DEPARTMENT OF MATHEMATICS

Master of Science in Mathematics

Bayesian Methods for Tabular Reinforcement Learning

Supervisor:
Prof. Claudio Agostinelli

Candidate:
Andrea Fox

Academic Year 2020-2021

Abstract

In recent years Reinforcement Learning has become an important field of interest in Artificial Intelligence. Due to its flexibility, it has gained importance in several areas of scientific research and its applications in robotics or finance proved to be very effective. Reinforcement Learning differentiates from the other major paradigms of machine learning (supervised and unsupervised learning) for its ability to learn from repeated interactions with the environment, with the goal of maximizing an appropriately defined reward.

Most of the applications to the real world require some kind of approximation due to the complexity of the problems studied. In the following pages, the focus will be on more simple environments, which can be studied using methods that do not involve any kind of approximation. These methods, called *tabular methods*, represent also the building block for all the approximated ones, and are thus very relevant to understand even the more complex problems.
One of the main issues in the field is the so called *exploration-exploitation dilemma*, that arises as soon as we have to define a strategy to select the actions to execute. This can be summarized by the question we face at each step: is it more convenient to explore and therefore improve our knowledge of the environment, or exploit the available information and choose only those actions that are currently thought to be optimal?
The following chapters will introduce some of the most important methods to train an agent to find an optimal strategy that copes with simple problems, as well as an introduction to some of the exploration strategies that try to deal with the exploration-exploitation dilemma. In particular, of mathematical interest will be the duality between the *model-free* methods and the *Bayesian* ones, with the former that build their optimal strategy through punctual estimates and the latter that make use of appropriate distributions and the Bayesian framework to improve the performances.

Contents

Introduction

Human learning is a process that starts at birth and continues until death as a consequence of ongoing interactions between people and the environment where they live. In Reinforcement Learning we try to teach to a machine how to perform different task by making it interact with the environment.
The focus of this work will be on those algorithms that effectively learn how to solve problems of scientific or economic interest and we will evaluate different designs through mathematical analysis and computational experiments. The main goal of Reinforcement Learning is to learn which action yields the highest reward, merging a simple trial and error approach, with a more elegant approach derived from optimal control problems.

To formally describe the environment and the behavior of the agent, we will make use of Markov Decision Processes. Intuitively, they are able to describe the three most important aspects that the learning agent will have to face to solve the real problem. In particular, it must be able to understand the state of its environment to some extent and must be able to take actions that affect the state.
Reinforcement Learning can be included in the group of *Machine Learning*, however it is different from both *supervised* and *unsupervised* learning, the two most important paradigms. In particular, it is different from the former, as in the problems studied it can be impractical to have examples of the desired behavior that are correct and, at the same time, representative of the situations in which the agent has to act. At the same time, Reinforcement Learning is also different from unsupervised learning, as the goal is not to find hidden patterns in the data, but rather to maximize a reward somehow associated to the environment.
These are some of the reasons for which usually Reinforcement Learning is considered a third paradigm of machine learning, alongside the two most used methods.

One of the main challenges of Reinforcement Learning is the problem known as *exploitation-exploration dilemma*. Each learning agent has a goal, which is to maximize the reward associated to the environment, however the experience, and therefore a good estimate of the return of each action, is build only by executing all the actions. At some point, the agent has to decide whether it is more important to *explore*, thus improving all the estimates, or

to *exploit* the current information and follow the action that is considered the best one in that particular moment. Unfortunately this problem is still unsolved, but many methods to reduce its negative effect have been created.
One of the most exciting aspect of Reinforcement Learning is its ability to adapt to a broad field of applications, that go from robotics to playing board games (even complex ones as Chess or Go) or even the most modern and complex video games. Complex algorithms can also find applications in healthcare and finance.

The field of reinforcement learning as it is known today, dates back to the end of the 1980s: before that, the research related to the topic was divided into two well separated groups, one focusing one learning by trial and error, and one concerning the problems of optimal control and its solution using value functions and dynamic programming.
The latter originated in the mid-1950s, through the work, among others, of Richard Bellman. The class of methods for solving optimal control problems known as *Dynamic Programming* was introduced in Bellman (1956). The same author, later that year (Bellman, 1957), applied the ideas of the Markov Decision Processes to the optimal control problem. From that moment, dynamic programming has been extensively developed, with, for instance, extensions to partially observable MDPs and the introduction of approximation and asynchronous methods. A modern survey on those methods is found in Bertsekas (2012).
The other thread which lead to the modern Reinforcement Learning, is the one centered on the idea of trial-and-error learning. These ideas date back to the 19^{th} century, with the work of psychologist such as Alexander Bain and Conway Lloyd Morgan. The term *reinforcement* in the context of animal learning, was probably first used in 1928 in the English translation of Pavlov's *Conditioned Reflexes* (Pavlov and Gantt, 1928). Applications of this idea to solving simple problems were developed up to the 1950s, building electro-mechanical machines such as the one described in Ross (1933), capable of solving a simple maze and remember the path through a set of switches. This thread of research, however, slowed down during the 1960s and became relegated to sporadic experiments.
The beginning of the modern Reinforcement Learning can be dated to the introduction of *Temporal-difference learning*, due to the work of Richard Sutton in the late 1970s (Sutton, 1978a,b,c) , and Sutton and Barto at the beginning of the 1980s (Sutton and Barto (1981), Barto and Sutton (1982)). They were the first to build a psychological model of classical conditioning based in temporal-difference learning. The two authors, alongside Anderson, in 1983 introduced the *actor-critic architecture*, one of the first methods that combined temporal-difference learning and trial-and-error threads. Another key step was the introduction of the $TD(\lambda)$ method, in Sutton (1988). This work contained the first instance of the separation between temporal-difference learning and control, as well as a proof of the convergence properties. The merge with the ideas of optimal control, finally, dates back to 1989, with the development of Q-learning, due to Watkins (1989).

The proof of the properties of the Monte Carlo methods was developed in Singh and Sutton (1996), while the Exploring Starts version was introduced in Sutton et al. (1998). The off-policy versions of these algorithms (with the introduction of the importance sampling), were first discussed at the beginning of the 2000s.
As stated above, the introduction of Q-learning dates back to 1989, while all the other main temporal-difference methods were introduced later: Sarsa was developed in Rummery and Niranjan (1994) and its convergence proved in Singh et al. (2000). Expected Sarsa, instead, was independently proved by John (1994) and Van Seijen et al. (2009), with the latter also proving the convergence of the method. Finally, the maximization bias problem was heavily studied in van Hasselt (2010, 2011).
The idea of n-step returns is due to Watkins, who first discussed the problem in the previously mentioned article which intrdouced Q-learning. The work of Cichosz (1994) and van Seijen (2016) showed their practical feasibility. The most recent and relevant result in this field is the $Q(\sigma)$ algorithm, presented in Sutton and Barto (2018) and inspired by the work in De Asis et al. (2018).
Bayesian methods were first introduced in Berry and Fristedt (1985) for the simplified case of the *bandit*. Applications to more complex problems were firstly done by Dearden et al. (1998) , who defined Bayesian Q-learning for more general Markov Decision Processes. Another important work in the context of Bayesian Reinforcement Learning is the one from Strens (2000), who developed a model-based algorithm based on Bayesian statistics. Nowadays the research is mostly focused on the field of approximate methods, which allow us to obtain interesting results in several fields. Some practical and relevant results obtained applying this methods to robotics are found in Kober et al. (2013). Another field in which Reinforcement Learning proved to be very effective is the one of board games, where AlphaGo and AlphaZero (Silver et al., 2018) proved to be much better than the best human players, respectively in the games of go and chess.
A great survey of all the modern Reinforcement Learning algorithms, both in the tabular and approximate case, as well as its applications, is found in Sutton and Barto (2018), which also contains a more complete account of the history of the topic.

This work will focus on model-free methods and in particular on the tabular case: we will assume that the number of states in the environment considered is sufficiently small to be represented in a table. In chapter 1 will be introduced the theoretical framework of Reinforcement Learning, the Markov Decision Processes, and some of the most important ideas that will be used throughout the whole work, such as the distinction between the prediction and control phases. Chapters 2 to 4 will be devoted to the most important methods in traditional Reinforcement Learning: Monte Carlo methods, 1-step Temporal Difference and its generalization to n steps.
In chapter 5 the Exploitation-exploration dilemma will be studied in the simplified case of the Bernoullian bandit: several types of exploration strategies will be showed. Chapter 6

will contain a generalization of the topics of the previous chapter to general Markov Decision Processes, and in particular will introduce some of the ideas of Bayesian methods.
Chapter 7 will finally contain a short description of an alternative approach to tabular problems and an introduction to the broad field of the approximate methods, with some examples of applications of these methods.
In all the chapters, the theoretical results will always be accompanied by several examples that will highlight the differences between the different methods studied.

1

Markov Decision Processes

This chapter is dedicated to defining the building blocks of the reinforcement learning theory and can be divided into two sections: the first one will focus on the theoretical framework of this theory, the Markov Decision processes. The second part will explain how it is possible to solve the problems studied when assuming a complete knowledge of the environment. The rest of the work will focus on solving problems in which the environment is not completely known, however some of the ideas presented in this section, in a simplified situation, will become useful even in the practical reinforcement learning applications.
Sutton and Barto (2018, Chapters 3 and 4) contains a more detailed treatment of the topics discussed.

1.1 Finite Markov Decision Processes

Markov Decision Processes (MDP) can be seen as the formal framing for the problem described in Reinforcement Learning. Are based on two entities: the *agent*, which is the one who does the action and learns, and the *environment*, which contains everything outside of the agent. These two continually interact: the agent chooses an action and then the environment responds, creating a new situation in which the agent has to act. The environment also gives some rewards, which correspond to the quantity that the agent is trying to maximize.

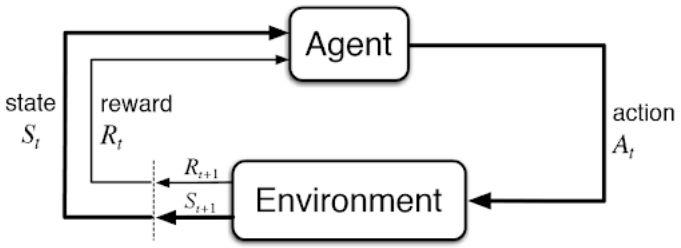


Figure 1.1: Graphical description of the agent-environment interaction in a Markov Decision Process, taken from Sutton and Barto (2018)

More specifically, following the diagram at Figure 1.1, at each discrete time step t the agent observes a representation of the environment, a state S_t, and responds with an action A_t. As a consequence of that, the environment gives the reward R_{t+1} and it also transforms into the new state S_{t+1}. In the whole work we will consider finite Markov Decision Processes, characterized by a finite set of states ($\mathcal{S}$), a finite set of actions ($\mathcal{A}$) and a finite set of rewards ($\mathcal{R}$).
At least for the moment, it is possible to define a function p which determines the *dynamics* of the MDP: it describes how the probability of obtaining a certain state s' and a reward r given the action a and the previous state s:

$$p\left(s', r \mid s, a\right) = \mathbb{P}\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right). \tag{1.1}$$

Note that eq. (1.1) is a probability measure, hence for each couple of s and a we have

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right) = 1 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

From eq. (1.1) it is possible to define further functions that describe the environment.

Definition 1.1. State-transition probability
It is a three argument function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ and defines the probability of transitioning from a state to the other given a certain action:

$$p(s' \mid s, a) = \mathbb{P}\left(S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right) = \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right).$$

Definition 1.2. Expected rewards for state-action pairs
It is a function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r\left(s, a\right) = \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in S} p\left(s', r \mid s, a\right).$$

Definition 1.3. Expected rewards for state-action-next state triplets
It is a function with three arguments, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r\left(s, a, s'\right) = \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \frac{p\left(s', r \mid s, a\right)}{p\left(s' \mid s, a\right)}.$$

One of the major challenges of Reinforcement Learning is defining appropriately the states and the actions that an agent can do and how it can interact with the environment, as well as defining appropriately the environment. The experience suggests that we consider the environment everything that cannot be directly controlled by the agent. Moreover, we do not assume that everything about the environment is unknown to the agent; actually, it is possible to find situations where the agent knows everything about the environment, but at the same time is is difficult to find the optimal strategy. An example of this situation is one where we want to build a robot whose goal is to solve the Rubik's cube: due to the difficulty of the problem, the agent might not be able to solve it.
As stated in Sutton and Barto (2018), in Reinforcement Learning, defining appropriately the states and the actions is more art then science.

1.1.1 Goals and rewards

In Reinforcement Learning the goal of the agent is to maximize the sum of the rewards obtained after each action. This means that the purpose is to maximize the cumulative value in the long term.
The reward does not have to be influenced by the prior knowledge, but rather should only bring to the final goal for which we define the model.
For example, consider an agent which plays chess: in this case the reward does not have to be related to taking an opponent's piece, but rather just on winning the match. Then the agent will learn by itself that in order to win the match one possible strategy is to capture all the opponent's pieces, however the wrong reward might produce an agent that solely focus on taking pieces, even if that brings to a defeat.
So the rewards should be defined to communicate to the agent the result we want to achieve, not how we think it should reach that result. As in the case of defining states and actions, this is no trivial task.

1.1.2 Returns and episodes

As said in the previous section, the goal of Reinforcement Learning is to maximize the cumulative reward. In practice, as it is not possible to know the real return, we try to maximize the expected return, where this value is computed as a function of the sequence of the rewards obtained at each step.

Suppose that there is a final time step T, in this case the return, indicated as G_t, is computed as

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T. \tag{1.2}$$

If for every time that the agent interacts with the environment we get to a final state (called *terminal state*), the process studied is called *episodic*. In this case we can define an *episode* as a subsequence of the interaction between the agent and the environment. For example, if we want to create an agent capable of playing chess (clearly an episodic task, albeit quite complex, as eventually the match will end), an episode will be a match. The terminal state will be the end of the match: a checkmate, for instance. The next episode, so the next match, will begin independently from the outcome of the previous one.
Not every situation can be divided into episodes: a robot with a long life span is one example: in this case, the task is defined *continuing*.
When studying problem of this kind the reward cannot be defined as in eq. (1.2), as we would have to maximize potentially infinite quantities. To cope with this problem it is sufficient to introduce a *discount factor* $\gamma \in [0, 1]$. With this term the reward becomes

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1.3}$$

which is always going to be a finite value for $\gamma < 1$ as long as the rewards are bounded.

1.1.3 Policies and Value functions

In order to achieve the goal of Reinforcement Learning, one strategy involves estimating a functions that describes how good a state (or an action) is. It means that we seek to understand the expected value of the return when the agent is in a state and has chosen a certain action. Clearly, this value is going to be influenced by the *policy*, which maps from states to the probability of selecting each possible action from there.

Definition 1.4. Policy
A policy is a function $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ which indicates the probability that, at time t the agent chooses action A_t while being in state S_t. It is indicated as $\pi\left(A_t \mid S_t\right)$.

Given a certain policy, it is possible to define the value functions that depend on it:

Definition 1.5. State-value function for policy π
Given a policy π, the state-value function of a state s under π is the expected return when starting in s and assuming that the following actions are chosen according to π:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \quad s \in \mathcal{S}. \tag{1.4}$$

Note that if s is a terminal state, then its corresponding value of v_π is 0. The converse may not be true, as the real expected return from a state might be 0 due to the structure of the problem studied.
Similarly we can define the value of taking the action a while staying in s under the policy π:

Definition 1.6. Action-value function for policy π
Given a policy π, the action-value function of action a in state s under π is the expected return when taking action a in s and assuming that the following actions are chosen according to π:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \quad s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned} \tag{1.5}$$

The functions in eqs. (1.4) and (1.5) are going to be estimated from experience, in the form of samples of states, actions and rewards from real or simulated interaction with an environment.
An important property of the value functions is that they can be computed recursively:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \sum_a \pi \left(a \mid s \right) \sum_{s'} \sum_r p \left(s', r \mid s, a \right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi \left(a \mid s \right) \sum_{s', r} p \left(s', r \mid s, a \right) \left[r + \gamma v_\pi(s') \right]. \end{aligned} \tag{1.6}$$

Equation (1.6) can be read as an expected value: for each triple (a, s', r) we compute its probability $\left(\pi \left(a \mid s \right) p \left(s', r \mid a, s \right) \right)$, weight the quantity in brackets for that probability and then sum all over the possible triples to get an expected value. The value in the parenthesis is the (expected) return of the following step. This equation is called *Bellman equation*.
Note that, given a policy π and the function p that describes the dynamics of the environment, the function v_π is the only solution to its Bellman equation. Assuming $\gamma \in (0, 1)$, this can be proved using A.1, as showed in Szepesvári (2010).

1.1.4 Optimal Policies and Optimal Value Functions

To solve a reinforcement learning problem one has to find the policy that at each step maximizes the reward. For a finite Markov Decision Process the following proposition holds.

Proposition 1.1. *For a finite Markov Decision Process it is always possible to find an optimal policy $\pi_\star$.*

A proof of the proposition is found in Szepesvári (2010).
Note that there might be more than one optimal policy, however in that case they all share the same state-value function $v_\star$, called *optimal state-value function*, defined as

$$V_\star(s) = \max_\pi v_\pi(s) \qquad \forall s \in \mathcal{S}.$$

These optimal policies have the same action-value function as well: it is called the *optimal action-value function*, it is indicated as $q_\star$ and it is defined as

$$q_\star(s,a) = \max_\pi q_\pi(s,a) \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Both the optimal value functions have to follow the Bellman equation (eq. (1.6)). In the case of the state-value function we get the *Bellman optimality equation*

$$\begin{aligned} V_\star(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_\star}(s,a) \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\star(s')\right]. \end{aligned} \tag{1.7}$$

A similar equation can be obtained for the action-value function:

$$q_\star(s,a) = \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} q_\star(s', a')\right]. \tag{1.8}$$

If one knows the dynamic of the enviroment (so the function $p\left(\cdot, \cdot \mid \cdot, \cdot\right)$), for a finite Markov Decision processes both eq. (1.7) and eq. (1.8) have a unique solution, which can be found solving the system for all states.
Once one has $v_\star$ it is straightforward to obtain the optimal policy, as it will just be the one that assigns a non zero probability to whichever action has the maximum value from that state. It is then going to be the *greedy* policy with respect to the optimal value function. Meanwhile, retrieving the optimal policy from $q_\star$ is even easier:

$$\pi(a \mid s) = \operatorname*{argmax}_a q_\star(s,a).$$

Unfortunately, this route is rarely feasible, for three main reasons:

(1) it is very hard to know accurately the dynamics of the environment,

(2) in general, the state do not necessarily have the Markov property,

(3) the number of states can be very high and there is not enough computational power to solve the system in a moderately low time.

Regarding the last point, consider the game of chess: if one defines the states as the position of all legal combinations of pieces on the board the number of states is approximately 10^{50}: using this method to retrieve optimal strategy would require solving a system of 10^{50} equations, which is clearly not possible in a reasonable time even with the most advanced technologies available.

1.2 Dynamic Programming

In this section we will consider algorithms that solve problems in a situation where $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ are finite and the dynamic of the environment are known through the probability function $p\left(s', r \mid s, a\right)\ \forall s, s' \in \mathcal{S}, a \in \mathcal{A}, r \in \mathcal{R}$. The ideas described in this section will be used also in the following ones, however the assumptions on the environment are too restrictive, hence the methods will be described very briefly as they rarely can be translated into an application in the real world.

1.2.1 Policy evaluation

First we consider how we can compute the value-function v_π when the policy π is given. Recall the Bellman Equation eq. (1.6):

$$v_\pi(s) = \sum_a \pi\left(a \mid s\right) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi(s')\right].$$

To find the function $v_\pi(s)$ it is possible to define a sequence of functions, $\{v_k\}_{k=0}^{\infty}$, such that each one is, in some sense, closer to the goal than the previous. Such sequence can be defined using the Bellman Equation as an update rule:

$$v_{k+1}(s) = \sum_a \pi\left(a \mid s\right) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v_k(s')\right] \quad \forall s \in \mathcal{S}. \tag{1.9}$$

This update rule has only one fixed point, v_π, and can be shown that the sequence $\{v_k\}_k$ converges to that as k goes to infinity. As seen in Mitchell (2018), the proof is based on the observation that the Bellman Operator

$$F_\pi(v) = \sum_a \pi\left(a \mid s\right) \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma v(s)\right].$$

is a contraction and thus the result is given by the Banach fixed-point theorem.

1.2.2 Policy improvement

The goal of this method is to find the policy that gives, at each step, the optimal action. Given a value function v_π we need to understand if we have to find a new improved policy or not. To evaluate policies, consider the following comparison rule:

Definition 1.7. Given two policies, π and π', it is said that π' is a better policy than π, or $\pi' \geq \pi$ if and only if

$$v_{\pi'}(s) \geq v_{\pi}(s) \quad \forall s \in \mathcal{S}.$$

The following theorem defines further way to compare policies:

Theorem 1.2. ***Policy improvement theorem***
Let π, π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$

$$q_{\pi}\left(s, \pi'(s)\right) \geq v_{\pi}.(s) \tag{1.10}$$

Then the policy π' must be as good as or better than π.
Moreover, if eq. (1.10) is strict at any state, then at those states $v_{\pi'}(s) \geq v_{\pi}(s)$.

Proof. Consider a deterministic policy π and a changed policy π', such that π' is equal to the other for all the states except one: there will be a stets $s \in \mathcal{S}$ such that $\pi'(s) = a \neq \pi(s)$. For all the states for which the policy is equal, (1.10) holds, thus, if $q_{\pi}(s, a) > v_{\pi}(s)$, the changed policy would be better than π.
To prove the theorem, we start from (1.10) and then we expand $q_{\pi}(s, a)$, using its definition, unitl we obtain $v_{\pi'}(s)$.

$$\begin{aligned}
v_{\pi}(s) &\leq q_{\pi}\left(s, \pi'(s)\right) \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s, A_t = \pi'(s)\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_{\pi}\left(S_{t+1}, \pi'(S_{t+1}\right) \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}\left[R_{t+2} + \gamma v_{\pi}\left(S_{t+2}\right) \mid S_{t+1}, A_{t+1} = \pi'\left(S_{t+1}\right)\right] \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots\right] \\
&= v_{\pi'}(s).
\end{aligned}$$

We just proved the simplified case with a change in the policy in a single state. The extension to the general case with an higher number of changes can be proved considering

the new greedy policy π':

$$\begin{aligned}\pi'(s) &= \operatorname*{argmax}_a q_\pi(s,a) \\ &= \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \operatorname*{argmax}_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi(s')\right].\end{aligned}$$

This new policy meets the conditions of equation (1.10), so it is, at least, as good as the original policy π. □

It can further be observed that if two policies π and π' are equally good, then $v_\pi = v_{\pi'}$, from which it follows that

$$\begin{aligned}v_{\pi'}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_{\pi'}(s')\right].\end{aligned} \tag{1.11}$$

Finally, observe how (1.11) is exactly the same as (1.7), implying that both π and π' are optimal policies.
Similar results can also be obtained in the case of stochastic policies.

