties of the classical one, like contractivity and the existence of a unique fixed point, potentially enabling its usage in value-based or actor-critic reinforcement algorithms. The diversity in the Q-value ensemble should be ideally consistent with the uncertainty of the estimation; e.g. 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 Q-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 $\max_m Q_m(s,a)$. However selecting the highest estimate makes poor use of the information provided by the ensemble and can be sensible 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 (i.e., $\max_{b(s,a)} \sum_m b_m(s,a) Q_m(s,a) + c \sum_m p_m \log p_m$); or to bound the information loss (i.e., $\sum_m p_m \log p_m < \psi$). Hard constraint on the information loss is more appealing since the introduced hyper-parameter does not depend on the magnitude of the rewards but has no closed-form solution. In contrast, the penalization weighting constant introduced by the soft-constraint regularization term is sensitive to the magnitude of the rewards, but admits a closed-form solution. We define two different problems where we use an optimistic estimate of the Q-value function.

Entropy-Regularized Optimistic Q Selection.

We define here a Bellman equation over the Q-function ensemble by introducing an optimistic estimate penalized by an entropic regularization term.

Problem 1 (Regularized version)**.**

$$Q_i(s,a) = \max_{b(s,a)\in\mathcal{P}^M} f\big(s,a;b(s,a)\big) - \tfrac{1}{\eta} D_{\mathrm{KL}}\big(p(s,a)\big\|u\big)$$

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

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

with $f(a,s;b) = R(s,a) + \gamma \sum_m b_m(s,a) V'_m(s,a)$, $u_m = 1/M$, $D_{KL}(b(s,a)\|u)$ is the Kullback-Leibler (KL) divergence between the belief $b(s,a)$ and the uniform distribution u, and $V'_m(s,a) = \sum_{s'} P(s'|s,a) \max_{a'} Q_m(s',a')$. The choice of using the relative entropy 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 λ as Lagrangian multiplier for the constraint, we write the Lagrangian

$$\begin{aligned} L_i(s,a) &= f(s,a;b(s,a)) - \frac{1}{\eta} D_{\mathrm{KL}}\big(b(s,a)\big\|u\big) \\ &\quad + \lambda\Big(\sum_m b_m(s,a) - 1\Big). \end{aligned} \tag{6.1}$$

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

$$b_m(s,a) = \frac{e^{\eta\gamma V'_m(s,a)}}{\sum_{k=1}^M e^{\eta\gamma V'_k(s,a)}}. \tag{6.2}$$

By substituting b_m in (6.1), we obtain the solution to the problem (a detailed derivation is provided in the Supplement)

$$Q_i(s,a) = \begin{cases} \overline{R}(s,a) + \frac{1}{\eta}\log\frac{\sum_{m=1}^{M} e^{\eta\gamma V'_m(s,a)}}{M} & \text{if} \eta \neq 0 \\ \overline{R}(s,a) + \frac{\gamma}{M}\sum_{m=1}^{M} V'_m(s,a) & \text{otherwise} \end{cases}. \tag{6.3}$$

Notice that $\eta > 0$ leads to a positive (optimistic) biased estimation, while $\eta < 0$ will 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 η 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 Q 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 [58].

Problem 2 (Constrained version)**.**

$$\begin{aligned} Q_i(s,a) = \max_{b(s,a)\in\mathcal{P}^M} \quad & f(s,a;b(s,a)) \\ \textit{s.t.} \quad & D_{\mathrm{KL}}\big(b(s,a)\big\|u\big) \leq \iota_{\max} \\ & \textstyle\sum_{m=1}^{M} b_m(s,a) = 1 \\ \forall s,a,i \in \mathcal{S}\times\mathcal{A}\times\{1,\dots,M\} \end{aligned}$$

By letting β be the Lagrangian multiplier associated with the KL constraint, we obtain the Lagrangian

$$\begin{aligned} L_i &= f(s,a;b(s,a)) + \beta(D_{\mathrm{KL}}(b(s,a)\|u) - \iota_{\max}) \\ &\quad + \lambda(\sum_m b_m(s,a) - 1). \end{aligned} \tag{6.4}$$

Substituting β with $-1/\eta$ we note that (6.1.1) becomes identical to (6.1) except for a constant factor. Since we can not solve η (or β) analytically, we obtain an approximate solution by iteratively optimizing η (or β) 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: [3] propose an interesting use of the entropic mapping as a soft-max over the action in the Bellman equation; [58] instead obtain it from an entropic regularization over the state-action distribution.