Corollary 1.3. *Let π, π' be any pair policies such that, for all $s \in \mathcal{S}$*

$$q_\pi\left(s, \pi'(s)\right) \geq v_\pi(s). \tag{1.12}$$

Then the policy π' must be as good as or better than π.
Moreover, if eq. (1.12) is strict at any state, then in those states $v_{\pi'}(s) \geq v_\pi(s)$.

Proof. If the policies are deterministic, then we are in the case of Theorem 1.2, otherwise, observe how $q_\pi(s, \pi'(s)) = \sum_a \pi'\left(a \mid s\right) q_\pi(s,a)$. □

1.2.3 Policy iteration

Once we have found a way to improve π using the value function, we can find a monotone sequence of alternate evaluations of v_{π_k} and improvement from that function. This yields a sequence such as

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \dots \xrightarrow{I} \pi_\star \xrightarrow{E} v_\star$$

where E indicates an evaluation and I an improvement. Moreover, due to the finiteness of the set of states and actions in a finite Markov Decision Process, there is only a finite number of policies. Then this process must converge to the optimal policy and the optimal value function.

This way of finding the optimal policy is called *policy iteration* and the full algorithm is in Algorithm 1.

Algorithm 1: Policy iteration for estimating $\pi \approx \pi_\star$

(1) Initialization
  $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
(2) Policy evaluation
  repeat
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s',r} p\left(s', r \mid s, \pi(s)\right)\left[r + \gamma V(s')\right]$
      $\Delta \leftarrow \max\left(\Delta, |v - V(s)|\right)$
  until $\Delta < \theta$;
(3) Policy improvement
  $policy\text{-}stable \leftarrow true$
  for $s \in \mathcal{S}$ **do**
    $old - action \leftarrow \pi(s)$
    $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V(s')\right]$
    if $old - action \neq \pi(s)$ **then**
      $policy - stabel \leftarrow false$
  if $policy - stable$ **then**
    return $V \approx v_\star$ *and* $\pi \approx \pi_\star$
  else
    go to (2)

1.2.4 Generalized policy iteration

The main drawback of policy iteration is that it requires the policy improvement part until we have found a convergence to the value function v_π. One possible way to solve it consists in letting the policy evaluation part and the policy improvement part interact. This approach is known as *Generalized Policy Iteration* (GPI) and it is one of the most used methods in Reinforcement Learning. This process converges to the optimal value function: if both processes stabilize, then the value function and the policy have to be optimal, as the former stabilizes only when it is consistent with the policy considered, while the latter stabilizes only when it is greedy with respect to the current value function. It follows that both processes stabilize when the Bellman Optimality equation eq. (1.7) holds.
There exist alternative policy iteration processes, which can be seen as intermediate between the first policy iteration seen in Algorithm 1 and the GPI. More information can be found in Sutton and Barto (2018).

2

Monte Carlo Methods

Unlike in the previous chapter, from now on we will not assume complete knowledge of the environment: the methods described will only require experience, obtained by actual or simulated interactions with the environment.Moreover, in this section, we will also consider only episodic processes.
One approach to deal with this framework is to use Monte Carlo methods. Hereafter we will describe these methods, based in the idea of observing the final outcome of the episode to estimate the value associated to a state (or to a state-action pair).
The contents of this chapter can also be found in Sutton and Barto (2018, Chapter 5).

2.1 Monte Carlo prediction

Given a state $s \in \mathcal{S}$ and a set of episodes obtained by following π and passing from s, our first goal is to estimate the quantity $v_\pi(s)$. Each occurrence of state s in an episode is called a *visit* to s. Since a state can be visited multiple times during the same episode, we can distinguish between two ways of computing the estimates: in *first-visit MC methods* it is computed as the average of the returns following the first visit to s in each episode, while in *every-visit MC methods* we compute the average of the returns following all visits to a state.
These two methods have different properties; the first-visit algorithm is easier to study, while the every-visit one better suits the function approximation methods.
Algorithm 2 contains the pseudocode for the first-visit Monte Carlo algorithm for estimating v_π:

Algorithm 2: First visit Monte Carlo prediction

input : a policy π to be evaluated
input : a parameter γ, the discount factor
output: value function V, estimate of v_π

1 **Initialize** $V(s) \in \mathbb{R} \forall s \in S$
2 **Initialize** *Returns(S) <- empty list* $\forall s \in \mathcal{S}$
3 **for** *each episode* **do**
4 | generate an episode following π : $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
5 | $G \leftarrow 0$
6 | **for** *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ **do**
7 | | $G \leftarrow R_{t+1} + \gamma G$
8 | | **if** S_t *is the first visit to state* s *in the episode* **then**
9 | | | Append G to $Returns(S_t)$
10 | | | $V(S_t) \leftarrow average(Returns(S_t))$
11 | | **end if**
12 | **end for**
13 **end for**

It can be observed that the initialization of $V(s)$ is irrelevant: in general, all values will be initialized to 0. The only difference between the *first-visit* algorithm and the *every-visit* one is in the final lines: in the latter, we do not have the `if` condition in line 7, as all the visit to the state s will be considered.

Theorem 2.1. ***Convergence of First visit algorithm***
The estimate $V(s)$ computed using the first-visit Monte Carlo prediction, converges to $V_\pi(s)$ as the number of visits to s goes to infinity.

Proof. The building block of this proof is the law of large numbers. First of all we notice how each value of $Return(S_t)$ is an i.i.d. estimator of $V_\pi(s)$ with finite variance: the independence is due to the fact that each sample is obtained independently from different samples, while the identically distributed part can be proved by noticing that each sample is obtained following the same policy π. Finally, the variance is finite, as the process studied is episodic and hence the number of steps is finite.
We have then showed that we can use the law of large numbers: by doing so, we get that the average of the returns converges to the value function computed in the state s. □

As shown in Singh and Sutton (1996), a similar result can also be proved for the *every-visit algorithm.*

2.2 Monte Carlo estimation of Action Values

In a situation as the one studied in this chapter, where the environment is unknown, it can be more useful to consider the estimates of the action values. This means that the problem of estimating the *state-value function* can be turned into one where we try to compute the estimate of the *action-value function*. The algorithms will not change, except for the fact that we will consider the couples *(state, action)* and not only the states on their own. In the *first-visit MC algorithm*, analogous to the one presented in Algorithm 2, only the first visits to each couple $(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ will be considered. As previously, the algorithm converges to q_π as the number of visits to each pair (s, a) gets to infinity.
The main issue with this approach is that many pairs may never be visited, especially when the policy π is deterministic. By following only that, it might be impossible to reach certain couples, hence not allowing any improvements of the estimate of q_π. In order to solve this problem, we should find a way to visit all the possible couples to improve all the estimates of the action-value function. One possible solution is to work on the distribution of the initial states and the initial action.
This idea brings us to a new algorithm, called *Monte Carlo with Exploring Starts*, which assigns a non zero probability of being selected at the start to each couple state-action. This guarantees that, with infinite samples, each couple will be visited infinitely many times and its estimate will be correct. The pseudocode is in Algorithm 3.

Algorithm 3: Monte Carlo with Exploring Starts for prediction

input : a policy π to be evaluated, $\gamma \in [0, 1]$ discount factor
output: value function Q, estimate of q_π

Initialize $Q(s, a) \in \mathbb{R} \forall s \in S, a \in \mathcal{A}$
Initialize *Returns(s, a) <- empty list* $\forall s \in \mathcal{S}, a \in \mathcal{A}$
for *each episode* **do**
    Choose $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}$ randomly such that all couples have a probability greater than 0
    generate an episode following π, starting from $S_0, A_0 :\ S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow R_{t+1} + \gamma G$
        if S_t *is the first visit to the couple* (S_t, A_t) *in the episode* **then**
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
        end if
    end for
end for

In any case, this algorithm does not completely solve our problems, as we still need

infinite simulations in order to have good estimates of $q_\pi(s, a)$.

2.3 Monte Carlo control

To solve the issues of the Exploring Starts algorithm, it is necessary to consider both the policy and the value function and evaluate both simultaneously, following the idea of the *Generalized Policy Iteration*.
The policy improvement part is done by choosing the greedy improvement policy with regard to the current policy: since we are considering the action-value function we do not need to know the model to construct the greedy policy. Indeed, for any $s \in \mathcal{S}$, the action suggested by the greedy policy is the one which maximizes the action-value function in that state:

$$\pi(s) := \operatorname*{argmax}_a q(s, a) \quad \forall s \in \mathcal{S}.$$

The policy improvement can then build by constructing each π_{k+1} as the greedy policy with regard to the value function v_{π_k}. Moreover, with this choice, the policy improvement theorem holds (Theorem 1.2), as

$$\begin{aligned} q_{\pi_k}\left(s, \pi_{k+1}(s)\right) &= q_{\pi_k}(s, \operatorname*{argmax}_a q_{\pi_k}(s, a) \\ &= \max_a q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

This implies that π_{k+1} is a better policy than π_k: as the number of deterministic policy is finite, we get that the process converges to the optimal policy and to the optimal value function.
This shows that the Monte Carlo algorithm converges to the optimal values, however we have done a couple of unlikely assumptions that make the process studied up to this point almost useless in the real world. One was that the episodes have exploring starts, while the second was assuming that the policy evaluation could be done with infinite steps.
To solve the second problem, we might, for example, give up trying to complete the policy evaluation at each step and evaluate only a finite number of episodes between each step of the policy improvement: using the GPI, choosing only one evaluation of the value function at each step, as in the context of Monte Carlo algorithms, it is very natural to alternate between an evaluation of the value function and an improvement of the policy at every episode.
Algorithm 4 contains the Monte Carlo with Exploring Starts algorithm for the control part:

Algorithm 4: Monte Carlo with Exploring Starts for estimating $\pi \approx \pi_\star$

input : discount factor $\gamma \in [0, 1]$
output: a policy π, estimate of $\pi_\star$
Initialize *policy* $\pi(s) \in \mathcal{A}(s)\ \forall s \in \mathcal{S}$
Initialize $Q(s, a) \in \mathbb{R} \forall s \in S, a \in \mathcal{A}$
Initialize *Returns(s, a) <- empty list* $\forall s \in \mathcal{S}, a \in \mathcal{A}$
for *each episode* **do**
  Choose $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}$ randomly such that all couples have a probability greater than 0
  generate an episode following π, starting from $S_0, A_0 :\ S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow R_{t+1} + \gamma G$
    if S_t *is the first visit to the couple* (S_t, A_t) *in the episode* **then**
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \underset{a}{\text{argmax}} Q(S_t, a)$
    end if
  end for
end for

With this method, all the returns of a certain couple *(state, action)* are averaged, irrespective of the policy used when the respective episodes were built.
It is easy to see that Monte Carlo ES cannot converge to any policy which is not the optimal one: if it did, then the value function would eventually converge to the value function for that policy, and that in turn would cause the policy to change once more. This implies that stability is achieved only when both the policy and the value function are optimal. However, the convergence to the optimal policy has not yet been formally proved, even though it seems inevitable.

2.4 Monte Carlo Control without Exploring starts

The research community has developed two alternative ways to avoid exploring starts that guarantee a convergence to an optimal policy. In this section, the first method will be presented, while the second one will be the topic of next section.
Before getting into the details of the algorithm, we define the following types of policies:

Definition 2.1. Soft policy
Given a constant value $\epsilon > 0$, a policy $\pi : \mathcal{S} \to \mathcal{A}$, is said soft if $\forall s \in \mathcal{S}, a \in \mathcal{A}$

$$\pi(a|s) > 0$$

Definition 2.2. ϵ-soft policy
A policy $\pi : \mathcal{S} \to \mathcal{A}$, is said soft if $\forall s \in \mathcal{S}, a \in \mathcal{A}$

$$\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|},$$

where $\mathcal{A}(s)$ indicates the space of possible actions from the state s.

The method presented in this section proposes the use of the constant-ϵ-greedy policy, which can be seen as the closest $\epsilon - soft$ policy to the actual optimal policy $\pi_\star$.

Definition 2.3. Constant-ϵ-greedy improvement
Given a constant value $\epsilon > 0$ and an action value function q_π, the constant-ϵ-greedy policy with respect to the action-value function is defined as:

$$\pi_\epsilon(a \mid s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; q_\pi(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise.} \end{cases}$$

The idea used in the control part is still GPI, however the exploring starts will not be considerd and the policy improvements will be searched among the $\epsilon - soft$ policies: in particular, the improved policy will always be a constant-ϵ-greedy one.
The convergence is guaranteed due to the following theorem, which shows that the constant-ϵ-greedy improvement brings us to the best policy among the $\epsilon - soft$ ones. Of course this does not imply that we obtain the optimal policy, but is still a relevant result as we have eliminated the assumption of exploring starts.

Theorem 2.2. *$\forall$ $\epsilon - soft$ policy π, any constant-ϵ-greedy policy π' obtained with regard to the value function q_π is a better policy than π. Moreover, the equality holds only when both π' and π are optimal among the $\epsilon - soft$ policies.*

Proof. To show this result, we will show that $q_\pi\left(s, \pi'(s)\right) \geq q_\pi\left(s, \pi(s)\right) = v_\pi(s) \quad \forall s \in \mathcal{S}$.

Then the *policy improvement theorem* (Theorem 1.2) will give us the desired result.

$$\begin{aligned} q_\pi\left(s, \pi'(s)\right) &= \sum_a \pi'(a \mid s) q_\pi(s,a) \\ &= \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\epsilon)\max_a q_\pi(s,a) \\ &\geq \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\epsilon)\sum_a \frac{\pi(a \mid s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} q_\pi(s,a) \\ &= \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) - \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + \sum_a \pi(a \mid s) q_\pi(s,a) \\ &= v_\pi(s). \end{aligned}$$

The inequality is verified, as the sum in the second term indicates a weighted average with nonnegative weights summing up to 1: the sum is smaller or equal than the largest number averaged.

We can now prove the equality part: consider a new environment which has the same states and actions as the original one, but behaves in the following way:
fix $s \in \mathcal{S}$ and $a \in \mathcal{A}$, then, with probability $1-\epsilon$ the environment behaves as the original one, while with the remaining probability ϵ it picks a new action randomly, with uniform distribution probability. With the new action, the environment behaves as the previous one.
In this new environment each policy can be seen as a constant-ϵ-greedy policy in a deterministic environment, due to the properties defined above. This implies that the general optimal policy in the new environment is equivalent to the optimal constant-ϵ-greedy policy in the original one.
Let $\tilde{v_\star}$ and $\tilde{q_\star}$ be the optimal functions, associated to a general policy, in the new environment. We need to prove that the constant-ϵ-greedy policy $\pi_\star$ is optimal among the $\epsilon - soft$ policies, so that $v_{\pi_\star} = \tilde{v_\star}$.

$$\begin{aligned} \tilde{v_\star}(s) &= (1-\epsilon)\max_a \tilde{q_\star}(s,a) + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a \tilde{q_\star}(s,a) \\ &= (1-\epsilon)\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \tilde{v_\star}(s')\right] + \\ &+ \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \tilde{v_\star}(s')\right]. \end{aligned} \tag{2.1}$$

To obtain the second term, we have applied the Bellmann Equation eq. (1.6).

We also know that for the optimal $\epsilon - soft$ policy $v_{\pi_\star}$ the following equation is true:

$$\begin{aligned} v_{\pi_\star}(s) &= (1-\epsilon)\max_a q_\pi(s,a) + \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) \\ &= (1-\epsilon)\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_{\pi_\star}(s')\right] + \\ &+ \frac{\epsilon}{|\mathcal{A}(s)|}\sum_a\sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_{\pi_\star}(s')\right]. \end{aligned} \tag{2.2}$$

The function $p(\cdot,\cdot \mid \cdot,\cdot)$ which is found in both equations is the same, due to the definition of the environments. The only difference between eq. (2.2) and eq. (2.1) is that in the former we consider $\tilde{v_\star}$, while in the latter we find $v_{\pi_\star}$.
However, it is known that there is only one optimal function which satisfies that equation, meaning that $\tilde{v_\star} = v_{\pi_\star}\ \forall s \in \mathcal{S}$. □

This algorithm is referred to, as *On-policy first-visit MC control for $\epsilon - soft$ policies to estimate the optimal policy.* The term *on-policy* refers to the fact that the policy that we are trying to improve is the same we use to generate samples. Algorithm 5 describes it.

Algorithm 5: On-policy first-visit MC control for ϵ-soft policies

input : a parameter $\epsilon > 0$, discount factor $\gamma \in [0,1]$
output: policy π, estimate of $\pi_\star$

Initialize *policy* $\pi(s) \in \mathcal{A}(s)\ \forall s \in \mathcal{S}$ *arbitrary* $\epsilon - soft$ *policy*
Initialize $Q(s,a) \in \mathbb{R}\ \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Returns(s,a) \leftarrow$ *empty list* $\forall s \in \mathcal{S}, a \in \mathcal{A}$
for *each episode* **do**
  generate an episode following π, starting from $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow R_{t+1} + \gamma G$
    if *It is the first visit to the couple* (S_t, A_t) *the episode* **then**
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
      $A^\star \leftarrow \underset{a}{\text{argmax}} Q(S_t, a)$
      for *all* $a \in \mathcal{A}(S_t)$ **do**
        $$\pi(a \mid S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = A^\star \\ \epsilon/|\mathcal{A}(s)| & otherwise \end{cases}$$
      end for
    end if
  end for
end for

2.5 Off-Policy prediction via importance sampling

Up to this point we have presented on-policy learning methods: this term indicates those methods in which the policy used for the sampling is the same one we want to improve. This approach, however, has an important flaw: it tries to learn the action values conditional on subsequent *optimal* behavior, but at the same time needs to behave non-optimally in order to explore all actions with the goal of finding the optimal actions.
With the off-policy learning we cope with this problem by considering two separate policies, one that is learned and will get closer to the optimal policy and one that is more exploratory and is used to generate samples. The first one is called *target policy* (π), while the second one is the *behavior policy* (b). The algorithms that consider the two separate policies are called *off-policy learning* algorithms.
In order to use episodes sampled from b to estimate values for π, we need to impose a condition on the two policies. The property is called *coverage* and assuming that it is verified implies that, whenever $\pi(a \mid s) > 0$ we also have $b(a \mid s) > 0$. This implies that the behavior policy has to be stochastic in those states where it is not equal to π and that, at the same time, the target policy can be deterministic.
As we will see in the following section, dedicated to the control part, we will usually assume a greedy target policy and a constant-ϵ-greedy behavior policy. Before getting there, as done in the on-policy case, we will first focus on the prediction part.
One problem with the off-policy estimate is that uses a different policy to sample the episodes and hence the expected value of the final return is not the value function of π, but rather the one associated to the behavior policy b:

$$\mathbb{E}_b\left[G_t \mid S_t = s\right] = v_b(s). \tag{2.3}$$

One solution to this problem is a technique called *importance sampling*, created for estimating expected values under one distribution given samples from another one. We apply importance sampling to off-policy learning by weighting returns according to the relative probability of their trajectories occurring under the target and behavior policies. The ratio between the two probabilities is known as *importance-sampling ratio*.
Before defining this quantity, it is important to remember that, given a starting state S_t, the probability of obtaining a certain trajectory under the generic policy π is given by

$$\begin{aligned}\mathbb{P}\left(A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\right) = \\ &= \pi\left(A_t \mid S_t\right) p\left(S_{t+1} \mid A_t, S_t\right) \pi\left(A_{t+1} \mid S_{t+1}\right) \ldots p\left(S_T \mid S_{T-1}, A_{T-1}\right) \\ &= \Pi_{k=t}^{T-1} \pi\left(A_k \mid S_k\right) p\left(S_{k+1} \mid S_k, a_k\right).\end{aligned} \tag{2.4}$$

where $p(\cdot \mid \cdot, \cdot)$ depends on the dynamics of the environment.
Given eq. (2.4), the importance sampling ratio is given by the ratio between the probability

of the trajectory under the target and the behavior policy formally,

$$\rho_{t:T-1} = \frac{\Pi_{k=t}^{T-1}\pi\left(A_k \mid S_k\right) p\left(S_{k+1|S_k,A_k}\right)}{\Pi_{k=t}^{T-1} b\left(A_k \mid S_k\right) p\left(S_{k+1|S_k,A_k}\right)} = \frac{\Pi_{k=t}^{T-1}\pi\left(A_k \mid S_k\right)}{\Pi_{k=t}^{T-1} b\left(A_k \mid S_k\right)}. \tag{2.5}$$

As showed in (2.3), the expected value of the return obtained by sampling according to the behavior policy cannot be used as an estimate for $v_\pi(s)$, however adding the importance sampling ratio to the equation gives the correct estimate:

$$\begin{aligned}
\mathbb{E}_b\left[\rho_{t:T-1} G_t \mid S_t = s\right] &= \sum_{trajectories} \rho_{t:T-1}\ r(traj)\mathbb{P}\left(traj \mid S_t = s\right) \\
&= \sum_{trajectories} \rho_{t:T-1}\ r(traj)\frac{\Pi_{k=t}^{T-1}\pi\left(A_k \mid S_k\right)}{\Pi_{k=t}^{T-1} b\left(A_k \mid S_k\right)} \\
&= \sum_{trajectories} \Pi_{k=t}^{T-1}\frac{\pi\left(A_k \mid S_k\right)}{b\left(A_k \mid S_k\right)} b\left(A_k \mid S_k\right) p\left(S_{k+1} \mid S_k, A_k\right)\ \ r(traj) \\
&= \sum_{trajectories} \Pi_{k=t}^{T-1}\pi\left(A_k \mid S_k\right) p\left(S_{k+1} \mid S_k, A_k\right)\ \ r(traj) \\
&= \mathbb{E}_\pi\left[G_t \mid S_t = s\right]
\end{aligned}$$

where $r(traj)$ is a function that gives the total reward associated to a trajectory.
So, by adding the importance sampling ratio, we can sample from the behavior policy, defined to be adequately exploratory, and at the same time improve the target policy.

Before proceeding, it is important to introduce some notation that will be used in the following pages:

- first of all, from now on, the time steps of different episodes will be renamed so that they increase across episodes,
- $\tau(s)$ will be the set of time steps at which the state s is visited. If the method considered is a first-visit one, then the set will contain only the time steps of the first visit in each episode,
- $T(t)$ will be the time of first terminations following time t,
- G_t will indicate the return from time t up to $T(t)$,
- $\{G_t\}_{t\in\tau(s)}$ will be the set of the returns of the visits that start in s,
- $\{\rho_{t:T(t)-1}\}_{t\in\tau(s)}$ is the set of the importance sampling ratios corresponding to the returns above defined.

Given the notation defined above, we can define two different methods to estimate the values of $v_\pi(s)$. The first method, called *ordinary importance sampling*, consists in scaling

all the returns by the respective ratios and then average the results:

$$V(s) = \frac{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}G_t}{|\tau(s)|}. \tag{2.6}$$

The alternative way, called *weighted importance sampling*, uses a weighted average to find the estimate:

$$V(s) = \frac{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}G_t}{\sum_{t\in\tau(s)}\rho_{t:T(t)-1}} \tag{2.7}$$

and has value equal to 0 when the denominator is equal to 0. These two different estimators have different properties: in particular, the ordinary importance sampling estimate has unbounded variance and is unbiased in the first visit case and biased in the every visit one (even though the bias converges asymptotically to 0). Meanwhile, the weighted importance sampling method is biased, but its variance converges to zero when assuming bounded returns, even if the variance of the ratios themselves is infinite, as proved in Precup et al. (2001).
To better understand the different behavior of the two estimators, one can think to an example in which, for a certain state we have only one estimate of $v_\pi(s)$. Equation (2.6) gives us a value whose expected value is indeed $v_\pi(s)$, however the value of the importance sampling ratio might influence negatively the result giving a much more extreme value. On the other hand, even though the expected value of the estimate obtained using eq. (2.7) is not correct, the variance is going to be much lower. In practice, the most frequently used method is the every visit estimate computed using the weighted importance sampling method.

Before showing the implementation of the off-policy Monte Carlo prediction method, will be showed two different ways to write incremental formulas for the estimates: by applying them it will be possible to improve the predictions on an episode-by-episode basis without having to compute the estimates at each step. The complete calculations to obtain those formulas will be provided in appendix A.1.
When using the importance sampling method formula, given a state $s \in \mathcal{S}$, we can obtain the following formulation:

$$\begin{aligned} V_n(s) = &\frac{\sum_{t\in\tau_{n-1}(s)}\rho_{t:T(t)-1}G_t}{n-1} + \\ &+ \frac{1}{n}\left(\rho_{t_n:T(T_n)-1}G_{t_n} - \frac{\sum_{t\in\tau_{n-1}(s)}\rho_{t:T(t)-1}G_t}{n-1}\right). \end{aligned} \tag{2.8}$$

Instead, when using the weighted importance sampling, the incremental formula is given by

$$\begin{cases} V_{n+1}(s) &= V_n(s) + \frac{W_n}{C_n}\left(G_n - V_n(s)\right) \\ C_{n+1} &= C_n + W_{n+1} \end{cases} \tag{2.9}$$

where $W_i = \rho_{t_i:T(t_i)-1}$.

We can finally define the *Off-policy Monte Carlo prediction* algorithm. It is one of the most widespread version in practical applications: the pseudocode of the every visit version using eq. (2.9) is found in Algorithm 6.

Algorithm 6: Off-policy first-visit MC prediction for ϵ-soft policies

input : a policy π
input : the discount factor $\gamma \in [0, 1]$
output: value function Q, estimate of q_π

Initialize $Q(s, a) \in \mathbb{R}\ \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $C(s, a) \leftarrow 0\ \forall s \in \mathcal{S}, a \in \mathcal{A}$
for *each episode* **do**
  generate an episode following $b: S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ *while* $W \neq 0$ **do**
    $G \leftarrow R_{t+1} + \gamma G$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} \left(G - Q(S_t, A_t)\right)$
    $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
  end for
end for

2.6 Off-policy Monte Carlo control

We conclude this chapter by showing an off-policy control algorithm, which uses a fixed *behavior policy* and tries to improve the *target policy*.
In general, the second one is a greedy, deterministic, policy, which takes advantage of the information obtained using the soft and exploratory behavior policy. Off-policy Monte Carlo control methods uses the techniques presented in the previous section.
They follow the behavior policy, for which the property of coverage is required, while learning about and improving the target policy. The following pseudo code contains the algorithm based on GPI and weighted importance sampling which estimates $\pi_\star$ and $q_\star$. The target policy $\pi \approx \pi_\star$ is the greedy policy with respect to Q , the estimate of q_π. The behavior policy b can be anything, but in order to assure convergence to the optimal policy, an infinite number of returns must be obtained for each pair of state and action. This can be assured by choosing b to be ϵ-soft. Then the policy π converges to optimal at all encountered states even though actions are selected according to a different soft policy

b , which may even change between or within episodes. In the case described above, the behavior policy is fixed. Algorithm 7 contains the pseudocode for the off-policy first-visit MC control algorithm for constant-ϵ-soft policies.

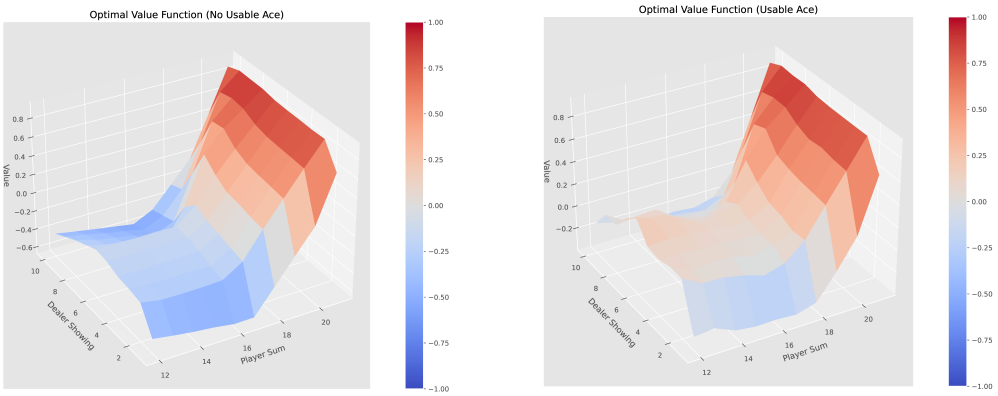
Algorithm 7: Off-policy first-visit MC control for constant-ϵ-soft policies

input : an arbitrary policy π
output: a policy π, estimate of $\pi_\star$
Initialize $Q(s,a) \in \mathbb{R} \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $C(s,a) \leftarrow 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi(s) \leftarrow \underset{a}{\text{argmax}}\ Q(s,a)$
for *each episode* **do**
  $b \leftarrow$ soft policy
  generate an episode following b : $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for *each time step of the episode:* $t = T-1, T-2, \ldots, 0$ *while* $W \neq 0$ **do**
    $G \leftarrow R_{t+1} + \lambda G$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} \left(G - Q(S_t, A_t)\right)$
    $W \leftarrow W \frac{\pi\left(A_t | S_t\right)}{b\left(A_t | S_t\right)}$
    $\pi(S_t) \leftarrow \underset{a}{\text{argmax}}\ Q(S_t, a)$
    if $A_t \neq \pi(S_t)$ **then**
      Exit inner loop
    else
      $W \leftarrow W \frac{1}{b\left(A_t | S_t\right)}$
    end if
  end for
end for

Due to the nature of the greedy *target policy*, if the action chosen by b is not the greedy one, we have that $\pi\left(A_t \mid S_t\right) = 0$, which would result in an importance sampling ratio equal to 0. So, if that happens, we simply exit from the loop, as it makes no sense to continue with the sampling of the episode. At the same time, if A_t is exactly the greedy action, its associated probability with regard to the *target policy* is exactly 1.
It can be observed that, due to this fact, the improvement of the value function (and consequently of the *target policy*) strongly depends on the tails of the episodes. As stated in Sutton and Barto (2018), it is still not clear inside the research community whether this is an issue.

2.7 Example: finding the optimal strategy in blackjack

We will conclude this chapter by showing, through an example, how powerful this methods can be. In particular, in this section it will be showed how to find, using the Monte Carlo methods, the optimal strategy to win at the casino card game of *blackjack*.
The goal of this game is to obtain cards whose sum of the numerical values is as close as possible to 21, without exceeding this value. All face cards (Jack, Queen and King) are worth 10, an ace can count either as 1 or as 11, while all the numerical cards are worth as much as their number indicates. We will consider the version in which each player competes independently against the dealer.
The game begins with two cards dealt to both dealer and player. One of the dealer's cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a *natural*. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (*hits*), until he either stops (*sticks*) or exceeds 21 (*goes bust*). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer uncovers the second card and then hits or sticks, according to a fixed strategy without choice: he sticks on any sum greater or equal than 17, and hits otherwise.
If the dealer goes bust, then the player wins; otherwise, the outcome, (win, lose, or draw) is determined by whose final sum is closer to 21.
It is possible to formulate the game of blackjack as an episodic finite Markov Decision Process, where each game is an episode. Rewards of $+1$, -1, and 0 are given for winning, losing, and drawing, respectively. All rewards within a game are zero, and there is no discount ($\gamma = 1$), therefore these terminal rewards are also the returns. The player has two actions: to hit or to stick and states depend on the player's cards and the dealer's showing card. We assume that cards are dealt from an infinite deck (i.e., with replacement) so that there is no advantage to keeping track of the cards already dealt.
If the player holds an ace that he could count as 11 without going bust, then the ace is said to be usable. In this case it is always counted as 11 because counting it as 1 would make the sum 11 or less, in which case there is no decision to be made because, obviously, the player should always hit. Thus, the player makes decisions on the basis of three variables: his current sum (a value between 12 and 21), the dealer's one showing card (a card from the ace to one whose value is 10), and whether or not he holds a usable ace. This makes for a total of 200 states.
At first we can use the Monte Carlo methods to estimate the value function associated to each state, given a certain policy. For example, assume that the player sticks if and only if the sum of the values of his cards is higher or equal than 20. FIGURE 2.1 shows how, just after 500.000 simulations (around a minute of computations using a consumer laptop), it is possible to have a good approximation of V_π.

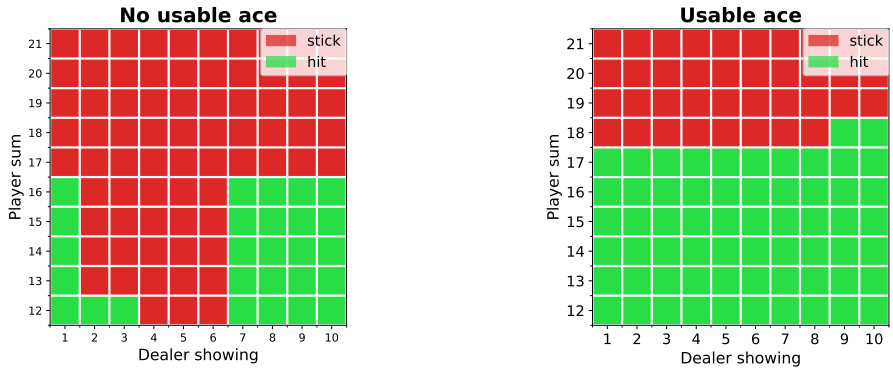


(a) No usable ace case

(b) Usable ace case

Figure 2.1: Optimal value function obtained with a strategy for which the player sticks if and only if the sum of the cards is greater than 20. The following values of the parameter have been used: $\gamma = 1, \epsilon = 0.1$.

However we can also use the control methods to find a better policy, starting from the naive one defined above. At the beginning of the 60s a mathematician called Edward O. Thorp found a way to beat the dealer in blackjack. The strategy is described in his book, *Beat the Dealer: a winning strategy for the game if twenty one* (Thorp (1966)), and was found using the Kelly criterion, a probabilistic formula that determines the optimal theoretical size for a bet, first described in Kelly (1956). Using this formula, Thorp earned a fortune in the 60s visiting casinos, however, after publishing the book its strategy became known to the public and consequently also to Casinos owners, who introduced a new shuffling method to cope with it and avoid losing great quantities of money.
The strategy suggested is the one depicted in the following graphs: on the x-axis we have the value of the uncovered card of the dealer, while the y axis contains the sum of the values of the cards of the players. A green tile indicates that we should hit, while a red one suggests to stick.

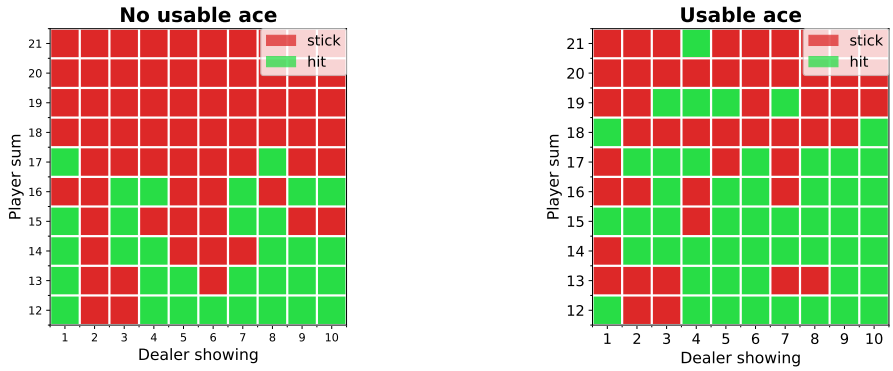


(a) No usable ace case

(b) Usable ace case

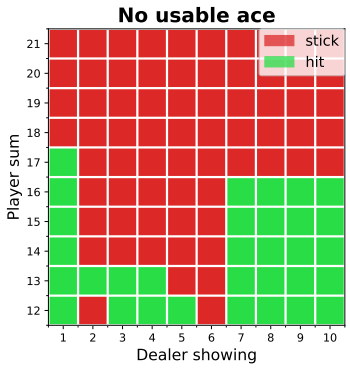
Figure 2.2: Optimal strategy to play blackjack, according to Thorp (1966).

Even though it is no longer possible to get rich using this idea, it can still be used to show the power of Reinforcement Learning and in particular of the Monte Carlo algorithm. Indeed, by using Algorithm 7 and increasing gradually the number of iterations (so, the number of episodes simulated) we can see how the optimal strategy found is always closer to the one suggested by Thorp in Thorp (1966).

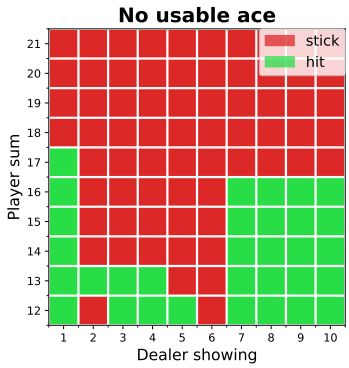


(a) Optimal strategy with 10000 iterations and no usable ace case

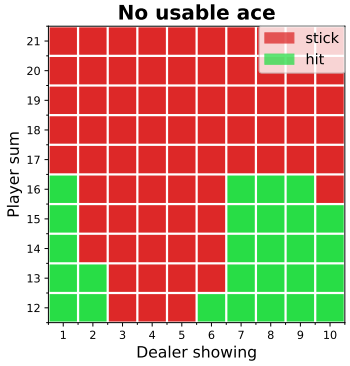
(b) Optimal strategy with 10000 iterations and usable ace case



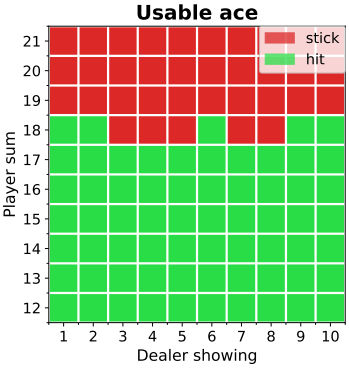
(c) 100.000 iterations and no usable ace case



(d) 100.000 iterations and usable ace case



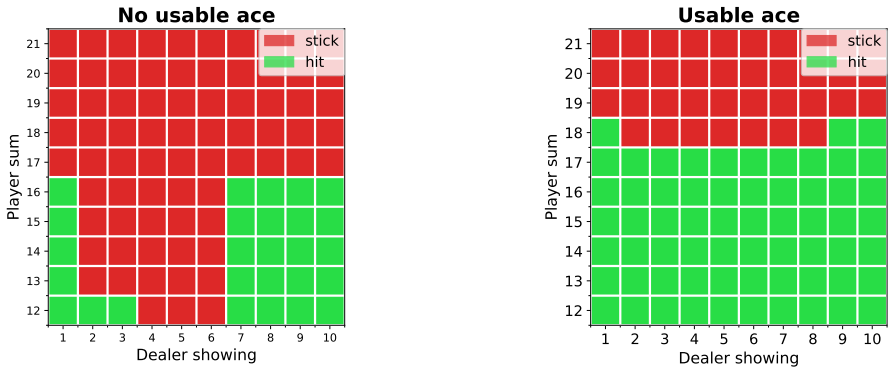
(e) 1 million iterations and no usable ace case



(f) 1 million iterations and usable ace case

Figure 2.3: Different sub-optimal strategies indicated by Monte Carlo method after different numbers of simulations. The results are given assuming $\gamma = 1$ and $\epsilon = 0.1$.

Setting as parameters $\gamma = 1$ and $\epsilon = 0.1$ and repeating for 10 million iterations we strategy obtained is almost identical to the one in Subfigure 2.2b.



(a) No usable ace case

(b) Usable ace case

Figure 2.4: Optimal strategy with 10 million iterations. The results have been obtained assuming $\gamma = 1$ and $\epsilon = 0.1$.

The only difference between the optimal strategy in the usable ace case in Subfigure 2.2b is the leftmost notch in the usable ace case, however, as stated in Sutton and Barto (2018) it is reasonable to assume that the policy showed in Subfigure 2.4b is indeed the optimal one.
It is important to notice that this remarkable results have been obtained using very primitive methods of Reinforcement Learning. Moreover, the computational power used for performing the necessary simulations is not an issue, as all the computations have been done on a single core using a consumer laptop and the most expensive task (simulating 10 million episodes) took around 10 minutes.

3

1-step Temporal Difference Learning

Temporal Difference Learning is one of the most important ideas of Reinforcement Learning: it combines both Monte Carlo and Dynamic Programming ideas. Like the former, they can learn directly from raw experience without needing a model of the environment dynamics, while as DP they can update the estimates based in part on previously learned estimates without waiting for the final outcome.
The temporal difference algorithms can build their estimates on different quantities of future observations: the value of this quantity is what differentiates the *1-step algorithm*, presented in this chapter, and the *n-step algorithm*, the focus of chapter 4.
As usual we start by focusing on the prediction problem, so at first we will assume a given policy π.

3.1 TD prediction

As said in the brief introduction above, TD methods can use experience (and consequently sampling) to solve the prediction problem. Differently from Monte Carlo problems, we only have to wait one time step before giving a new estimate. Assuming the learning rate α is a constant value, the simplest estimate we can get with the temporal difference method is given by

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{3.1}$$

This update rule gives us the *1-step TD method for estimating* v_π described in Algorithm 8.

Algorithm 8: Tabular TD(0) for estimating v_π

input : a policy π
input : step size $\alpha \in (0, 1]$
output: value function V, estimate of v_π
Initialize $V(s) \in \mathbb{R} \forall s \in \mathcal{S}$
Initialize $V(s') = 0 \forall s'$ *terminal state* **for** *each episode* **do**
  Initialize s
  for *each time step of the episode, until s is terminal* **do**
    $a \leftarrow$ action given by π for $\mathcal{S}$
    Take action a, observe r, s'
    $V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$
    $s \leftarrow s'$
  end for
end for

So, we might summarize the idea behind TD by saying that the value estimate for a state is updated on the basis of one sample transition from it to the immediately following state.

3.2 Advantages of TD prediction methods

Temporal difference learning is widely used due to its advantages with regard to both DP and Monte Carlo methods.
In the first case, the advantage is given by the fact that they do not need a model of the environment, of its reward and next-state probability distribution. The advantage of TD methods over Monte Carlo methods is that they can be naturally implemented in an online, fully incremental fashion. With Monte Carlo methods one has to wait until the end of an episode, whereas with TD methods the wait is limited to one time step. This can be a critical consideration, as some applications may have very long episodes or can even be represented by a continuing Markov Decision process, so delaying all learning until the end of the episode might be too slow or even impossible.
Regarding the convergence of such method, has been proved in Sutton (1988), that the method described in Algorithm 8 converges in mean to v_π for constant values of α sufficiently small. Moreover, has been showed in Jaakkola et al. (1994) that the method converges in probability when assuming the following conditions on the step size α_n:

Definition 3.1. Robbins-Monro conditions
A sequence of learning rates $\{\alpha_n\}_n$ is said to fulfill the Robbins-Monro conditions if and only if

(1) $\alpha_n \leq 1 \quad \forall n$

(2) $\sum_{n=1}^{\infty} \alpha_n = \infty$

(3) $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$

There is also an additional question that may rise regarding the convergence: does temporal difference converge faster than Monte Carlo? Unfortunately there is no mathematical proof that one method converges faster than the other, however in practice TD methods have been found to be faster.
In the following example we will show how the TD update rule converges faster when studying a random walk.

Example 1. In this example we will empirically compare the performances of the update rule in eq. (3.1) and an analogous one which uses the Monte Carlo return. We will call this method *constant* $-\ \alpha$ MC:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right] \tag{3.2}$$

where G_t stands for the return of the whole episode that starts in the state S_t.
To compare these two methods we can define a *Markov reward process*, a Markov decision process without the actions, whose graph is the one in Example 1.

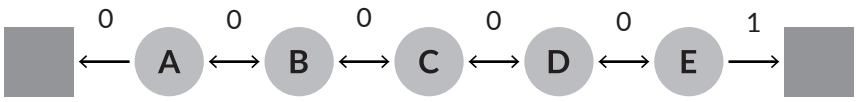


Figure 3.1: Graph used in Example 1 and analogous to the one seen in Sutton and Barto (2018, Example 6.2)

Suppose we always start from the state C and that from each state we can get to the states on the immediate left and right with equal probability. Moreover assume that to each transition is associated a reward of 0, with the exception of the one which brings from E the the adjacent terminal state: in that case the reward is equal to 1.
Assuming this conditions, the expected value of the returns from each state are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}$ and $\frac{5}{6}$ for the states from A to E respectively; this value corresponds to the probability of terminating in the right terminal state when we are in the corresponding state.
To compare the behaviors of the different methods, for different values of the *learning rate* α, we can compute the Root Mean Squared error for each of the estimates of the value function.

Definition 3.2. Root mean squared error
Given two vectors $a = (a_1, \dots, a_n), b = (b_1, \dots, b_n) \in \mathbb{R}^n$, the RMS error is defined as

$$RMS(a, b) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - b_i)^2}$$

Figure 3.2 has been obtained computing the RMS error between the values predicted by either Monte Carlo or Temporal Difference and the true values. The computations have been carried out for all possible values of the number of episodes from 1 up to 100 and for each value 100 different simulations (with different seeds that deal with random choices) have been performed. The results depicted are the average of all the simulations:

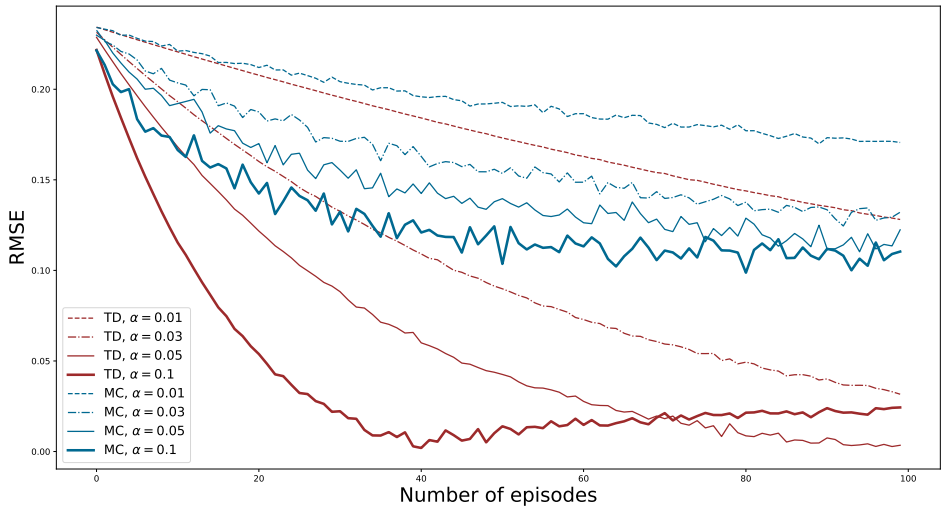


Figure 3.2: Comparison of the RMS error for Temporal Difference and Monte Carlo learning obtained for different values of the parameter α in the environment described in Example 1.