Relation to Intrinsic Motivation In order to highlight the connection between OBE and IM, we reformulate OBE utilizing the unbiased average of the estimates instead of the logsumexp, and by introducing the resulting exploratory bonus U which includes the positive bias

$$Q_i(s,a) \;=\; \overline{R}(s,a) + U(s,a) + \gamma \sum_{m=1}^{M} \frac{V'_m(s,a)}{M} \tag{6.5}$$

with

$$U(s,a) = \frac{1}{\eta} \log \sum_{m=1}^{M} \frac{e^{\eta\gamma V'_m(s,a)}}{M} - \gamma \sum_{m=1}^{M} \frac{V'_m(s,a)}{M}. \tag{6.6}$$

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

$$\begin{aligned} U(s,a) \;&=\; \lim_{N\to+\infty} \frac{1}{\eta} \log[1 + \sum_{n=2}^{N} \frac{(\eta\gamma)^n}{n!} \mathcal{M}_n(s,a)] \\ &=\; \eta\gamma\mathcal{M}_2(s,a) + O(\eta^2) \end{aligned} \tag{6.7}$$

where $\mathcal{M}_n$ is the n^{th} central moment of the random variable V'_m (Proof in Supplement)

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

with

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

Equation (6.5) shows that OBE is equivalent to BE with an additional bonus defined by Equation 6.7. The bonus U (for any positive η) 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 intrinsically motivated algorithms, is that the policy greedy to the obtained Q-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 [75]. In this way we obtain both the intrinsically motivated 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

$$\tilde{Q}_i(s,a) = R(s,a) + \gamma \sum_{m=1}^{M} \frac{\tilde{V}'_m(s,a)}{M} \quad \text{with} \tag{6.8}$$

$$\tilde{V}'_m(s,a) = \sum_{s'} P(s'|s,a) \max_{a'} \tilde{Q}_m(s',a') \tag{6.9}$$

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

$$\begin{aligned} Q_E(s,a) &= \sum_{t=0}^{T} \gamma^t U(s_t, a_t) \quad \text{where} \quad s_0 = s, a_0 = a \\ &= \eta^{-1} \log \frac{\sum_{k=1}^{M} e^{\eta\gamma \max_{a'} \tilde{Q}_k(s',a') + Q_E(s',a')}}{M} \\ &\quad - \frac{\sum_{k=1}^{M} \gamma \max_{a'} \tilde{Q}_k(s',a')}{M}. \end{aligned} \tag{6.10}$$

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 6.3.

6.1.2 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), as shown in the supplement. 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).

Optimistic Q-Learning. Motivated by the idea of employing an ensemble of regressors as is done in BDQN [52], we assume to have M randomly initialized Q-tables. Inspired by the well known Q-learning update rule, we derive an optimistic version which is consistent with the OBE as follows:

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

We show that, with the update rule proposed, 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 $(1-\alpha_t)^n$:

$$\mathcal{M}_{n,t+1}(s,a) = (1-\alpha_t)^n \mathcal{M}_{n,t}(s,a) \tag{6.11}$$

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 interesting considerations: 1) the bonus decrease 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 zero. Further details, including a proof of convergence, are given in the supplemental material[1]. All the considerations done so far provide a deeper insight about how the algorithm works and its properties. However, in a more complex settings, (e.g., function approximation) the convergence to zero of the exploratory bonus is not guaranteed in general.

---

[1] We based our convergence proof for OQL on the works of [47] and [38].

Optimistic Deep Q-Network In addition to the novel OQL algorithm described previously that can be used for limited discrete state spaces, we propose another algorithm for continuous state spaces based on our OBE. We take inspiration from the framework provided by BDQN [52] that uses an ensemble of neural networks as estimator for the Q value function. BDQN minimizes the loss

$$\mathcal{L}_B(s,a)=\sum_{k=1}^{M}(r+\gamma\max_{a'}Q_k^T(s',a')-Q_k(s,a))^2,$$

where Q_k^T is the target network of the k^{th} approximator. To get an unbiased performance evaluation, we decided to update $M-1$ components of the ensemble with the update rule provided by BDQN. We make this choice in order to maintain diversity between the approximations of the ensemble as shown in [**?**]. We use the remaining single 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

$$\begin{aligned}\mathcal{L}_O(s,a)=(\eta^{-1}\log\tfrac{\sum_{k=2}^{M}e^{\eta\gamma\max_{a'}Q_k^T(s',a')+Q_1^T(s',a')}}{M}\\ -\tfrac{\sum_{k=2}^{M}\gamma\max_{a'}Q_k^T(s',a')}{M}-Q_1(s,a))^2\\ +\textstyle\sum_{k=2}^{M}(r+\gamma\max_{a'}Q_k^T(s',a')-Q_k(s,a))^2.\end{aligned}\tag{6.12}$$

The exploratory bonus represented by $Q_E = Q_1$ in the proposed OQL and ODQN algorithms is needed to guide exploration during learning. During 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 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 Q-function approximation. In order to mitigate this problem, we introduce a hyper-parameter χ which denotes the probability to select an optimistic policy π_o in place of the unbiased one π_u. In this way, we can balance the number of unbiased and optimistic samples. Algorithm 12 shows the pseudocode of ODQN.