It is clear how, for all the values of the parameters, the Temporal Difference method is the better choice.
Clearly this is not a proof, however, as stated in Sutton and Barto (2018), this same behavior has been found in several other practical applications.
To conclude this example, Example 1 shows how the value function gets closer to the true values as the number of episodes increase. The following results have been obtained using the Temporal difference algorithm and a learning rate $\alpha = 0.05$:

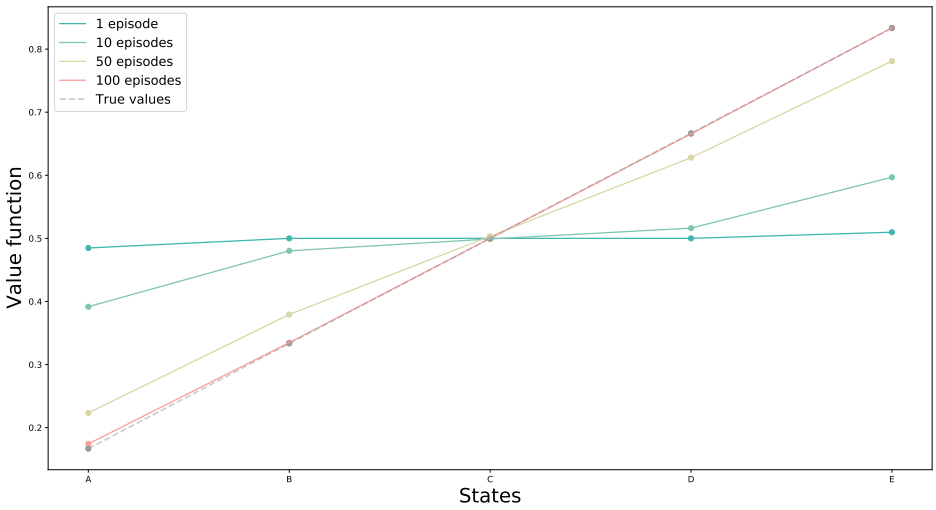


Figure 3.3: Value function computed by Temporal Difference method, with $\alpha = 0.05$ and increasing number of episodes in the environment described in Example 1.

3.3 Optimality of TD(0) in batch updating

Suppose to be in a situation where has been observed a finite amount of experience. A common approach consists in using all the observations to compute, for each one, the increment due to the return of the method chosen. After passing all the examples, the value function can be updated, using all the increments computed before. Then this process is repeated until the value function converges.
Comparing the results obtained using Monte Carlo and Temporal Difference to compute the increments it can observed how, for sufficiently small values of the learning rate α, both functions converge, but to different values. The following example shows how it is possible to get such odd result:

Example 2. Suppose we have the following observations from an unknown Markov process:

$$\{(A, 0, B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 0)\}$$

This suggests that we will have at least two states, A and B. Moreover, our observations suggest that B can bring to a terminal state with two different rewards. The information given by the examples above can be summarized in the following graph:

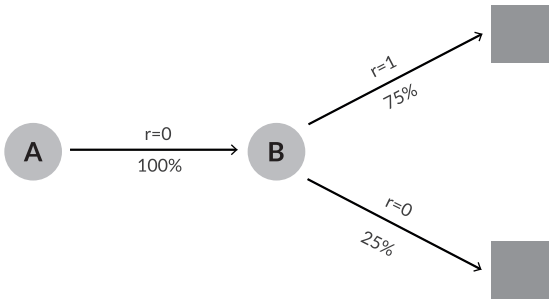


Figure 3.4: Graph that describes the environment used in Example 2 and also seen in Sutton and Barto (2018, Example 6.4)

Given this graph, it is clear how the best estimate for $V(B)$ is $\frac{3}{4}$, as 6 example out of 8 give a reward of 1, while the other two have a reward of 0. Meanwhile, one might propose two different estimates for $V(A)$: the first one is based on the idea that every time we start from A we obtain a reward of 0. The second estimate is based in the idea that every time we visit the state A, then we also get to B, so the estimate of $V(A)$ should be equal to the one obtained for $V(B)$. This two ideas are both correct: indeed, the first one is the answer given by the Monte Carlo batch update method, while the second one corresponds to the estimate of the temporal difference batch method.

Example 2 showed the main difference between the two methods: the Monte Carlo estimate is the one that minimizes the mean squared error in the training set, while the TD method finds the optimal estimate for the Markov process that describes the data.
What the TD method looks for is the maximum likelihood estimate of the Markov decision process that is generated from the examples provided, the one whose transition probabilities are exactly the fractions of the occurrences of that transition in the experience:

$$p_{i,j} = \frac{\#\text{samples that from } i \text{ go to } j}{\#\text{samples that from } i \text{ go somewhere}}$$

Given this model, the estimate of the value function is the one we would have if the model was the correct one. This is known as the *certainty-equivalence estimate*, because it is equivalent to assuming that the estimate of the underlying process was known with certainty rather than being approximate.
Sutton (1988) showed that batch temporal difference converges, in general, to the certainty-equivalence estimate.
Remark 1. If the number of states is $n = |S|$, creating the MLE estimate requires a space equal to n^2 in memory and computing the corresponding value functions costs n^3. This implies that these estimates can be done only with a small amount of states

3.4 Sarsa: TD(0) control

We can now finally focus in the control part and we will follow the idea of the *generalized policy iteration*. As previously showed, in the control part we will use the *action-value* function to improve the policy π. The first method that will be presented is an on-policy algorithm, called Sarsa. Some off policy methods will be presented in the remainder of the chapter.
The update rule of the method is a straightforward application of eq. (3.1):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{3.3}$$

and the update is done after every transition which starts from a non terminal state S_t. From eq. (3.3) it is natural to define Algorithm 9.

Algorithm 9: Sarsa for estimating $Q \approx q_\star$

input : step size $\alpha \in (0, 1]$, $\epsilon > 0$
output: value function Q, estimate of $q_\star$
Initialize $Q(s, a) \in \mathbb{R} \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s', a) = 0 \forall s'$ *terminal state*
for *each episode* **do**
    Initialize s
    Choose a from s, using a policy derived from Q
    for *each time step of the episode, until s is terminal* **do**
        Take action a and observe r, s'
        Choose action a' from s', using the policy derived from Q
        $a \leftarrow$ action given by π for $\mathcal{S}$
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s',\ a \leftarrow a'$
    end for
end for

The convergence of this method depend on the nature of the policy which derives from Q and on the values of α_t.
We will now show the proof of the convergence assuming a constant-ϵ-greedy policy, with $\epsilon = \epsilon_t$ converging to 0, and αs which respect the Robbins-Monro conditions on the step size. The conditions on the learning rate will be required for most of the convergence results.

Theorem 3.1. *Consider a finite state-action Markov decision process and fix an ϵ-greedy learning policy π. Assume that A_t is chosen according to π ant at time step t, π uses $Q = Q_t$, where Q_t are the values computed using the Sarsa update rule*

$$Q_{t+1}(S_t, A_t) \;=\; Q_t(S_t, A_t) \;+\; \alpha_t(S_t, A_t)\left[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)\right] \tag{3.4}$$

Then Q_t converges to $Q_\star$, provided that the following conditions are true:

(1) the Q values are stored in a lookup table

(2) the learning rates satisfy the Robbins-Monro conditions

(3) $\gamma \in [0, 1)$

(4) $Var(R_t) < \infty$ and the rewards are finite

This theorem, as well as other convergence results for other methods that will be presented in this chapter, Lemma 3.2 will be used:

Lemma 3.2. *Consider a stochastic process $(\alpha_t, \Delta_t, F_t), t \geq 0$, where α_t, Δ_t and F_t satisfy the following equation:*

$$\Delta_{t+1}(x) = \left(1 - \alpha_t(x)\right) \Delta_t(x) + \alpha_t(x) F_t(x) \quad x \in X, t = 0, 1, 2, \ldots \tag{3.5}$$

Let P_t be a sequence of increasing $\sigma - fields$ such that α_0 and Δ_0 are P_0-measurable and α_t, Δ_t and F_{t-1} are $P_t - measurable$, $t = 1, 2, \ldots$. Assume the following hold:

(1) the set X is finite

(2) $0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t^2(x) < \infty$ with probability 1

(3) $\|\mathbb{E}\left[F_t(\cdot) \mid P_t\right]\|_W \leq k\|\Delta_t\|_W + c_t$, where $k \in [0, 1)$, c_t converges to 0 with probability 1 and $\|\cdot\|_W$ denotes a weighted maximum norm with weight $W = (w_1, \ldots, w_n), w_i > 0 : if x \in \mathbb{R}^n$ then $\|\cdot\|_W = \max_i |x_i|/w_i$

(4) $Var\left(F_t(x) \mid P_t\right) \leq K\left(1 + \|\Delta_t\|_w\right)^2$

The proof of Lemma 3.2 is provided in Singh et al. (2000).

We will now show the proof of Theorem 3.1:

Proof. The proof is based on Lemma 3.2 and is a special case of the one proposed in (Singh et al., 2000, Theorem 1).

Starting from eq. (3.4) and adding and subtracting the appropriate quantities, it is possible to obtain

$$\begin{aligned} Q_{t+1}\left(S_t, A_t\right) - Q_\star\left(S_t, A_t\right) = &\left(1 - \alpha_t\left(S_t, A_t\right)\right)\left(Q_t\left(S_t, A_t\right) - Q_\star\left(S_t, A_t\right)\right) \\ &+ \alpha_t\left(S_t, A_t\right) F_t\left(S_t, A_t\right) \end{aligned} \tag{3.6}$$

where

$$F_t(S_t, A_t) = R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_\star(S_t, A_t) + \gamma Q_t(S_{t+1}, A_{t+1}) - \gamma \max_a (S_{t+1}, a)$$
$$= F_t^Q(S_t, A_t) + C_t(S_t, A_t)$$

$F_t^Q(S_t, A_t)$ indicates the corresponding term obtained in the proof of Theorem 3.3.
Now, by setting

$$\Delta_t(S_t, A_t) = Q_t(S_t, A_t) - Q_\star(S_t, A_t)$$

we get eq. (3.5).
We define $F_t(s,a) = F_t^Q(s,a) = C_t(s,a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ such that $(s,a) \neq (S_t, A_t)$: this means that we do not update the values for those pairs of states and actions which have not been visited.
Let P_t be the $\sigma - field$ generated by $\{S_t, \alpha_t A_t, R_t, \ldots, S_1, \alpha_1, A_1, Q_0\}$: then $Q_t, Q_{t-1}, \ldots, Q_0$ are P_t-measurable, as well as Δ_t, F_{t-1} are P_t-measurable $\forall t$.
As seen in Theorem 3.3's proof,

$$\left\| \mathbb{E}\left[F_t^Q(\cdot) \mid P_t \right] \right\|_\infty \leq \gamma \|\Delta_t\|_\infty$$

The inequality above gives us

$$\left\| \mathbb{E}\left[F_t(\cdot) \mid P_t \right] \right\|_\infty \leq \left\| \mathbb{E}\left[F_t^Q(\cdot) \mid P_t \right] \right\|_\infty + \left\| \mathbb{E}\left[C_t(\cdot) \mid P_t \right] \right\|_\infty$$
$$\leq \gamma \|\Delta_t\|_\infty + \left\| \mathbb{E}\left[C_t(\cdot) \mid P_t \right] \right\|_\infty$$
$$\leq \gamma \|\Delta_t\|_\infty + c_t$$

To prove point 3 in Lemma 3.2, it is sufficient to show that c_t goes to 0 with probability 1. The quantity c_t can be written as

$$c_t = \|\gamma Q_t(S_{t+1}, A_{t+1}) - \gamma \max_a Q_t(S_{t+1}, a) \mid P_t\|_\infty$$
$$= \gamma \|Q_t(S_{t+1}, A_{t+1}) - \max_a Q_t(S_{t+1}, a) \mid P_t\|_\infty$$

The ϵ-greedy policy considered is by definition greedy in the limit, indicating that, as $t \to \infty$ the non-greedy actions will have a vanishing probability of being chosen. Moreover, the Markov decision process underlying is finite and the quantities $Q_t(S_t, A_t)$ are finite for every value of t.
The finiteness of these quantities comes from finding an upper bound by taking the Q-values of a Q-learning process, while the lower bound is given by the values of an alternative method which is the same as Q-learning but uses the *min* instead of the *max*. Both these

value are limited, therefore also the $Q_t's$ are bounded.
Condition 4 of the lemma is proved as follows:

$$\begin{aligned}
\mathbb{V}ar\left(F_t \mid P_t\right) &= \mathbb{V}ar\left(R_{t+1} + \gamma Q_t\left(S_{t+1}, A_{t+1}\right) - Q_\star\left(S_t, A_t\right) \mid P_t\right) \\
&= \mathbb{V}ar\left(R_{t+1} \mid P_t\right) + \mathbb{V}ar\left(\gamma Q_t\left(S_{t+1}, A_{t+1}\right) - Q_\star\left(S_t, A_t\right) \mid P_t\right) \\
&\leq C + \mathbb{V}ar\left(\gamma \max\left(Q_t - Q_\star\right) \mid P_t\right) \\
&\leq C + \gamma^2 \mathbb{V}ar\left(\max \Delta_t \mid P_t\right) \\
&= C + \gamma^2 \mathbb{V}ar\left(\|\Delta_t\|_\infty \mid P_t\right) \\
&= C + \gamma^2 \|\Delta_t\|_\infty^2
\end{aligned}$$

Recall that R_{t+1} and the remainder of F_t are independent given S_t and A_t, hence are also independent given the whole past history P_t.

□

As stated in Singh et al. (2000) , the convergence of the Sarsa control algorithm can also be proved for other kinds of policies.

3.5 Q-learning

The algorithm introduced in this section is an off-policy one and was first introduced in Watkins (1989). Its update rule is the following:

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right] \tag{3.7}$$

This method is considered off-policy, as the learned action-value function Q, approximates directly the optimal q-function $q_\star$, independently of the policy followed for sampling.
The effect on the policy is just in the choice of the pairs of states and action visited. Algorithm 10 contains the pseudocode of this algorithm.

Algorithm 10: Q-learning for estimating $Q \approx q_\star$

input : $\epsilon > 0$, step size $\alpha \in (0, 1]$
output: value function Q, estimate of $q_\star$

Initialize $Q(s, a) \in \mathbb{R} \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s', a) = 0 \ \forall s'$ *terminal state*
for *each episode* **do**
  Initialize s
  for *each time step of the episode, until s is terminal* **do**
    Choose action a from s, using the policy derived from Q
    Take action a and observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
    $s \leftarrow s'$
  end for
end for

It is also possible to prove the convergence of this method, assuming the same conditions on the learning step α already used in Theorem 3.1.

Theorem 3.3. *Consider a finite state-action Markov decision process and fix an ϵ-greedy learning policy π with $\epsilon = \epsilon_t$ converging to 0. Assume that A_t is chosen according to π and at any time step t, the policy π is based on the values of $Q = Q_t$, where the values of Q_t are computed using the Q-learning update rule*

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t(S_t, A_t)\left[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t)\right] \tag{3.8}$$

Then Q_t converges with probability 1 to the optimal Q-function $Q_\star$ as long as the following conditions are true:

(1) the Q values are stored in a lookup table

(2) the learning rates satisfy the Robbins-Monro conditions

(3) $\gamma \in [0, 1)$

(4) $Var(R_t) < \infty \ \forall t$ and the rewards are finite

Proof. The proof is based on Lemma 3.2 and has first been proposed in Melo (2001). By subtracting from both sides of eq. (3.8) the quantity $Q_\star(S_t, A_t)$, by adding and subtracting on the right side the appropriate quantities and by defining the following:

(1) $\Delta_t(S_t, A_t) = Q_t(S_t, A_t) - Q_\star(S_t, A_t)$

(2) $F_t(S_t, A_t) = R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_\star(S_t, A_t)$

we obtain

$$\Delta_{t+1}(S_t, A_t) = (1 - \alpha_t(S_t, A_t))\,\Delta_t(S_t, A_t) + \alpha_t(S_t, A_t)\,F_t(S_t, A_t)$$

equivalent to eq. (3.5).
We can now build the sequence of $\sigma-$fields P_t as the $\sigma-$fields generated by $\{S_t, \alpha_t, A_t, R_t, \ldots, S_1, \alpha_1, A_1, Q_0$: this yields the condition on the measurability in the lemma.
The assumptions 1 and 2 of the lemma are verified, due to the hypothesis. It remains to prove that also assumptions 3 and 4 are verified in the case studied.
To prove 3 we can use the properties of a contraction H, applying it to Q_t.

$$(Hq)(S_t, A_t) = \sum_{S_{t+1} in \mathcal{S}} P_{A_t}(S_t, S_{t+1}) \left[R_{t+1} + \gamma \max_{b \in \mathcal{A}} Q_t(S_{t+1}, b) \right] \tag{3.9}$$

A general definition of this function, as well as the proof of it being a contraction, is found in appendix A.2.

$$\begin{aligned}
\mathbb{E}\left[F_t(S_t, A_t) \mid P_t\right] &= \sum_{S_{t+1}} P_{A_t}(S_t, S_{t+1}) \left[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_\star(S_t, A_t) \right] \\
&= (HQ_t)(S_t, A_t) - Q_\star(S_t, A_t) \\
&= (HQ_t)(S_t, A_t) - (HQ_\star)(S_t, A_t)
\end{aligned}$$

So,

$$\begin{aligned}
\|\mathbb{E}\left[F_t(S_t, A_t) \mid P_t\right]\|_\infty &= \|HQ_t - HQ_\star\|_\infty \\
&\leq \gamma \|Q - Q_\star\|_\infty = \gamma \|\Delta\|_\infty
\end{aligned}$$

By assuming $c_t = 0\ \forall t$ we prove that assumption 3 holds.
The proof can be concluded by showing that also condition 4 of the lemma holds:

$$\begin{aligned}
\mathbb{V}ar\left(F_t \mid P_t\right) &= \mathbb{V}ar\left(R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_\star(S_t, A_t) \mid P_t \right) \\
&= \mathbb{V}ar\left(R_{t+1} \mid P_t\right) + \mathbb{V}ar\left(\gamma \max_a Q_t(S_{t+1}, a) - Q_\star(S_t, A_t) \mid P_t \right) \\
&\leq C + \mathbb{V}ar\left(\gamma \max\left(Q_t - Q_\star\right) \mid P_t\right) \\
&\leq C + \gamma^2 \mathbb{V}ar\left(\max\left(\Delta_t\right) \mid P_t\right) \\
&= C + \gamma^2 \mathbb{V}ar\left(\|\Delta_t\|_\infty \mid P_t\right) \\
&= C + \gamma^2 \|\Delta_t\|_\infty^2
\end{aligned}$$

□

In Example 3 it is shown an example of the different approach that Q-learning and Sarsa can have on the same problem.

Example 3. Consider a gridworld, in which our goal is to get as fast as possible from the red square to the orange one.

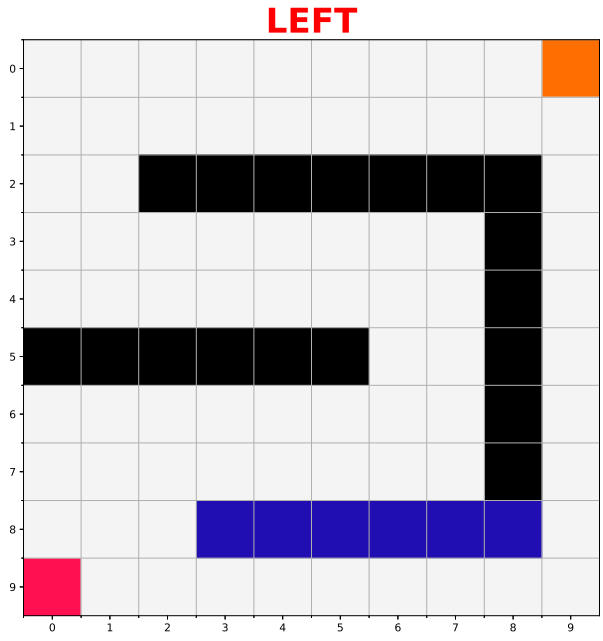


Figure 3.5: Gridworld considered in Example 3

Figure 3.5 shows the environment considered: we start from the red square on the bottom left and the goal is to get in the least amount of steps to the top right, in the orange square. In every position, the agent can move in 4 directions: top,left, bottom and down. To each step is associated a reward of -1, except if we end up in the water (blue squares): in this case the reward is equal to -100 and it also ends the episode. Moreover, if the agent is in a position adjacent to the wall or to the edge of the world and moves in this direction, no movement is registered but the reward associated to the action is still -1. If we choose to have a constant-ϵ-greedy policy and impose $\epsilon = 0.1, \alpha = 0.1$ and $\gamma = 0.95$ we obtain, for different methods, different strategies to reach the objective. The following image contains the optimal strategies obtained with 10000 iterations.

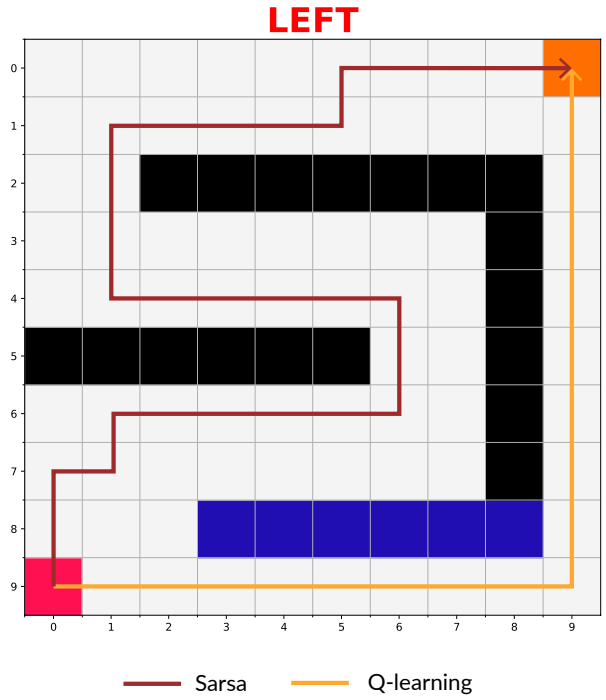
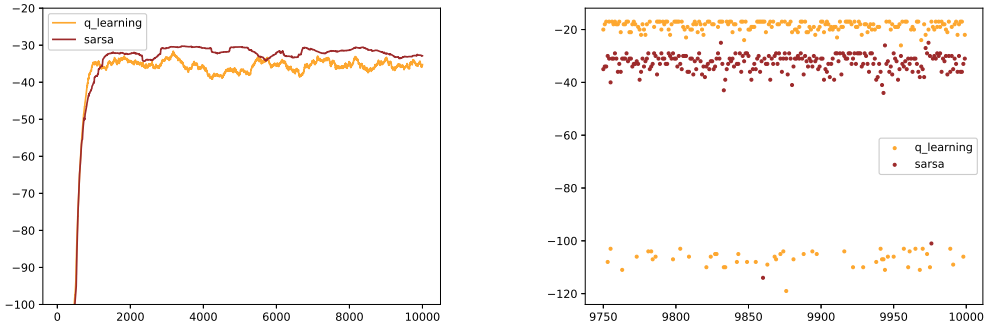


Figure 3.6: Gridworld considered in Example 3, with the trajectories given by the two methods after 100000 episodes. The parameters considered are $\epsilon = 0.1, \gamma = 0.95$ and $\alpha = 0.1$.

It is clear how the strategy indicated by Sarsa is longer, but at the same time safer, as it goes as far as possible from the water blocks, while Q-learning learns the shortest path, even though it passes close to the water.
This difference is due to the different way the two methods compute the return at each step: in Sarsa the second term is based on a sampling on the action, hence it also takes account of the possibility of taking the wrong step and ending up in a water block. Meanwhile Q-learning considers only the value given by the best possible action, hence, when computing the update for the blocks close to the water only takes account of the value of the q-function corresponding to the best action, which does not end in the water and follows the shortest path to get to the goal.
If we look at the value of the returns given by the two methods we can see even more the differences between the two methods: as seen in Subfigure 3.7a, the moving average of the returns given by Sarsa is in general higher, which confirms the idea that this method

is better in an online situation. Meanwhile, Subfigure 3.7b shows the reason behind the higher moving average: when the simulation performed by Q-learning does not end in the water, the final return is much lower than the same result obtained with Sarsa, however it is clear how Q-learning's simulation tend to fall in the water much more often, thus producing the lower moving average.



(a) Moving averages of the return

(b) Final returns

Figure 3.7: Comparison of the returns obtained in 3 using Q-learning and Sarsa, assuming $\epsilon = 0.1, \gamma = 0.95$ and $\alpha = 0.1$.

Clearly, if we reduced the value of ϵ, we would consequently reduce the possibility of taking the wrong action and falling in the water; indeed, for smaller values of the exploration parameter even Sarsa takes the shortest road.

3.6 Expected Sarsa

The algorithm described in this section can be seen as a generalization of Q-learning: given the next state S_{t+1} it considers the expected value of the return, given that state. The update rule is

$$\begin{aligned} Q(S_t, A_t) &\leftarrow (1-\alpha) Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[Q(S_{t+1}, A_{t+1}) \mid S_{t+1} \right] \right] \\ &\leftarrow (1-\alpha) Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) \right] \end{aligned} \tag{3.10}$$

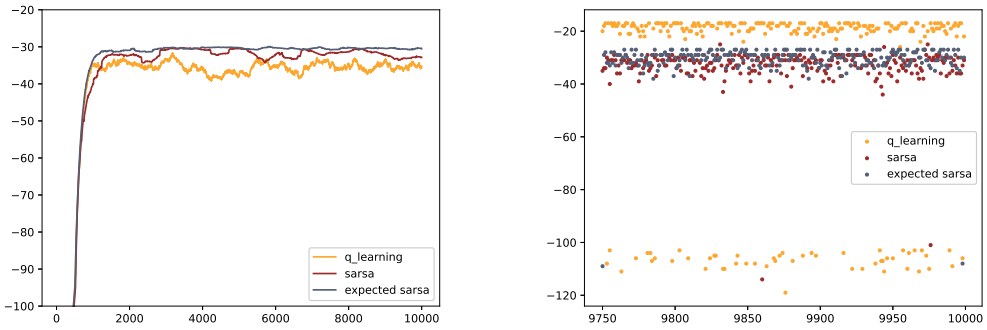
This means that given the next state S_{t+1}, the algorithm moves deterministically in the direction as Algorithm 9 moves in expectation.
If the policy π used in (3.10) is the one we want to improve, then the algorithm is said *on-policy expected Sarsa*, while in the opposite case we talk about *off-policy expected Sarsa*. If π is the greedy policy, then the update rule is exactly the same as the one in Q-learning.

The main advantage of expected Sarsa is that eliminates the variance due to the choice of the random selection of A_{t+1}, however it is more computationally complex than Sarsa. The pseudocode of Expected Sarsa, given in Algorithm 11, is very similar to the one seen in Algorithm 10:

Algorithm 11: Expected Sarsa for estimating $Q \approx q_\star$

input : $\epsilon > 0$, step size $\alpha \in (0, 1]$
output: value function Q, estimate of $q_\star$
Initialize $Q(s, a \in \mathbb{R}\ \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s', a) = 0\ \forall s'$ *terminal state*
for *each episode* **do**
    Initialize s
    for *each time step of the episode, until s is terminal* **do**
        Choose action a from s, using the policy derived from Q
        Take action a and observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma\ \sum_a \pi\left(a \mid S_{t+1}\right) Q\left(S_{t+1}, a\right) - Q(s, a)]$
        $s \leftarrow s'$
    end for
end for

It is possible to repeat experiment of Example 3 and add to the analysis the results obtained with the on-policy version of this algorithm: it is possible to see how the moving average is higher than both the results obtained using Q-learning and Sarsa.



(a) Moving averages of the return

(b) Final returns

Figure 3.8: Comparison of the returns obtained in 3 using Q-learning, Expected Sarsahj and Sarsa, assuming $\epsilon = 0.1, \gamma = 0.95$ and $\alpha = 0.1$.

In particular, we see how the moving average is much more stable. At the same time,

looking at Subfigure 3.8b we see how the single episodes have usually anhigher value than Algorithm 9, indicating even better online performances. Using the same parameters as in Example 3, the optimal path found using Expected Sarsa is the same as the one obtained with Sarsa.
As for the previous methods we can prove the convergence, assuming the conditions in Definition 3.1 for the learning rate $\alpha = \alpha_t$.

Theorem 3.4. *Consider a finite state-action Markov Decision process and fix an ϵ-greedy learning policy π with $\epsilon = \epsilon_t$ converging to 0. Assume that A_t is chosen according to π and at any time step t the policy π is based on the values of $Q = Q_t$, where the values of Q_t are computed using the update rule*

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q_t(S_t, A_t) \right]$$

Then Q_t converges in probability to the optimal value function $Q_\star$ whenever the following assumptions hold:

(1) the Q values are stored in a lookup table

(2) the learning rate respect the Robbins-Monro conditions

(3) $\gamma \in [0, 1)$

(4) the reward function is bounded and has bounded variance

Proof. The proof is based on Lemma 3.2 and has been first proposed in Van Seijen et al. (2009).
The sequence of increasing σ-fields can be built as in Theorem 3.1. Conditions 1 and 2 of the lemma are given by the assumptions, while condition 4 can be proved using the assumption on the reward function and the same observations done in the previous proofs. It remains to show that condition 3 is verified:
the quantity F_t can be defined as

$$F_t = \frac{1}{\alpha_t} \left(\Delta_{t+1} - (1 - \alpha_t) \Delta_t \right)$$

with $\Delta_t(S_t, A_t) = Q_t(S_t, A_t) - Q^\star(S_t, A_t)$
Then the following inequality is verified:

$$\left\| |F_t| \right\|_\infty = \left\| \left| R_{t+1} + \gamma \sum_a \pi_t(S_{t+1}, a) Q_t(S_{t+1}, a) - Q^\star(S_t, A_t) \right| \right\|_\infty$$

$$
\begin{aligned}
&\leq \left\| \left| R_{t+1} + \gamma \max_a Q_t\left(S_{t+1}, a\right) - Q^\star\left(S_t, A_t\right) \right| \right\|_\infty \\
&\quad + \left\| \left| \sum_a \pi_t\left(S_{t+1}, a\right) Q_t\left(S_{t+1}, a\right) - \max_a Q_t\left(S_{t+1}, a\right) \right| \right\|_\infty \\
&\leq \gamma \max_s \left| \max_a Q_t(s,a) - \max_a Q^\star(s,a) \right| + \\
&\quad + \gamma \max_s \left| \sum_a \pi_t(s,a) Q_t(s,a) - \max_a Q_t(s,a) \right| \\
&\leq \gamma \left(\left\| \Delta_t \right\|_\infty + \max_s \left| \sum_a \pi_t(s,a) Q_t(s,a) - \max_a Q_t(s,a) \right| \right)
\end{aligned}
$$

The second inequality is given by the definition of the optimal value function $Q^\star$ and the fact that the maximal difference in value over all states is always at least as large as a difference between value corresponding to a state S_{t+1}. The third inequality is a consequence of

$$\left\| \Delta_t \right\|_\infty = \max_s \max_a \left| Q_t(s,a) - Q^\star(s,a) \right|$$

To conclude the proof, we need to identify the quantity c_t, as

$$c_t = \gamma \max_s \left| \sum_a \pi_t(s,a) Q_t(s,a) - \max_a Q_t(s,a) \right|$$

By setting $k = \gamma$, ad noting how c_t converges to 0 assuming an ϵ-greedy policy, we have proved also this point. □

3.7 Double Q-learning

The algorithms presented up to this point have in common that their target policies are build upon a maximization operation. In Q-learning it is a maximization involving the way the return is computed, while in Sarsa this operation is in the definition of the constant-ϵ-greedy policy.
This operation might create a bias, as we are using a maximum over estimated values to find the estimate of a maximum value. Suppose, for example, to consider a single state, called s where there are many action whose true action value $q(s,a)$ is always equal to 0. The estimate of the q function will be somehow based on a maximum over estimate values and this will probably bring to have only positive estimates. This behavior is known as *maximization bias* and this last algorithm has been introduced in an attempt to mitigate this effect.

This problem might be due to the fact that we use the same samples to both determine the action which maximizes the q-function and to estimate these values. It is possible to eliminate this issue by dividing the plays into two sets and use these two to find independent estimates for Q: we will then have $Q_1(s,a)$ and $Q_2(s,a)$, $\forall a \in \mathcal{A}$. Then we can use one of the functions (suppose Q_1) to find the maximizing action and the other to find the actual estimate, so $Q_2(s, A^\star = Q_2(s, \text{argmax}_a Q_1(s,a))$. This new estimate will then be unbiased, as showed in van Hasselt (2010). The same process can also be repeated for the reversed couple of functions.
This idea of using two different estimates of the function $q(s,a)$ is known as *double learning*. One natural application is to Q-learning: the algorithm that we obtain is known as *Double Q-learning* and its pseudocode is given in Algorithm 12.

Algorithm 12: Double Q-learning for estimating $Q \approx q_\star$

input : $\epsilon > 0$, step size $\alpha \in (0,1]$
output: value function Q, estimate of $q_\star$
Initialize $Q_1(s,a), Q_2(s,a) \in \mathbb{R} \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q_1(s',a) = Q_2(s',a) = 0 \ \forall s'$ *terminal state*
for *each episode* **do**
  Initialize s
  for *each time step of the episode, until s is terminal* **do**
    Choose action a from s, using the policy derived from Q_1 and Q_2 (for example a constant-ϵ-greedy policy)
    Take action a and observe r, s'
    Sample $x \in [0,1]$
    if $x < 0.5$ **then**
      $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha[r + \gamma Q_2\left(s', \text{argmax}_a Q_2(s',a)\right) - Q_1(s,a)]$
    else
      $Q_2(s,a) \leftarrow Q_2(s,a) + \alpha[r + \gamma Q_1\left(s', \text{argmax}_a Q_1(s',a)\right) - Q_2(s,a)]$
    end if
    $s \leftarrow s'$
  end for
end for

The following example, taken from Sutton and Barto (2018, Example 6.7)[Example 6.7]sutton2018reinforcement, shows how in certain situations Double Q-learning is a much better choice than Q-learning, as it is able to eliminate the maximization bias.

Example 4. Consider a Markov decision process with just 4 states, two of which are also terminal. A graphical description of the environment is given in Its graph is the one in Example 4.

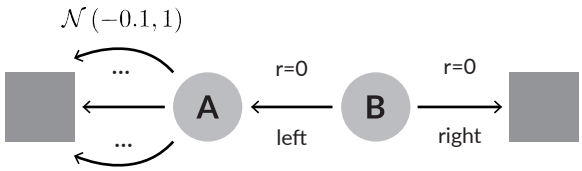


Figure 3.9: Graph used in Example 4 and analogous to the one in Sutton and Barto (2018, Figure 6.5)

The actions starting in B and going left have all a fixed return. All the action starting from B and going left will have a return drawn from a normal distribution with mean -0.1 and variance 1. Moreover suppose that the initial state is always going to be A.
The expected return of going left when starting in A is equal to -0.1, so taking that action is a bad choice. It is possible to show that initially, Q-learning takes the left action more often than the other, even though the expected return of going left is smaller than 0. However, due to the maximization bias, the algorithm, at least initially assigns to those actions a positive estimate, thus bringing to this wrong choice. At the same time, Double Q-learning is never affected by this problem.

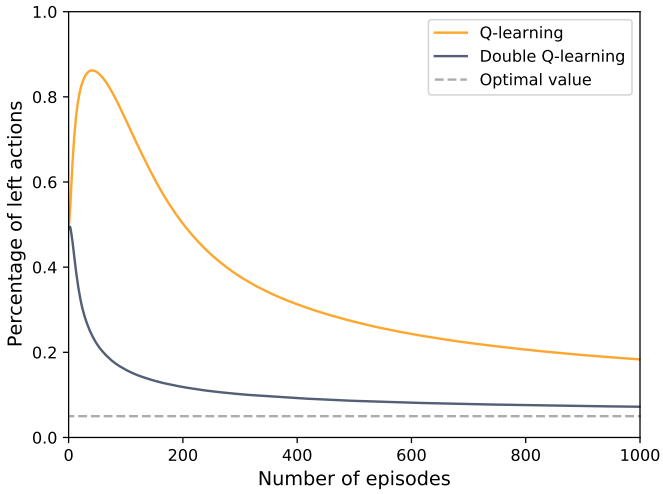


Figure 3.10: Percentage of episode which go left in the first step. These results have been obtained by averaging the results of 5000 simulations of 1000 episode each in the environment considered in Example 4. Both methods assumed $\alpha = 0.1, \gamma = 1$ and $\epsilon = 0.1$.

Another result shown in Figure 3.10 is that, in this case, Double Q-learning is faster in

getting closer to the optimal value of left-actions that should be chosen, which is 5%, due to the choice of the parameters ($\epsilon = 0.1, \alpha = 0.1, \gamma = 0.95$).

As for all the previous algorithms, the section will be concluded by the proof of the convergence of the algorithm under certain conditions.

Theorem 3.5. *Consider a finite state-action Markov decision process and fix an ϵ-greedy learning policy π with $\epsilon = \epsilon_t$ converging to 0. Assume that A_t is chosen according to π and at any time step the policy π is based on the values of $Q = Q_t$, where the values of Q_t are computed using the Double Q-learning update rule, given by either*

$$Q_1(s,a) = Q_1(s,a) + \alpha[r + \gamma Q_2\left(s', argmax_a Q_2(s',a)\right) - Q_1(s,a)]$$

or

$$Q_2(s,a) = Q_2(s,a) + \alpha[r + \gamma Q_1\left(s', argmax_a Q_1(s',a)\right) - Q_2(s,a)]$$

, depending on the episode considered.
Then Q converges to $Q_\star$ with probability 1 as long as the folliwing conditions are true:

(1) the Q-values are stored in a lookup table

(2) both Q_1 and Q_2 receive an infinite number of updates

(3) the learning rate respect the Robbins-Monro conditions

(4) $\gamma \in [0, 1)$

(5) the return is always finite and has finite variance

Proof. The proof is based on Lemma 3.2 and has first been proposed in van Hasselt (2010). As in previous cases, conditions 1, 2 and 4 are given by the hypothesis, the sequence of σ-fields is the one defined in the proof of Theorem 3.1 and it remains to prove condition 3: define $F_t(S_t, A_t)$ as

$$F_t(S_t, A_t) = F_t^Q(S_t, A_t) + \gamma\left(Q_t^B(S_{t+1}, a^\star) - Q_t^A(S_{t+1,a^\star})\right)$$

where F_t^Q is the F_t function defined in the proof of Theorem 3.3. From that proof, it is well known that

$$\mathbb{E}\left[F_t^Q \mid P_t\right] \leq \gamma \left\|\Delta_t\right\|_\infty$$

where $\Delta_t(S_t, A_t) = Q_t^A(S_t, A_t) - Q^\star(S_t, A_t)$, hence to apply the lemma we need to prove that

$$c_t = \gamma\left(Q_t^B(S_{t+1}, a^\star) - Q_t^A(S_{t+1}, a^\star)\right)$$

converges to 0 with probability 1. To do so, define

$$\Delta_t^{AB} = Q_t^B - Q_t^A$$

Depending on whether Q^B or Q^A is updated, the update of Δ_t^{BA} at time is one of

$$\Delta_{t+1}^{BA}(S_t, A_t) = \Delta_t^{BA}(S_t, A_t) + \alpha_t(S_t, A_t) F_t^B(S_t, A_t)$$
$$\Delta_{t+1}^{BA}(S_t, A_t) = \Delta_t^{BA}(S_t, A_t) - \alpha_t(S_t, A_t) F_t^A(S_t, A_t)$$

where $F_t^A(S_t, A_t) = R_{t+1} + \gamma Q_t^B(S_{t+1}, a^\star) - Q_t^A(S_t, A_t)$ and $F_t^B(S_t, A_t) = R_{t+1} + \gamma Q_t^A(S_{t+1}, b^\star) - Q_t^B(S_t, A_t)$.
Now define $\zeta_t^{BA} = \frac{1}{2}\alpha_t$, then

$$\begin{aligned}
&\mathbb{E}\left[\Delta_{t+1}^{BA}(S_t, A_t) \mid P_t\right] = \\
&\qquad = \Delta_t^{BA}(S_t, A_t) + \mathbb{E}\left[\alpha_t(S_t, A_t) F_t^B(S_t, A_t) - \alpha_t(S_t, A_t) F_t^A(S_t, A_t) \mid P_t\right] \\
&\qquad = \left(1 - \zeta_t^{BA}(S_t, A_t)\right) \Delta_t^{BA}(S_t, A_t) + \zeta_t^{BA} \mathbb{E}\left[F_t^{BA}(S_t, A_t) \mid P_t\right]
\end{aligned}$$

where $\mathbb{E}\left[F_t^{BA}(S_t, A_t) \mid P_t\right] = \gamma \mathbb{E}\left[Q_t^A(S_{t+1}, b^\star) - Q_t^B(S_{t+1}, a^\star) \mid P_t\right]$
Assume $\mathbb{E}\left[Q_t^A(S_{t+1}, b^\star) \mid P_t\right] \geq \mathbb{E}\left[Q_t^B(S_{t+1}, a^\star) \mid P_t\right]$: $a^\star$ is, by definition, the action that maximizes $Q_t^A(S_{t+1}, a)$, hence $Q_t^A(S_{t+1}, a^\star) \geq Q_t^A(S_{t+1}, b^\star)$ and therefore

$$\begin{aligned}
\left|\mathbb{E}\left[F_t^{BA}(S_t, A_t) \mid P_t\right]\right| &= \gamma \mathbb{E}\left[Q_t^A(S_{t+1}, b^\star) - Q_t^B(S_{t+1}, a^\star) \mid P_t\right] \\
&\leq \gamma \mathbb{E}\left[Q_t^A(S_{t+1}, a^\star) - Q_t^B(S_{t+1}, a^\star) \mid P_t\right] \\
&\leq \gamma \left\|\Delta_t^{BA}\right\|_\infty
\end{aligned}$$

Assuming $\mathbb{E}\left[Q_t^B(S_{t+1}, a^\star) \mid P_t\right] \geq \mathbb{E}\left[Q_t^A(S_{t+1}, b^\star) \mid P_t\right]$, it is possible to prove in analogous way that

$$\left|\mathbb{E}\left[F_t^{BA}(S_t, A_t) \mid P_t\right]\right| \leq \gamma \left\|\Delta_t^{BA}\right\|_\infty$$

At each time step, one of the two assumptions has to hold, hence it has been showed how

$$\left|\mathbb{E}\left[F_t^{BA} \mid P_t\right]\right| \leq \gamma \left\|\Delta_t^{BA}\right\|_\infty$$

By applying Lemma 3.2 to the stochastic process defined by $\zeta_t^{BA}, \Delta_t^{BA}$ and F_t^{BA}, we get that Δ_t^{BA} converges to 0: this yields the third condition of the lemma also for the stochastic process defined for Double Q-learning.
This proves the convergence of the method. □

Clearly, it is also possible to create double versions of Expected Sarsa and Sarsa.

4

n-step Bootstrapping

In this chapter the main focus are going to be the *n-step TD methods*, in an attempt of unifying the Monte Carlo methods and the Temporal Difference methods. .
Using only the methods presented up to this point, one can update the estimates of the value functions in two ways: when using the 1-step TD methods, the update is done by looking at the value of the following step of the simulation (1 step forward), while using the Monte Carlo methods the update was done after looking at the result obtained at the end of the episode, so after infinite steps forward, as we need to look at the entire sequence of states and rewards. The *n-step methods* can be seen as a midpoint between these two methods: as the name suggests, the update is based on the following n steps of the simulation.
By doing so, we maintain one of the advantages of Temporal Difference, which was the possibility of having an online learning without being forced to wait until the end of the simulation of each episode, but at the same time we are no longer forced to have only one step forward estimates. Moreover, by doing so, we also have a faster convergence towards the optimal value for appropriate values of the parameter n.
As in previous chapters, the first part will be dedicated to the estimate of the value function given a fixed policy, while in the second part the focus will be on the different control algorithms.