Automatic Hyper-parameter Adaptation. Recalling that the regularization coefficient η in the OBE is hard to tune, we want to focus our attention on Problem 2. We propose a way, inspired by [64], to optimize η. 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 update η. In more detail, suppose to have N state-action pairs and for each pair (s_i, a_i)

$$\sum_m b_m(s_i,a_i)(\log b_m(s_i,a_i)+\log M)\leq\iota_{\max},\tag{6.13}$$

Algorithm 12 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 (6.12)
    $V \leftarrow V + |$ violated constraints (6.13) in $B_{\text{batch}}|$
  end for
  Let $\rho = \frac{V}{C * \text{batch\_size}}$
  Update $\eta$ by (6.14)
  Update target network
end for
```

where $\iota_{\max}$ is defined in Problem **??**, while $b_m(s_i, a_i)$ is defined by (6.2). We define ρ as the ratio of violated constraints. We update η according to the following rule

$$\eta_{T+1} = \frac{\eta_T}{(0.5 + 10\rho)}. \tag{6.14}$$

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 12 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 positive bonus. For ODQN, as is done in BDQN, we choose to maintain the diversity between approximation, by a random initialization of each component's parameters and by using the bootstrapping technique, so by adding a *mask* in the replay memory which is sampled by and use different data samples per regressor.

6.2 Experimental Evaluation

In the experiments, we compare in the tabular Q-function case our novel OQL with the well-known state-of-the-art bootstrapped Q-learning method (BQL) [**?**], classical Q-learning 2.3.2, and Q-learning with optimistic initialization [74] in the 50-Chain [**?**], Taxi (also known as Maze) [22] and Frozen Lake [**?**] environments. In the case of neural networks approximation, we compare our ODQN method with BDQN and classical DQN in the Taxi and Acrobot environments already described in the previous chapters.

The environments are chosen to cover different types of dynamics, have sparse rewards, and include both discrete and continuous states. For Acrobot and Frozen Lake, we used the implementation provided by OpenAI Gym [**?**]. First, we will discuss the environments in more detail, then we will provide some details on the initialization of the methods, and finally finish with an analysis of the results.

The **N-Chain** environment [**?**] requires a long sequence of non-rewarding actions to achieve the optimal reward. The MDP consists of a chain with N states $\{s_i\}_{i=1}^{N}$, and two actions that move the agent to state s_{i+1} or s_{i-1}. The agent always starts in state s_2. In state s_1 the agent observes a small reward $r(s_1) = 1/1000$, while in the N^{th} state the agent observes $r(s_N) = 1$. The reward function is zero elsewhere. The agent needs to explore until reaching state s_N even if state s_1 looks promising.

The **Frozen Lake** environment is a 8×8 grid-world, in which the agent has to reach a goal position without falling into some holes. The stochastic perturbation of the agent's movement makes the environment challenging.

This problem consists of modeling a car which is placed in a valley, and the power of its engine is not enough to reach the top of the mountain [**?**]. The car needs to build momentum by driving backward. We use the implementation provided by OpenAI Gym [**?**].

Initialization of the Q-functions. In the tabular setting, for optimistically initialized Q-learning (OIQL), we initialize the Q-function to 15 in Taxi and to 1 in N-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 is initialized to 0. In the taxi environment, for both ODQN and BDQN, we use a shared convolutional layer with multiple heads as described in [**?**]. For the Acrobot, each component of the ensemble corresponds to a one-layer neural network. For ODQN, in both the environment, we initialize the output layer corresponding to Q_E to small values of the parameters, in order to obtain initially $Q_E \approx 0$.

Hyper-Parameter Tuning. For the tabular settings, we did not run any hyper-parameter optimization. With neural networks we performed a small grid search over the number of neurons of the network, and whether to use bootstrapping or not. More specifically, we selected hyper-parameters maximizing the mean return averaged over the whole learning curve, using 20 different seeds. In the plots; 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 $\iota_{\max} = 1$. For further details about the implementation of the algorithms, and the grid search, please see the Supplement.

6.2.1 Results

Figure **??** summarizes the results obtained by averaging over 64 different seed the tabular algorithms, and 100 seeds DQN and BDEN (more details, e.g. results for different hyper-parameter settings, can be found in the supplement). OQL learns faster than the other tabular algorithms. In the Chain environment, and in Taxi, our algorithm OQL finds the highest reward faster than Bootstrapped Q-Learning (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 finds high values fast in all the problems

(50-Chain, Taxi, Frozen Lake) using the optimistic Bellman equation while converging to a near-optimal solution as suggested by our convergence proofs.

With neural network approximation of the value function, ODQN outperforms BDQN in the Taxi environment demonstrating that the principles of OBE work even with function approximation. In the Cartpole environment, the learning curves of ODQN and ODQN are nearly identical, possibly, due to the simplicity of the environment.

Part IV

Final Considerations

CHAPTER 7

Mushroom

CHAPTER 8

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

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

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

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

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

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

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

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

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

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

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

[33] 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.

[34] 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.

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

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

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

[38] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994.

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

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

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

[42] 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.

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

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

[45] 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.

[46] 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.

[47] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

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

[49] 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.

[50] 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 multimachine power system. *IEEE Transactions on Industrial Electronics*, 54(1):86–96, 2007.

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

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

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

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

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

[62] 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.

[63] 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.

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

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

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

[67] 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.

[68] 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.

[69] 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.

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

[71] 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.

[72] 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.

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

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

[75] 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.

[76] 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.

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

[78] 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.

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

[80] 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.

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

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

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

[84] 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.

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

[86] 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.

[87] 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.

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

[89] 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.