4.1 n-step TD prediction

These methods that use n steps forward to make their updates are still considered Temporal Difference methods, as the new estimates will still be based on the previous values obtained. Up to this point the two classes of methods studied had two completely different ways of

computing the target G_t. In particular, assuming an episode which lasts up to time T, the Monte Carlo estimate was

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

while the one obtained with the temporal difference was

$$G_t = R_{t+1} + \gamma V_t \left(S_{t+1}\right) \tag{4.1}$$

with V_t being the estimate at time t of the optimal value function v_π.
In the *n-step* methods we can define the return as $G_{t:t+n}$, where the subscript indicates that the estimate is based on the terms that have been obtained from time t up to $t+n$. These new estimates can be obtained in a natural way: we can rename eq. (4.1) as $G_{t:t+1}$, indicating that it is based only on terms up to time $t+1$:

$$G_{t:t+1} = R_{t+1} + \gamma V_t \left(S_{t+1}\right).$$

The last term is still an estimate and we might substitute it with the return associated to the second step of the simulation (the one going from S_{t+1} to S_{t+2}) and another estimate of V computed in S_{t+2} at time $t+1$. By doing so, we obtain the return using 2 steps of the simulation:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1} \left(S_{t+2}\right).$$

This operation can be iterated and the return after n steps is going to be

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+1} + \gamma^n V_{t+n-1} \left(S_{t+n}\right). \tag{4.2}$$

This definition works as long as $n \geq 1$ and $0 \leq t \leq T-n$. All the *n-step* returns can be considered approximations of the full return, truncated after n steps and corrected with the most recent available estimate of the value function in the state visited at time $t+n$, $V_{t+n-1}\left(S_{t+n}\right)$.
It might happen that in the case of episodic MDPs, the n steps considered go beyond the terminal state ($t+n \geq T$): in this case, the missing terms will be considered equal to 0.

Whenever $n > 1$, eq. (4.2) involves terms which have not yet been discovered at time t: indeed, they will all be available no earlier than at time $t+n$, when the last two terms, R_{t+n} and $V_{t+n-1}\left(S_{t+n}\right)$ will be known. Considering that, the update rule using the *n-step* returns is going to be

$$V_{t+n}\left(S_t\right) = V_{t+n-1}\left(S_t\right) + \alpha \left[G_{t:t+n} - V_{t+n-1}\left(S_{t+n}\right)\right] \quad 0 \leq t < T \tag{4.3}$$

while all the values from the other states will remain unchanged if $s \neq S_t$.
This algorithm is called *n-step Temporal Difference*.
One of its drawbacks is that for the first $n-1$ steps no changes will be made and all the

values will have to be stored. At the same time, an equal number of operations will be made at the end of the episodes, before starting the next episode.
Algorithm 13 contains the pseudocode for the method described.

Algorithm 13: n-step TD for estimating $V \approx v_\pi$

input : a policy π
input : step size $\alpha \in (0, 1]$, $n > 0$ integer
output: value function V, estimate of v_π

Initialize $V(s) \in \mathbb{R} \forall s \in S$
Initialize *All store and access operations for* S_t *and* R_t
for *each episode* **do**
  Choose S_0 non-terminal state and store it
  $T \leftarrow \infty$
  for $t = 0, 1, 2 \ldots,$ *until* $\tau = T - 1$ **do**
    if $t < T$ **then**
      Take an action according to $\pi(\cdot \mid S_t)$
      Observe and store R_{t+1} and S_{t+1}, respectively the next reward and the next state
      if S_{t+1} *is terminal* **then**
        $T \leftarrow t + 1$
      end if
    end if
    $\tau \leftarrow t - n + 1$
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      if $\tau + n < T$ **then**
        $G \leftarrow G + \gamma^n V(S_{t+n})$
      end if
      $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
    end if
  end for
end for

Up to this point, it has been showed how one might use n steps to build an estimate of the value function. The following proposition shows how the expectation of the n-step return is a better estimate than V_{t+n-1} in a worst-state-sense, suggesting that it might be convenient to use the former estimate.

Proposition 4.1. *Let* $G_{t:t+n}$ *be the n-step returns,* V_{t+n-1} *the estimate of the value function at time* $t + n - 1$ *and* $\gamma \in [0, 1]$ *the discount factor. Then*

$$\max_s \left| \mathbb{E}_\pi \left[G_{t:t+n} \mid S_t = s \right] - v_\pi(s) \right| \leq \max_s \left| V_{t+n-1}(s) - v_\pi(s) \right|. \tag{4.4}$$

Equation (4.4) is called *error reduction property*: the quantity $\mathbb{E}_\pi \left[G_{t:t+n} \mid S_t = s \right]$ indicates the expected value of the return obtained using the n-step method, so the left term is the maximum error obtained considering the expected value of the n-step return, hence an estimate of v_π at time $t+n$. Meanwhile, the right term indicates the maximum error we have when considering the estimate of v_π at time $t+n-1$. This indicates that, on average, doing a n-step estimate is a good choice. As noted in Sutton and Barto (2018), this formula is the building block for all the proofs of convergence of the n-step Temporal Difference methods.
We can now formalize the proof of the proposition above:

Proof. A sketch of the proof has been first proposed in Soemers (2019).

$$
\begin{aligned}
\max_s \left| \mathbb{E}_\pi \left[G_{t:t+n} \mid S_t = s \right] - v_\pi(s) \right| &= \\
&= \max_s \left| \mathbb{E}_\pi \left[G_{t:t+n} \mid S_t = s \right] - \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \right| \\
&= \max_s \left| \mathbb{E}_\pi \left[G_{t:t+n} - G_t \mid S_t = s \right] \right| \\
&= \max_s \Big| \mathbb{E}_\pi \Big[\Big(R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}\left(S_{t+n}\right) \Big) - \\
&\qquad\qquad - \Big(R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T \Big) \Big] \Big| \\
&= \max_s \left| \mathbb{E}_\pi \left[\gamma^n R_{t+n+1} + \cdots + \gamma^{T-t-1} R_T - \gamma^n V_{t+n-1}\left(S_{t+n}\right) \mid S_t = s \right] \right| \\
&= \gamma^n \max_s \left| \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-n-1} \gamma^k R_{t+n+k+1} - V_{t+n-1}\left(S_{t+n}\right) \mid S_t = s \right] \right| \\
&\overset{(a)}{\leq} \gamma^n \max_s \mathbb{E}_\pi \left[\left| V_{t+n-1}\left(S_{t+n}\right) - \sum_{k=0}^{T-t-n-1} \gamma^k R_{T+n+k-1} \right| \mid S_t = s \right] \\
&\overset{(b)}{\leq} \max_s \mathbb{E}_\pi \left[\left| V_{t+n-1}\left(S_t\right) - \sum_{k=0}^{T-t-1} \gamma^k R_{T+k-1} \right| \mid S_t = s \right].
\end{aligned}
$$

The inequality indicated as (a) can be verified using the Jensen inequality, which is true as the absolute value is a convex function.
Proving (b) is a bit trickier: suppose to have two functions $f : X \to Y$ and $g : X \to X$, with X finite set. Clearly there exists at least $x^\star \in X$ such that

$$
f(x^\star) = \max_{x \in X} f(x)
$$

and suppose there is only one such value for simplicity. It might be that $x^\star \notin Im(g)$, hence

$$\max_x f\left(g(x)\right) \leq \max_x f(x). \tag{4.5}$$

The inequality (4.5) is analogous to (b): to see the analogy, consider two functions,

$$f : \mathcal{S} \to \mathbb{R}, \quad f(s) = V_{t+n-1}(s)$$

and

$$g : \mathcal{S} \to \mathbb{R}, \quad g(s) = \sum \text{weighted rewards after } s$$

where the weights are given by γ elevated to the corresponding certain power.
Now, define a new function

$$h : \mathcal{S} \to \mathbb{R} \quad h(s) = \left|f(s) - g(s)\right|$$

Inequality (b) can be rewritten as

$$\max_{s\in\mathcal{S}} \mathbb{E}_\pi \left[h(S_{t+n}) \mid S_t = s\right] \leq \max_{s\in\mathcal{S}} \mathbb{E}_\pi \left[h(S_t) \mid S_t = s\right].$$

S_{t+n} is a function of S_t, hence the inequality. Intuitively, given a completely free, unrestricted choice for S_t, the set of possible states S_{t+n} is a subset of the set of possible states S_t. Moreover the probability distributions over those sets will be different: when the choice of S_t is unrestricted, we can assign full probability mass to whichever is the best state, meanwhile even if that best state may still be possible at time $t+n$, it may no longer have full probability mass.
The proof is concluded by showing that

$$\begin{aligned}
\max_s \left|\mathbb{E}_\pi \left[G_{t:t+n} - v_\pi(s) \mid S_t = s\right]\right| \leq & \\
&\leq \gamma^n \max_s \mathbb{E}_\pi \left[\left|V_{t+n-1}\left(S_t\right) - \sum_{k=0}^{T-t-1} \gamma^k R_{t+k-1}\right| \mid S_t = s\right] \\
&\leq \gamma^n \max_s \mathbb{E}_\pi \left[\left|V_{t+n-1}\left(S_t\right) - v_\pi(S_t)\right| \mid S_t = s\right] \\
&\leq \gamma^n \max_s \left|V_{t+n-1}(s) - v_\pi(s)\right|.
\end{aligned}$$

The last inequality is a consequence of the fact that the maximum of a random variable is an upper bound on the expectation of that random variable. □

One of the main drawbacks of the n-step Temporal Difference methods is that there is no general rule to find an optimal value of the parameter n. The following simple example shows how even in simple cases it is not possible to predict in advance the optimal value of n

Example 5. Consider a random walk, as the one seen in Example 1, with k states and assume that, from each state, one can move only towards the immediately close states, with equal probability. The rewards associated to each transition is 0, with the exception of the one that moves from the rightmost state to the adjacent terminal state: in this case the reward is equal to 1.
What we are trying to compute is the value function associated to each node of the graph, which will coincide with the probability that a random walk starting from s ends in the terminal state on the right.
Given the total number of states k, and assuming that each walk starts from the middle point, the correct value function is

$$V(s) = 1 - \frac{distance\ from\ the\ right\ terminal\ state}{k\ +1} = \frac{s}{k+1}$$

assuming that the states go from 1 (the leftmost) to k (the rightmost).
In the case seen in the previous example, we considered 5 states: the probabilities were $[\frac{1}{6}, \dots, \frac{5}{6}]$, for the states starting from the left.
This example is divided into two parts, which correspond to different results that are showed: in the first part it will be showed how, for a fixed number of states, the values of n and α deeply influence the ability of the algorithm to compute the value function, while in the second part the focus will be on the relation between k and n.

For the first part assume that the number of states k is equal to 20. Figure 4.1 shows how for different values of α and n we can get very different values of the RMSE, the error measure defined in 3.2. These results have been obtained using all the the possible couples of the parameters:

- $\alpha \in [0.01, 0.02, 0.04, \dots, 0.18, 0.2, 0.25, 0.3, \dots, 0.95, 1]$
- $n \in [1, 10, 50, 100, 200]$

For each couple of parameters, the RMSE has been obtained as the average over 100 simulations of the error after 10 episodes. Figure 4.1 shows how both parameters influence the quality of the estimate of the value function. In particular, in this case it seems that the smallest error is obtained when considering $\alpha = 0.08$ and $n = 5$.
Clearly, the n-step algorithm take longer to perform and this behavior is more pronounced as the parameter increases.
The analysis can go even further: indeed it can be interesting to repeat the same observations done above for different values of k. For each value of the number of states we might obtain a plot analogous to Figure 4.1. In this case, we fix $\alpha = 0.05$ and then we compare the different values of the error for different pairs (n, k): Figure 4.2 clearly shows how the optimal value of n depends on k . The only exceptions are given by the plots corresponding to low values of k and high values of n, where the error does not increase as n grows. It is

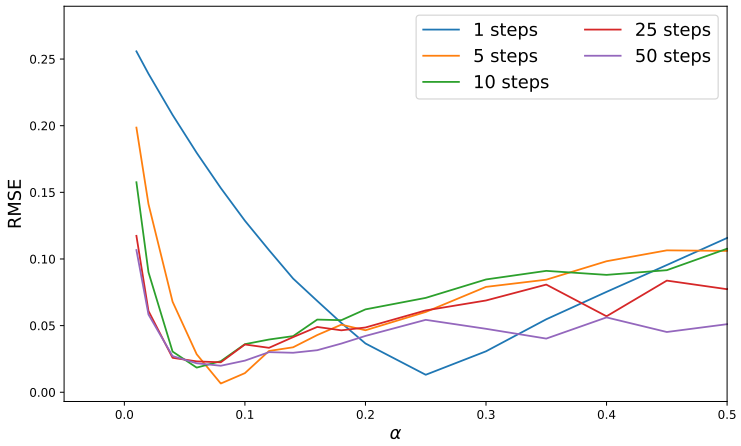


Figure 4.1: Comparison of the RMSE obtained using different values of n and α over 100 simulations. We assumed $\gamma = 1$.

due to the fact that in those examples, after a certain point, the value of n is irrelevant: when considering a graph with just 5 states (the blue line) having an algorithm with 150 or 250 steps does not make any difference, as in both cases we are *de facto* just using a Monte Carlo algorithm, as most episodes will conclude in far less steps.

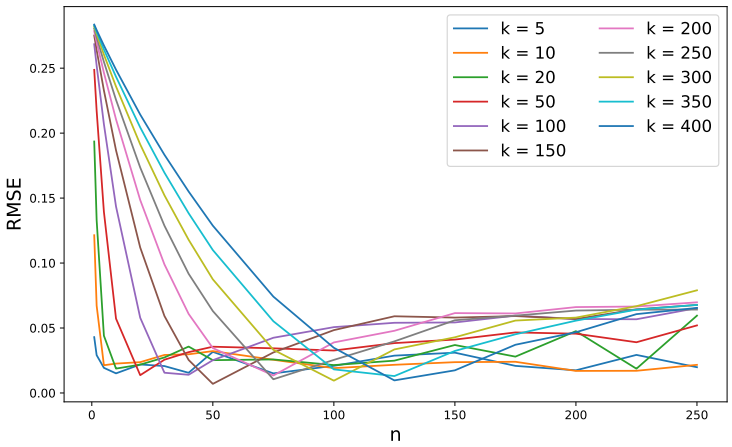


Figure 4.2: Comparison of the RMSE obtained using different values of n and k over 100 simulations. $\gamma = 1$ is all the simulations.

4.2 n-step on policy learning

From eq. (4.3) it is pretty straightforward to obtain a control algorithm.
The first algorithm proposed is the n-step Sarsa: the main idea is to simply consider state-action pairs and add the policy improvement part, for example choosing the constant-ϵ-greedy policy. We can define the n-step returns as follows

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}\left(S_{t+n}, A_{t+n}\right)$$

with $G_{t:t+n} = G_t$ if $t + n \geq T$. The update rule is then given by

$$Q_{t+n}\left(S_t, A_t\right) = Q_{t+n-1}\left(S_t, A_t\right) + \alpha\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right] \quad 0 \leq t < T \tag{4.6}$$

while all the values corresponding to the other couples of states and actions remain unchanged. The algorithm defined by this update rule is called *n-step Sarsa* and the pseudocode is given in Algorithm 14.
The other on-policy algorithm seen in the 1-step case is Expected Sarsa. The only difference between its n-step version and n-step Sarsa is in the computation of the return, which is defined as follows

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \tilde{V}_{t+n-1}\left(S_{t+n}\right) \tag{4.7}$$

where the last term stands for the estimated value of the action value function at time t and can be explicitly computed as

$$\tilde{V}\left(s\right) = \sum_a \pi\left(a \mid s\right) Q_t(s, a) \quad \forall s \in \mathcal{S}.$$

Algorithm 14: n-step TD for estimating $Q \approx q_\star$

input : step size $\alpha \in (0, 1]$, $\epsilon > 0$, $n > 0$ integer
output: value function Q, estimate of $q_\star$

Initialize $Q(s, a) \in \mathbb{R} \forall s \in S, \forall a \in \mathcal{A}$
Initialize *All store and access operations for* S_t, A_t *and* R_t
Initialize *the policy* π*, constant-*ϵ*-greedy with regard to* Q
for *each episode* **do**
    Choose S_0 non-terminal state and store it
    Select and store $A_0 \sim \pi(\cdot \mid S_0)$
    $T \leftarrow \infty$
    for $t = 0, 1, 2 \ldots,$ *until* $\tau = T - 1$ **do**
        if $t < T$ **then**
            Take an action A_t
            Observe and store R_{t+1} and S_{t+1}, respectively the next reward and the next state
            if S_{t+1} *is terminal* **then**
                $T \leftarrow t + 1$
            else
                Select and store $A_{t+1} \sim \pi(\cdot \mid S_{t+1})$
            end if
        end if
        $\tau \leftarrow t - n + 1$
        if $\tau \geq 0$ **then**
            $G \leftarrow \sum_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
            if $\tau + n < T$ **then**
                $G \leftarrow G + \gamma^n Q(S_{t+n}, A_{t+n})$
            end if
            $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
            If π is being learned, update it to reflect changes in Q
        end if
    end for
end for

4.3 n-step off policy learning

Recall how the off policy algorithms use two different policies: one, π, which is the estimate of the value function and another, b, usually more exploratory, to sample the data.
In order to sample from b, has to be taken into account the *importance sampling ratio*, which defines the relation between the relative probabilities of taking a certain action under both policies. When comparing the probability of making the actions from A_t to A_{t+n-1}, it is computed as

$$\rho_{t:t+n} = \prod_{k=t}^{min(t+n,T-1)} \frac{\pi\left(A_k \mid S_k\right)}{b\left(A_k \mid S_k\right)}.$$

The new update rule which considers it, is given by

$$V_{t+n}\left(S_t\right) = V_{t+n-1}\left(S_t\right) + \alpha\rho_{t:t+n-1}\left[G_{t:t+n} - V_{t+n-1}\left(S_t\right)\right].$$

This update generalized the one given in eq. (4.3), thus can be used to define a new algorithm, which is going to be a generalization of the *on-policy n-step Sarsa* seen in Algorithm 14.
We can indeed build an off-policy version, whose update rule is

$$Q_{t+n}\left(S_t, A_t\right) = Q_{t+n-1}\left(S_t, A_t\right) + \alpha\rho_{t+1:t+n}\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right] \quad 0 \leq t < T,$$

as it has to take account of the different policies considered. Note how the importance sampling ratio this time starts from $t+1$, as the first action does not have to be sampled.
The off-policy version of n-step Expected Sarsa uses the same update rule as in off-policy Sarsa, with the usual difference in the way the last element of the return is computed. Moreover, the importance sampling ratio has to be modified into $\rho_{t+1:t+n-1}$, as the last action A_{t+n} is never sampled. In the Expected Sarsa algorithm we consider the expected value from the state S_{t+n}, hence we will consider all possible actions from there.
The update rule for the off-policy algorithm is given by

$$\begin{aligned} Q_{t+n}\left(S_t, A_t\right) = & Q_{t+n-1}\left(S_t, A_t\right) \\ & + \alpha\rho_{t+1:t+n-1}\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right] \quad 0 \leq t < T \end{aligned} \tag{4.8}$$

with

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \tilde{Q}_{t+n-1}\left(S_{t+n}, A_{t+n}\right)$$

and

$$\tilde{Q}_t\left(s\right) = \sum_a \pi\left(a \mid s\right) Q_t(s,a) \quad \forall s \in \mathcal{S}.$$

As stated in the previous chapter, it is possible to see the Q-learning algorithm as a special case of the off-policy version of Expected Sarsa where a greedy policy is considered. Thus, by computing the return as follows,

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \max_a Q_{t+n-1}\left(S_t, a\right)$$

As noted in Sutton and Barto (2018) it might be possible to define more efficient ways to compute the quantities involved in the off-policy algorithms presented up to this point.

Algorithm 15: n-step TD for off-policy estimate $Q \approx q_\star$

input : an arbitrary behavior policy b such that $b\left(s \mid s\right) > 0 \; \forall s \in \mathcal{S}, a \in \mathcal{A}$
input : step size $\alpha \in (0, 1]$,$n > 0$ integer
output: value function Q, estimate of $q_\star$

Initialize $Q(s, a) \in \mathbb{R} \forall s \in S, \forall ac \in \mathcal{A}$
Initialize *All store and access operations for* S_t, A_t *and* R_t
Initialize *the policy* π*, greedy with regard to* Q
for *each episode* **do**
    Choose S_0 non-terminal state and store it
    Select and store $A_0 \sim b(\cdot \mid S_0)$
    $T \leftarrow \infty$
    for $t = 0, 1, 2 \ldots,$ *until* $\tau = T - 1$ **do**
        if $t < T$ **then**
            Take an action A_t
            Observe and store R_{t+1} and S_{t+1}, respectively the next reward and the next state
            if S_{t+1} *is terminal* **then**
                $T \leftarrow t + 1$
            else
                Select and store $A_{t+1} \sim b\left(\cdot \mid S_{t+1}\right)$
            end if
        end if
        $\tau \leftarrow t - n + 1$
        if $\tau \geq 0$ **then**
            $\rho \leftarrow \prod_{i=\tau+1}^{\min\left(\tau+n, T-1\right)} \frac{\pi(A_i mid S_i)}{b\left(A_i \mid S_i\right)}$
            $G \leftarrow \sum_{i=\tau+1}^{min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
            if $\tau + n < T$ **then**
                $G \leftarrow G + \gamma^n Q(S_{t+n}, A_{t+n})$
            end if
            $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho \left[G - Q(S_\tau, A_\tau)\right]$
            If π is being learned, update it to reflect changes in Q
        end if
    end for
end for

4.4 n-step off policy without importance sampling ratio

One might recall the 1-step off policy algorithm, Q-learning and Expected Sarsa, where there was no need for the importance sampling ratio. That happened because all the actions from the future state S_{t+1} were considered when building the last term of the return.
To reproduce this same idea also in the n-step case, one should define an algorithm that, at each step, considers all possible actions and does not sample just one.
To do as described, we can, at each step, select an action from S_t, A_t and then take account also of the expected return of the discarded actions, with a weight proportional to the probability of choosing that action under the target policy π.
Thus, an action at the first step, a contributes to the final value of the return with a weight equal to its probability of being chosen. In the case of the actions not selected, their value will be the one given by the corresponding value if the Q-function, while the selected action A_t contributes with all the action values from the state S_{t+1}. And so on.
In the case with $n = 1$, the return is the same one given by the 1-step Expected Sarsa algorithm:

$$G_{t;t+1} = R_{t+1} + \gamma \sum_a \pi\left(a \mid S_{t+1}\right) Q_t\left(S_{t+1}, a\right) \quad t < T - 1. \tag{4.9}$$

The two step return is given by

$$\begin{aligned} G_{t:t+2} &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi\left(a \mid S_{t+1}\right) Q_{t+1}\left(S_{t+1}, a\right) + \\ &\quad + \gamma \pi\left(A_{t+1} \mid S_{t+1}\right) \left(R_{t+2} + \gamma \sum_a \pi\left(a \mid S_{t+2}\right) Q_{t+1}\left(S_{t+2,a}\right)\right) \\ &= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi\left(a \mid S_{t+1}\right) Q_{t+1}\left(S_{t+1}, a\right) + \\ &\quad + \gamma \pi\left(A_{t+1} \mid S_{t+1}\right) G_{t+1:t+2} \end{aligned} \tag{4.10}$$

for $t < T - 2$.
Equation (4.10) suggests a general recursive formula:

$$\begin{aligned} G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi\left(a \mid S_{t+1}\right) Q_{t+n-1}\left(S_{t+1}, a\right) + \\ + \gamma \pi\left(A_{t+1} \mid S_{t+1}\right) G_{t+1:t+n} \end{aligned} \tag{4.11}$$

for $t < T - 1, n \geq 2$.
The case with $n = 1$ is Expected Sarsa, as noted in (4.9), while $G_{T-1:t+n} = R_T$ when T indicates the terminal time.
From this equation it is possible to define an action-value update rule, similar to the one of the on-policy n-step Sarsa:

$$Q_{t+n}\left(S_t, A_t\right) = Q_{t+n-1}\left(S_t, A_t\right) + \alpha\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right] \tag{4.12}$$

for $0 \leq t < T$ and such that for all the states not involved the value remains unchanged. The pseudocode for this algorithm is given in Algorithm 16. The selection of the action might be given by any function of the state

Algorithm 16: n-step Tree Backup for estimating $Q \approx q_\star$

input : step size $\alpha \in (0, 1]$, $\epsilon > 0$, $n > 0$ integer
output: value function Q, estimate of $q_\star$
Initialize $Q(s, a) \in \mathbb{R} \forall s \in S, \forall ac \in \mathcal{A}$
Initialize *All store and access operations for* S_t, A_t *and* R_t
Initialize *the policy* π*, greedy with regard to* Q
for *each episode* **do**
  Choose S_0 non-terminal state and store it
  Select and store A_0, function of S_0
  $T \leftarrow \infty$
  for $t = 0, 1, 2 \ldots,$ *until* $\tau = T - 1$ **do**
    if $t < T$ **then**
      Take an action A_t
      Observe and store R_{t+1} and S_{t+1}, respectively the next reward and the next state
      if S_{t+1} *is terminal* **then**
        $T \leftarrow t + 1$
      else
        Choose A_{t+1} as function of S_{t+1} and store it
      end if
    end if
    $\tau \leftarrow t - n + 1$
    if $\tau \geq 0$ **then**
      if $t + 1 \geq T$ **then**
        $G \leftarrow R_T$
      else
        $G \leftarrow R_{t+1} + \gamma \sum_a \pi \left(a \mid S_{t+1}\right) Q \left(S_{t+1}, a\right)$
      end if
      for $k = \min(t, T - 1)$ *down through* $\tau + 1$ **do**
        $G \leftarrow R_k + \gamma \sum_{a \neq A_k} \pi \left(a \mid S_k\right) Q \left(S_k, a\right) + \gamma \pi \left(A_k \mid S_k\right) G$
      end for
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, update it to reflect changes in Q
    end if
  end for
end for

4.5 Q(σ) ALGORITHM

This final algorithm is an unification between n-step Sarsa, n-step Expected Sarsa and the Tree Backup algorithm. To understand how and why this algorithm was build, one has to notice the different ways the three previous algorithms build their estimate.
In n-step Sarsa, every step is a simple transition, using a sampled action. In Expected Sarsa, all but the last step follow the same behavior as in Sarsa, while in the remaining we compute the expected value, considering all the actions and their probability of being chosen. Lastly, in the Tree Backup algorithm every step considers the expected value over all the actions.
So, one might notice that in all the algorithms at each step we can decide to sample (hence considering only one action) or computing the expected value, thus considering all the actions. In a nutshell, in Sarsa we always sample, in Expected Sarsa we sample in all but the last step, where we compute the expected value, while in the Tree Backup we compute the expected value at each step. The $Q(\sigma)$ algorithm positions in the middle of this algorithms: through the parameter σ, it is possible to decide, at each step, to sample or to compute the expected value. Moreover, we might even consider a weighted average between these two different ways of computing the return. Indeed, let $\sigma = \{\sigma_t\}_t$, with $\sigma_t \in [0, 1]$. At each step, this value will define the degree of sampling at step t: $\sigma_t = 1$ denotes full sampling, while $\sigma_t = 0$ indicates pure expectation. σ_t might be a function of the state, the action or both. Clearly the sampling might even be done using an external policy, in an *off-policy* fashion: in this case we have to consider the importance sampling ratio as well.
The equations of this algorithm are quite complex, as they have to generalize all the previous algorithms.
At first,we define δ_t^σ, a value which depends on the time step t and on σ_t. It represents the error between the $Q(\sigma)$ estimate at time t of the value function and the value function itself. It is computed as a moving average between the error given by a full sampling (the error that Sarsa would give) and the error obtained using a full expectation (the error of the Tree Backup algorithm):

$$\begin{aligned}
\delta_t^\sigma &= \sigma_{t+1}\delta_t^{\sigma=1} + (1-\sigma_{t+1})\delta_t^{\sigma=0} \\
&= \sigma_{t+1}\left[R_{t+1} + \gamma Q_t\left(S_{t+1}, A_{t+1}\right) - Q_{t-1}\left(S_t, A_t\right)\right] \\
&\quad + (1-\sigma_t)\left[R_{t+1} + \gamma \tilde{V}_t\left(S_{t+1}\right) - Q_{t-1}\left(S_t, A_t\right)\right] \\
&= R_{t+1} + \gamma\left[\sigma_{t+1} Q_t\left(S_{t+1}, A_{t+1}\right) + (1-\sigma_{t+1})\tilde{V}_t\left(S_{t+1}\right)\right] - Q_{t-1}\left(S_t, A_t\right).
\end{aligned} \tag{4.13}$$

Even the importance sampling has to be modified in order to take account of the value of σ, as there is no need to consider it when using the expected values ($\sigma = 0$):

$$\rho_{t+1}^{t+n} = \prod_{k=t+1}^{\tau}\left(\sigma_k \frac{\pi\left(A_k \mid S_k\right)}{\mu\left(A_k \mid S_k\right)} + 1 - \sigma_k\right) \tag{4.14}$$

where $\tau = \min(t + n - 1, T - 1)$ and T is the terminal time step.
Using these quantities, the return is computed as

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\tau} \delta_k^{\sigma} \prod_{i=t+1}^{k} \gamma \left[(1 - \sigma_i)\pi(A_i \mid S_i) + \sigma_i\right] \tag{4.15}$$

Finally, the update rule for this algorithm is given by

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1}^{t+n} \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right] \tag{4.16}$$

with ρ_{t+1}^{t+n} given by eq. (4.14) and $G_{t:t+n}$ given by eq. (4.15).
Algorithm 17 contains the pseudocode corresponding to $Q(\sigma)$.
The proof of convergence in the case with $n = 1$ is given in De Asis et al. (2018) and is based on Lemma 3.2.

Appendix A.3 shows how the returns correspond to the one of n-step Sarsa if $\sigma_t = 1\ \forall t$ and to the one in Tree Backup if $\sigma_t = 0\ \forall t$.
As stated at the beginning of this section, $Q(\sigma)$ is a generalization of all the n-step algorithms presented in this chapter, both in the on-policy and off-policy case. This implies that also almost all the algorithms presented in chapter 2 and chapter 3 are a particular case of $Q(\sigma)$: indeed, to get the Monte Carlo methods it is sufficient to take $n = \infty$, or more practically use a very large number, and select $\sigma = 1$ at every step. To obtain the on-policy methods one will have to select a single policy that is obtained as the constant-ϵ-greedy improvement of the Q function. To get the off policy version, it is sufficient to have the two policies, the behavior one and target one, as usual.
To define the 1-step methods as a special case of $Q(\sigma)$ clearly one has to choose $n = 1$. Then, to obtain Sarsa, one has to set $\sigma_t = 1\ \forall t$ and the constant-ϵ-greedy policy, while the Expected Sarsa estimate is obtained with $\sigma_t = 0\ \forall t$. Q-learning is a special case of off-policy Expected Sarsa, hence can be obtained as well.
The only exception is Double Q-learning: due to its particular structure that features two different Q functions, one should define a new version of $Q(\sigma)$ to find a generalization, as done in Dumke (2017).

Algorithm 17: n-step $Q(\sigma)$ algorithm for estimating $Q \approx q_\star$

input : a behavior policy b and an initial target policy π (for example greedy with regard to Q)
input : step size $\alpha \in (0, 1]$, $\gamma > 0$, $n > 0$ integer
input : a fixed value of σ or a function that computes
output: value function Q, estimate of $q_\star$

Initialize $Q(s, a) \in \mathbb{R},\ \forall s \in S, \forall a \in \mathcal{A}$
Initialize *All store and access operations for* S_t, A_t *and* R_t
for *each episode* **do**
  Choose S_0 non-terminal state and store it
  Select and store $A_0 \sim b\left(\cdot \mid S_0\right)$
  $T \leftarrow \infty$
  Compute σ_0
  for $t = 0, 1, 2 \ldots,$ *until* $\tau = T - 1$ **do**
    if $t < T$ **then**
      Take an action A_t
      Observe and store R_{t+1} and S_{t+1}, respectively the next reward and the next state
      if S_{t+1} *is terminal* **then**
        $T \leftarrow t + 1$ Store $\delta_t^\sigma = R_{t+1} - Q\left((S_t, A_t)\right)$
      else
        Choose A_{t+1} according to $b\left(\cdot \mid S_{t+1}\right)$ and store it
        Compute σ_{t+1} and store it
        Compute $V_{t+1} = \sum_a \pi\left(a \mid S_{t+1}\right) Q\left(S_{t+1}, a\right)$
        Store
        $\delta_t^\sigma = R_{t+1} + \gamma\left[\sigma_{t+1} Q\left(S_{t+1}, A_{t+1}\right) + \left(1 - \sigma_{t+1}\right) V_{t+1}\right] - Q\left((S_t, A_t)\right)$
        Compute and store the importance sampling ratio
        $\rho_{t+1} = \frac{\pi\left(A_{t+1} \mid S_{t+1}\right)}{b\left(A_{t+1} \mid S_{t+1}\right)}$
      end if
    end if
    $\tau \leftarrow t - n + 1$
    if $\tau \geq 0$ **then**
      $\rho \leftarrow 1,\ E \leftarrow 1,\ G \leftarrow Q\left(S_\tau, A_\tau\right)$
      for $k = \tau, \ldots, \min(\tau + n - 1, T - 1)$ **do**
        $G \leftarrow G + E\delta_k^\sigma$
        $E \leftarrow \gamma E\left[\left(1 - \sigma_k\right) \pi\left(A_{k+1} \mid S_{k+1}\right) + \sigma_{k+1}\right]$
        $\rho \leftarrow \rho\left(1 - \sigma_k + \sigma_k \rho_k\right)$
      end for
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
      If π is being learned, update it to reflect changes in Q
    end if
  end for
end for

4.6 Comparison of the performance of the methods

As stated above, the $Q(\sigma)$ is a generalization of all the methods studied above. The following example will contain a comparison between the performances of these algorithms.

Example 6. Consider the Random Walk environment as the one studied in Example 1, with 5 states. In addition to the usual assumption on the value of the reward from the leftmost state, in this example assume that the transition from the left most non-terminal state to the adjacent terminal state is associated to a reward of -1. By doing so, the optimal value function of each state is given by

$$\begin{aligned} V(s) &= \frac{s}{n\_states\ +1} - \left(1 - \frac{s}{n\_states\ +1}\right) \\ &= \frac{2s - n\_states - 1}{n\_states + 1} \quad \forall s \in \{1, \dots, n\_states\} \end{aligned} \tag{4.17}$$

To show the power of the $Q(\sigma)$ method, we compare the results obtained for different values of the parameter. Finally, a dynamic way of computing σ will be presented.
We will assume $n = 3$ and $\alpha = 0.4$, while the different values of σ will be $0, 0.25, 0.5, 0.75$ and 1. Recall that assuming $\sigma = 0$ at each step corresponds to using the Tree Backup algorithm presented in Algorithm 16, while assuming always $\sigma = 1$ gives the same results obtained with the n-step Sarsa algorithm seen in Algorithm 15. Figure 4.3 summarizes the results obtained by averaging the values for 100 simulations.

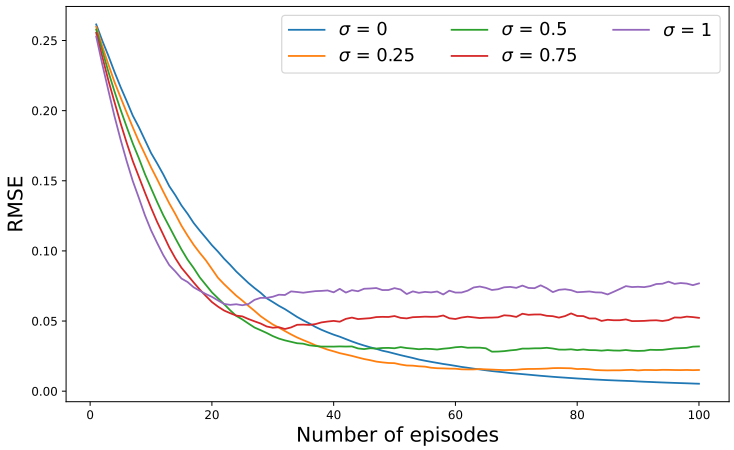


Figure 4.3: Comparison between different values of σ over 100 simulations

Figure 4.3 shows how, in the first episodes, the value of σ which gives the lowest result is 1, however, as the number of episode increases, the algorithms with lower value of the

parameter become more precise. This suggests a way to define a new value of σ dynamically: it can be convenient to define $\sigma = w^t$, with $w = 0.95$, where t indicates the episode considered: by doing so it is possible to have at the same time a fast decrease in the error at the beginning and great performances when the number of episodes considered increases.

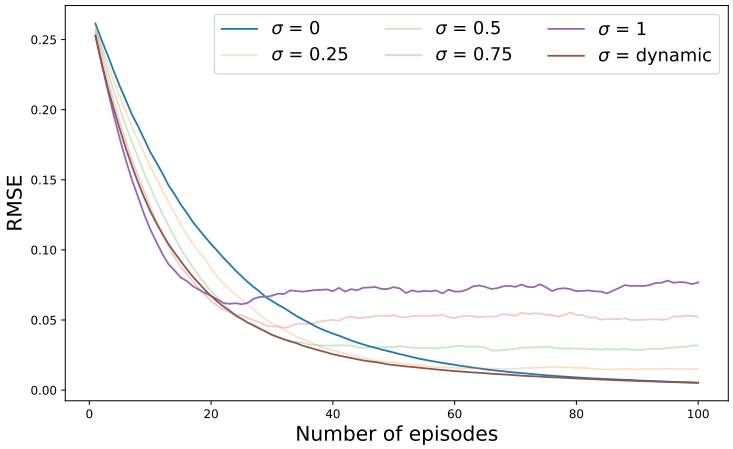


Figure 4.4: Comparison between different values of σ over 100 simulations

The brown line in Figure 4.4 follows the behavior described above, as the initial performances are on par with $Q(1)$, while at the end the results are the same as the one obtained with $Q(0)$.
One might obtain even better results for different dynamic functions that define the parameter σ.

Example 6 shows one little example of the power and flexibility of this method. Unfortunately, it is not possible to define a general optimal function that defines the value of σ. For example, different values of the number of states in the case studied above suggest different values of w. Moreover, different environments not necessarily obtain good results with this function. Further examples are found in De Asis et al. (2018).

5

Different Learning Policies

The most important feature distinguishing reinforcement learning from other types of learning is that it learns from the training information evaluated from the actions taken, rather than instructing by giving correct actions. This creates the need for active exploration, for an explicit search for good behavior, and leads to the exploration-exploitation dilemma: whenever we have to simulate an event, in order to improve our knowledge of the environment, it might be convenient, sometimes, to choose an action whose estimate of the Q function is not the highest, instead of selecting always the one which gave, with the information known at the moment, the best result.
This same problem exists also in real life: if one goes to the same restaurant every time, he would be confident of what will get, but will miss the chances of discovering an even better option. If one tries new places all the time, very likely you he is gonna have to eat unpleasant food from time to time, but might also discover his new favorite restaurant.
In previous chapters we have only considered the constant ϵ-greedy exploration. The focus of this chapter is going to be on more efficient methods to complete this task.
An early survey of exploration methods is found in Thrun (1992).

5.1 GLIE methods

The first class of methods described contains instances of *undirected exploration* methods, that utilize no exploration-specific knowledge and ensure exploration by merging randomness into the choice of the actions. These methods are also part pf the class of GLIE learning policies.

Definition 5.1. GLIE learning policy
A learning policy is defined GLIE (Greedy in the Limit with Infinite Exploration) if it satisfies the following properties:

(1) if a state is visited infinitely often, then each action in that state is chosen infinitely many times

(2) in the limit, as $t \to \infty$, the learning policy is greedy with respect to the learned Q-function with probability 1

The first condition requires that the exploration proceeds indefinitely, while the second requires that its magnitude decays in time.
One example of GLIE learning policy is a slight variation of the constant-ϵ-greedy considered up to this point. This new exploration strategy, called ϵ-greedy, considers an exploration rate that depends on the parameter ϵ and on the state: ϵ will be divided by a factor that increases as the number of visits to that state grows.

Definition 5.2. ϵ-greedy policy
A learning policy is said ϵ-greedy if has the following learning policy:

$$\pi_t\left(a \mid s\right) = \begin{cases} 1 - \epsilon_t & a = \underset{b}{\operatorname{argmax}}\ Q(s, b) \\ \epsilon_t & \text{otherwise} \end{cases}$$

with $\epsilon_t = \epsilon / n_t(s)$, $n_t(s)$ indicating the visits to state s and $0 < c < 1$.

Before proving that the ϵ-greedy learning policy defined above is GLIE, we state a lemma that will be used in the proof:

Lemma 5.1. *Consider a finite Markov Decision Process, an occurrence*

$$\omega = (S_0, A_0, R_1, \ldots, S_t, A_t, R_{t+1}, \ldots)$$

and define the following quantities:

- $n_t(s)$ = *number of visits to state s up to time t*
- $n_t(s, a)$ = *number of occurrences of action a from state s up to time t*
- $t_s(i)$ = *time step of the i-th visit to state s*

Finally, assume that, given a statistics D_t

$$\mathbb{P}\left(A_t = a \mid D_t, A_{t-1}, \ldots, A_0, D_0\right) = \mathbb{P}\left(A_t = a \mid D_t\right)$$

and the learning policy π is such that

$$\{\omega : \lim_{t \to \infty} n_t(s)(\omega) = +\infty\} \subset \{\omega : \sum_{i=0}^{\infty} \mathbb{P}\left(a_{t_s(i)} = a\right)(\omega) = \infty\}$$

Then, $\forall (s, a) \quad n_t(s) \to +\infty a.s.$ and $n_t(s, a) \to +\infty$ a.s.

The proof of this lemma is found in Singh et al. (2000).
We can now prove the following proposition:

Proposition 5.2. *A learning policy defined as in Definition 5.2 is GLIE.*

Proof. Using this policy learning, we pick a random action with probability ϵ_t and the greedy action with probability $1 - \epsilon_t$, with $\epsilon_t = c/n_t(s)$, hence, assuming that ϵ_t is sufficiently large, $\mathbb{P}\left(a \mid s, t_s(i)\right) \geq \frac{\epsilon_t}{m} = \frac{c}{n_t(s)\ m}$, with m number of actions.
Assumption (1) in Lemma 5.1 is always verified in the case studied by defining

$$D_t = \left(s_t, t, n_t(s), Q_t\right).$$

To prove the second one we have to observe that $\sum_{i=0}^{\infty} c/i = \infty$ and consequently

$$\sum_{i=0}^{\infty} \mathbb{P}\left(a \mid s, t_s(i)\right) = +\infty$$

as well. This implies that infinite explorations in state s give a diverging sum of the probabilities and thus also assumption (2) is verified. Lemma 5.1 gives us that we have infinite explorations.
The greedy in the limit part is straightforward, as it is sufficient to notice that, as $\lim_{t\to\infty} n_t(s) = \infty$ we also have $\lim_{t\to\infty} \epsilon_t(s) = 0$ □

The results given by the ϵ-greedy algorithm are strongly influenced by the value of the parameter ϵ_t at each step: given the definition above, the constant c is the one responsible for the performance of the method. An higher value of this parameter will favor exploration and can more convenient at the beginning, while having smaller value exploits the known values of the Q function. On the other hand, a small value exploits predominantly the highest value of the Q function even in the first steps and it can get convenient if we have better initial estimates of the optimal values of the function.

An alternative to ϵ-greedy is obtained considering a learning policy that takes account of all the values if the Q-function and has probabilities of choosing an action proportional to all the Q-values involved: with ϵ-greedy, if two (or even more) action have very similar values, only to the very highest is assigned a significant probability, while all the other action will have a negligible probability of being chosen.
To solve this problem, it can be introduced the Boltzmann exploration method, which uses the *softmax* function to assign probabilities:

Definition 5.3. Boltzmann exploration
Given a value function at time t, Q_t, for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, the probability assigned to an action of being chosen in state s is defined as:

$$\mathbb{P}\left(A_t = a \mid S_t = s, Q_t\right) = \frac{\exp\left(\beta_t(s) Q_t(s,a)\right)}{\sum_{b\in\mathcal{A}} \exp\left(\beta_t(s) Q_t(s,b)\right)}$$

where $\beta_t(s) = \frac{\log n_t(s)}{C_t(s)}$ and $C_t(s) \geq \max_{a,a'} |Q_t(s,a) - Q_t(s,a')|$.

In this case the parameter that influences the exploration is $C_t(s)$: higher values of it, will result in favoring exploration, as it flattens the softmax function.
In the applications, we will define an exploration rate (c) and define C_t as

$$C_t = c \max_{a,a'} \left| Q_t(s,a) - Q_t(s,a') \right|.$$

Using Lemma 5.1, it can be proved the following result:

Proposition 5.3. *The Boltzmann exploration learning policy defined in Definition 5.3 is GLIE.*

Proof. As done in the proof of Proposition 5.2, assumption (1) is obtained assuming $D_t = (S_t, t, n_t(s), Q_t)$.
To satisfy (2) we need that for a certain $c \in (0,1)$

$$\mathbb{P}\left(a \mid s, t_s(i)\right) \geq c/i \tag{5.1}$$

as this result would yield that infinite visits to a state give a divergent sum of the probabilities of having those visits.
(5.1) is equivalent to having that

$$\begin{aligned} \forall a: \quad & \frac{\exp\left(\beta_t(s) Q_t(s,a)\right)}{\sum_b \exp\left(\beta_t(s) Q_t(s,b)\right)} \geq \frac{c}{n_t(s)} \\ & \Rightarrow n_t(s) \exp\left(\beta_t(s) Q_t(s,a)\right) \geq c \sum_b \exp\left(\beta_t(s) Q_t(s,b)\right) \end{aligned}$$

So a stronger condition can be defined by requiring that

$$n_t(s) \exp\left(\beta_t(s) Q_t(s,a)\right) \geq c\ m \exp\left(\beta_t(s) Q_t(s, b_{max})\right) \tag{5.2}$$

where $b_{max} = \operatorname*{argmax}_{b \in \mathcal{A}} Q_t(s,b)$ and m is the number of actions.
It is straightforward to obtain from (5.2) the following inequality

$$\log\left(n_t(s)\right) - \log\left(cm\right) \geq \beta_t(s) \left(Q_t\left(s, b_{max}\right) - Q_t(s,a)\right) \tag{5.3}$$

By setting $c = 1/m$ we get the final condition on $\beta_t(s)$:

$$\beta_t(s) \leq \frac{\log\left(n_t(s)\right)}{\max_a \left|Q_t\left(s, b_{max}\right) - Q_t\left(s,a\right)\right|} = \frac{\log\left(n_t(s)\right)}{C_t(s)} \tag{5.4}$$

where $C_t(s)$ is a finite quantity as all the rewards are finite.
So, if (5.4) is verified, we can use Lemma 5.1 to obtain infinite explorations in each

(state,action) pair.
To prove that the learning policy is also *greedy in the limit*, it is sufficient to take

$$\beta_t(s) = \frac{\log\left(n_t(s)\right)}{C_t}$$

This yields that, as $n_t(s) \to \infty$, we have $\beta_t(s) \to \infty$ and therefore the probability associated to each action, as defined in Definition 5.3, is going to be equal to 1 for the best action and 0 for the others. □

A remarkable result of these class of methods, is that all the proof of convergence in the q-step methods presented in chapter 3 still hold. In particular, observe how the only place in which the policy learning is relevant in the proofs of Theorem 3.1, Theorem 3.3, Theorem 3.4 and Theorem 3.5 is where we need to prove that the respective quantity c_t converge to 0. In all the proofs presented, these value converges to 0 even assuming a GLIE learning policy, of which the ϵ-greedy is a special case.

5.2 Multi-armed bernoullian bandit

Throughout this chapter, to compare different methods of exploration, we will use the Multi-armed Bernoullian Bandit. Presented for a limited number of methods also in Russo et al. (2017), it is a special case of the Multi-armed bandit studied in Sutton and Barto (2018).

Example 7. Suppose to be in an environment with two states and K actions, all going from one state to the terminal one, as seen in Figure 5.1.

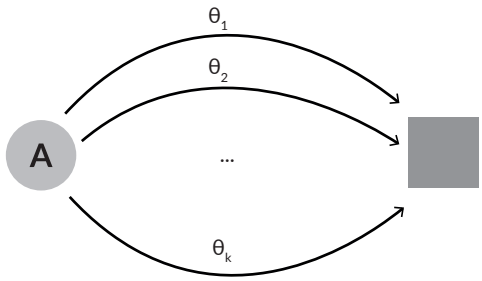
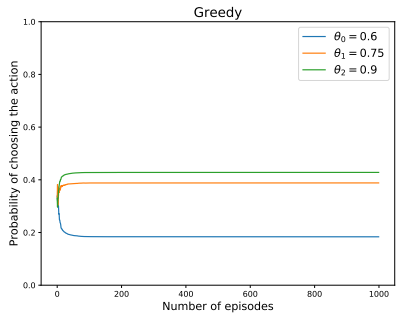


Figure 5.1: Representation of a graph equivalent to the Bernoullian Bandit described in Example 7.

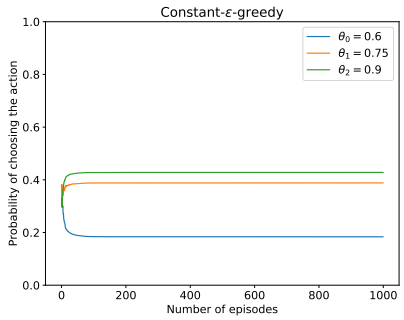
Each action, indicated with k, produces a reward of one with probability θ_k and a reward of 0 with probability $1 - \theta_k$. Each value of θ_k can be interpreted as the probability of success of action k, hence also as the mean reward of each action. Note that the mean rewards θ_i are unknown, but fixed over time.
To update the parameters after each action, we will use the sample average: the estimate is going to be given by the fraction of samples which resulted in a positive reward.
To reduce the number of operations required, we will consider an incremental implementation: given Q_n, the average of the first $n-1$ elements and the n-th reward, R_n, the new average which considers all the n values is given by

$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right] \tag{5.5}$$

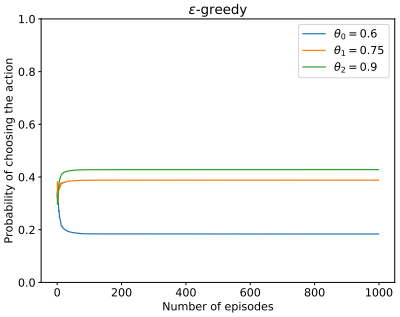
The detailed calculations to get to this formula are analogous to those presented in appendix A.1, in the section dedicated to the Ordinary Importance Sampling and therefore will not be repeated.
Note how the update rule defined in (5.5) is a simple application of all the update rules used in previous chapters, with a learning rate $\alpha_n = \frac{1}{n}$. Due to the particular structure of the simple Markov Decision Process considered, the value of the Q function in the state at time $t+1$ is always equal to 0, as that is a terminal state.
We can now compare the results of the different undirected exploration methods in the environment described: for each method a plot will represent the evolution of the probability of choosing each action as the number of episodes considered increases. The results presented in the plot have been obtained by computing the average of the results of 10000 simulations (all with a different seed): the value of the probability is the fraction of simulations that select the specific action at the time step considered.
Finally, the number of actions was equal to 3, the true parameters were $\theta_0 = 0.6, þ_1 = 0.75$ and $\theta : 2 = 0.9$ and the value of the Q function is initialized to 0.5 for each action.
The first learning method considered is the greedy one: the action selected is always the one with the highest value of the Q function: due to the choice of the parameters θ_i and the fact that the first action is chosen randomly, the results are very poor. One possible reason for this behavior, is that if the first choice is one of the non-optimal actions and the result of the first step is a success, then the value associated to this non-optimal action is going to be the highest and hence it will be chosen also at the following step. Due to the choice of the true parameters (all greater than 0.5), the expected values of all the actions is greater than the initial value and hence a poor initial choice, alongside a sequence of successes for this action, can influence negatively the whole simulation.
A similar, negative, result is also obtained for the constant-ϵ learning rate and for the ϵ-greedy algorithm defined in Definition 5.2. Meanwhile, the Boltzmann exploration learning policy defined in Definition 5.3 is the undirected method that gives the best results and is able to recognize the best action.



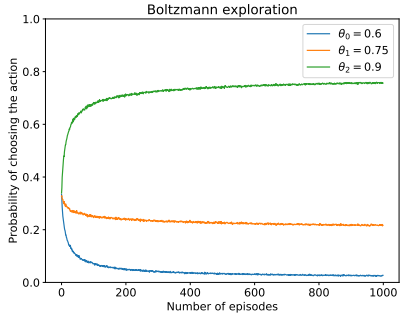
(a) Greedy action selection



(b) Constant-ϵ-greedy action selection



(c) ϵ-greedy action selection



(d) Boltzmann exploration

Figure 5.2: Results of the Bernoullian bandit for the undirected methods after 10000 simulations of 1000 episodes each: In the Boltzmann exploration case we assumed an exploration rate of 1, while for ϵ-greedy and constant-ϵ-greedy we assumed $\epsilon = 0.1$.

A peculiar result is that the constant ϵ exploration method seems to work better than the ϵ greedy one: this is due to the fact that there is only one state visited and hence the exploration rate ϵ_t decreases very fast. It can be showed that by defining $\epsilon_t = c/\sqrt{n_t(s)}$ this negative effect is mitigated and the results get better.

5.3 Upper confidence bound action selection

The methods studied up to this point consider only a punctual estimate for each value of the parameter: in this section, we will briefly introduce an approach that tries to take into consideration the potential of each action to be optimal by defining a confidence interval for each θ_k.
When considering a situation such as the one presented in Example 7, it is possible to

build, for each action k, the following confidence interval:

$$\left[\theta_k - c\sqrt{\frac{\log(t)}{N_t(k)}}, \theta_k + c\sqrt{\frac{\log(t)}{N_t(k)}}\right]$$

where c is an hyperparameter, set by the user, which defines the amount of exploration desired: since it influences the variance (and therefore the amplitude of the interval), higher values of c will give an higher exploration rate throughout all the calculations. $N_t(k)$ indicates the occurrences of action k up to time t.
After defining for each action (and hence each parameter) a confidence interval, the action selected is the one that potentially can obtain the highest value:

$$A_t = \underset{a}{\text{argmax}}\ \theta_k + c\sqrt{\frac{\log(t)}{N_t(k)}}$$

If an action has never been selected before, hence has $N_t(k) = 0$ it is always selected.
Each time k is selected the uncertainty is presumably reduced: $N_t(k)$ increments, and, as it appears in the denominator, the uncertainty term decreases. On the other hand, each time an action other than k is selected, t increases but $N_t(k)$ does not: due to t appearing in the numerator, the uncertainty estimate increases. The use of the natural logarithm means that the increases get smaller over time, but are still unbounded. All actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently and have shown a worse behavior, will be selected with decreasing frequency over time.

The results of this action selection approach in the Bernoullian bandit example are

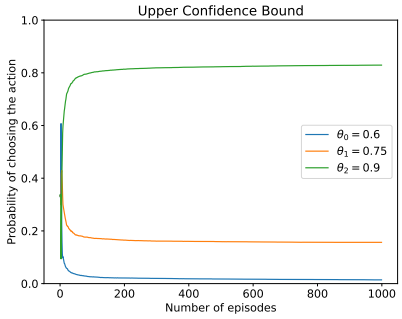


Figure 5.3: Results with UCB action selection after 10000 simulations of 1000 episodes each. The constant c, which indicates the exploration rate, has been assumed equal to 0.1.

represented in Figure 5.3:

This approach converges fast to a suboptimal policy, that chooses with a probability slightly higher than 0.8 the optimal action: this is a better result than the one obtaines

with all the GLIE learning policy presented in the previous sections. It is not excluded that different values of the exploration rate c, initially chosen as $c = 0.1$, give better results.
One difficulty of this exploration approach is in dealing with nonstationary problems and when studying large state spaces, particularly when using function approximation. In these more advanced settings the idea of UCB action selection is usually not practical.

5.4 Thompson sampling

This method introduces the ideas of Bayesian Statistics to solve the exploration-exploitation problem: the uncertainty about a parameter $\theta_i \in \mathbb{R}^n$ (associated to the action i) is represented through a distribution p_i. A generalization of these idea to more general Markov Decision Processes will be the focus of chapter 6. At the beginning, we define a *prior distribution* on the parameter; after applying an action $A_t \in \mathcal{A}$ the agent will observe a reward R_{t+1}, that will influence the *posterior distribution*.
We can define two different action selection approaches: the first one, called *Greedy selection*, consists in choosing the action associated to the highest expected value, with regard to the distribution p_i, of the corresponding parameter. The second, called *Thompson sampling* approach, samples a value of the parameter for each action, from each different distribution p_i: the action selected is the one with the highest sampled value.
Using the Bayes rule, it is then possible to update p according to the reward observed after performing the selected action, thus obtaining the *posterior distribution* of the parameter θ.
The shape of the prior distribution, as well as the way it is updated, is strongly correlated with the problem studied. In the case of the Bernoullian bandit seen in Example 7 the calculations are fairly simple: each parameter θ_k, for $k \in \{1, \dots, K\}$ is unknown, but fixed over time. We choose, as prior distribution, a beta distribution, with parameters α_k and β_k, where the index k indicates the action considered at the moment. For each action, the prior probability density function of θ_k is

$$p(\theta_k) = \frac{\Gamma(\alpha_k + \beta_k)}{\Gamma(\alpha_k)\,\Gamma(\beta_k)} \theta_k^{\alpha_k - 1} (1 - \theta_k)^{\beta_k - 1} \tag{5.6}$$

where $\Gamma(z)$ indicates the Gamma function:

$$\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt$$

By choosing the parameters $\alpha_k = \beta_k = 0.5$ for each action, we get an uniform distribution over each parameter θ_k. this choice is ideal in a situation like the one studied where we have no information on the value of the parameter.
After each observation, the distribution is updated according to the Bayes' rule: it can be

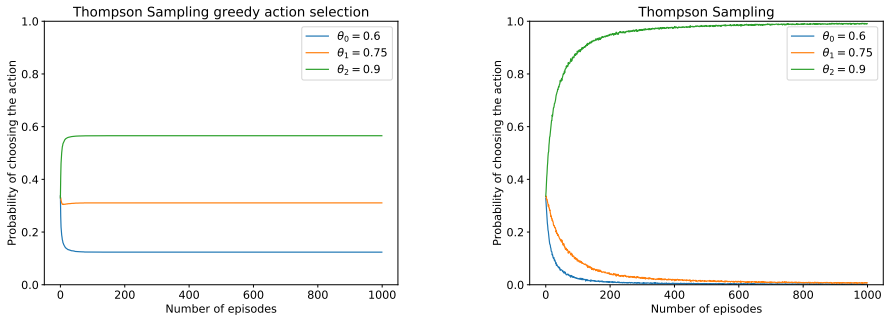
showed that the posterior is still a beta distribution, with the following updated parameters:

$$(\alpha_k, \beta_k) \to \begin{cases} (\alpha_k, \beta_k) & \text{if } A_t \neq k \\ (\alpha_k, \beta_k) + (R_{t+1}, 1 - R_{t+1}) & \text{if } A_t = k \end{cases} \tag{5.7}$$

Note how only the parameters of the selected action are updated.
A distribution as the one defined in (5.6), with parameters (α_k, β_k) has mean

$$\hat{\theta}_k = \frac{\alpha_k}{\alpha_k + \beta_k}.$$

When using the greedy action selection, the action with the largest value of $\hat{\theta}_k$ is always applied. The result, as showed in Subfigure 5.4a simulating 1000 episodes with parameters $\theta_0 = 0.6, \theta_1 = 0.75$ and $\theta_2 = 0.9$, are underwhelming. This is due to the possibility that the algorithm gets stuck on repeatedly applying a non optimal action. To understand the poor performances, consider an example in which the first action selected is A_0: if it gives a reward of 1 (and it has a probability of 0.6 of doing so), the parameters for the posterior distribution become $(\alpha_0, \beta_0) = (1, 0)$ therefore giving an expected value of 1. This means that at the following step, the action selected is going to be still A_0. Since the true value of the parameter θ_0 is 0.6, it is not impossible that the action A_0, which is the worst one among the three possible actions, will be selected throughout all the episodes, giving non optimal results.



(a) Greedy choice (b) Thompson sampling

Figure 5.4: Results of the Bernoullian bandit with the Bayesian approach after 10000 simulations of 1000 episode each. We assumed an uniform prior distribution, initially described by a Beta distribution of parameters $\alpha = 0.5, \beta = 0.5$.

On the other hand, Subfigure 5.4b shows how the Thompson sampling action selection has a fast convergence to the optimal policy, which consists in selecting always the action with the highest value of the parameter θ_k.

6

Bayesian Reinforcement learning

In this chapter we will try to understand how to use some of the exploration techniques defined in chapter 5 when dealing with more complex Markov Decision processes such as the one studied in the previous chapters. To compare the performances of different methods, a couple of examples, inspired by those proposed in Dearden et al. (1998), will be considered: the first one, indicated as *chain*, consists in a graph of 5 states, as the one shown in Figure 6.1.

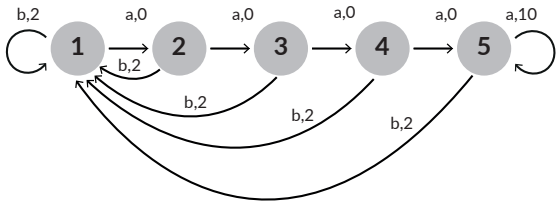


Figure 6.1: Graph representing the *chain* environment studied in this chapter and already seen in Dearden et al. (1998)

At each state, the agent can perform 2 actions (a and b): the former will always bring the agent in the following state with a reward of 0, except in the case in which the starting state is the right-most, as this situation gives a reward of 10. On the other hand, choosing action b always brings to the initial state, with a reward of 2. Moreover, every time an action is chosen, with a probability of 0.2 the the agent moves in the direction opposite

to the one indicated by the action. Every episode starts from S_0: the optimal strategy would be getting to the right-most state as fast as possible and from there perform action a until the maximum number of episodes is reached, thus the optimal policy suggests the action a at each step.. As will be seen in the rest of the chapter, some learning algorithms give a policy which gets stuck with always performing b. Clearly obtaining the optimal result is not likely, due to the probability of moving in the opposite direction, and this influences negatively the results. It can be noticed how deleting this condition induces all the exploration methods to a convergence to the optimal policy.

The second example (*maze*) consists in going from point A to B in the gridworld in Figure 6.2. After each step, a reward of -1 is given to the agent, however, for every flag collected in the path, a reward of +20 is awarded at the end of the episode. Moreover, at each step there is a probability of 0.1 of taking an action and then moving in the perpendicular direction. As in the previous example, the introduction of the possibility of moving differently to what intended creates some issues to most of the algorithm.

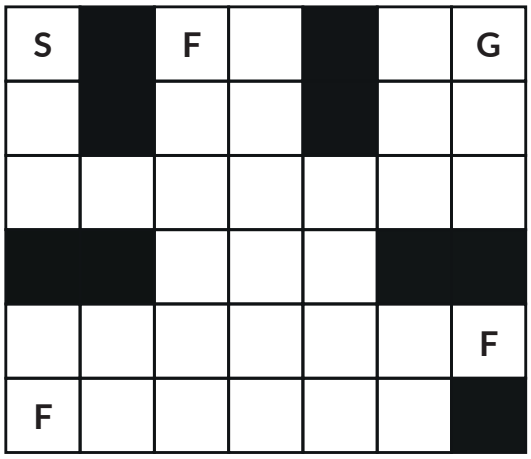


Figure 6.2: Representation of the *maze* environment studied in this chapter and already seen in Dearden et al. (1998)

6.1 Q-learning with undirected learning policies

Using the GLIE learning policies presented in section 5.1 is no different to what has been seen in chapters 2 to 4 when using the *constant-ϵ* exploration: we still define a Q value

function and then use the new exploration methods to choose, at each step, the appropriate action. The phase with the updates of the Q-values remains untouched and the convergence is still guaranteed, as observed before.

6.1.1 Results in the chain example

The following results have been obtained when studying the chain example and represent the average over 10 independent simulations of the results obtained when using the greedy approach, constant ϵ, ϵ-greedy and Boltzmann exploration. The optimal values of the hyperparameters have been obtained as the result of 50 simulations for different values and are indicated in 6.1.

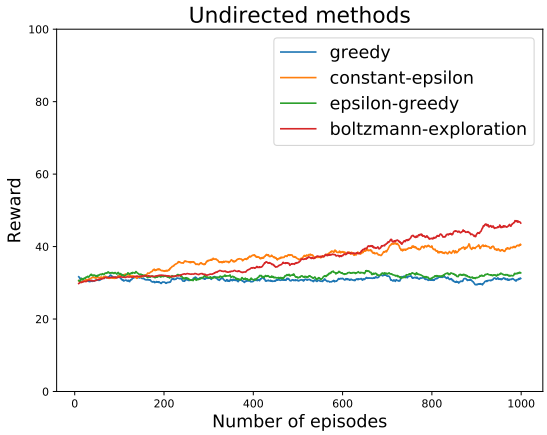


Figure 6.3: Moving average (period = 25) of the rewards obtained by each undirected method as the number of episode increases in the chain example. The hyperparameters used are collected in table 6.1.

This plot shows how there is little to no difference between the results obtained with these methods: in particular, in this case, the Boltzmann exploration approach seems to be the best one.
In the following pages, it will be showed how these results can be improved using the right tools.
To compare the results of the different methods, we will consider, for each simulation, the last 5 episodes: their results will be combined, giving us a more complete view of the mean and the standard deviation of the performance of each method. The results obtained with this naive methods are collected in the following table:

	Hyperparameters	Mean	Std dev
Greedy exploration		160.76	90.76
Constant ϵ	$\epsilon = 0.25$	202.84	64.28
ϵ greedy	$\epsilon = 0.05$	164.52	86.14
Boltzmann exploration	$c = 2.5$	232.68	67.96

Table 6.1: Numerical results obtained using undirected exploration methods in the chain example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

6.1.2 Results in the maze example

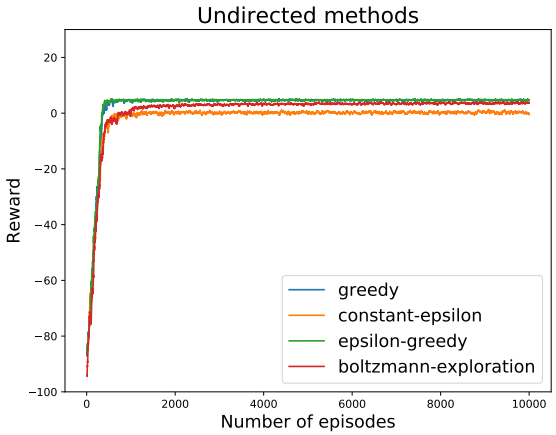


Figure 6.4: Moving average (period = 25) of the rewards obtained by each undirected method as the number of episode increases in the maze example. The hyperparameters used are collected in table 6.2.

Similar observations can be done also for the maze example: the following table collects the results, that will be improved by future methods. It is interesting to notice the fast convergence of all the methods to a sub-optimal strategy.

	Hyperparameters	Mean	Std dev
Greedy exploration		23.7	4.34
Constant ϵ	$\epsilon = 0.2$	-2.9	8.50
ϵ greedy	$\epsilon = 0.25$	23.6	4.98
Boltzmann exploration	$c = 2.25$	17.8	7.52

Table 6.2: Results obtained using undirected exploration methods in the maze example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

The values for this example have been obtained after 10 simulations of 10000 episodes each.

6.2 Bayesian Q-learning

As seen in the trivial examples in chapter 5, introducing a Bayesian framework in the known methods can vastly improve the results, improving the quality of the exploration strategy in the initial stages. This section will contain an extension of the traditional Q-learning method introduced in section 3.5.
First introduced in Dearden et al. (1998), this method considers, for each state-action pair, a probability distribution : the action selection will be then more informed, as it will take into consideration the current estimate of Q, as well as the uncertainty about it.

6.2.1 Q-value distribution

To formalize the idea briefly described above, consider a random variable, $R_{s,a}$, which corresponds to the total discounted reward received when we are in state s, execute action a and follow the optimal policy afterwards. Our goal is to learn the expected value of this random variable, even though at the beginning we are also uncertain about its distribution. To simplify the calculations, we will assume that $R_{s,a}$ is normally distributed: this is also a reasonable assumption due to the central limit theorem and the ergodicity of the Markov Decision processes studied.
This implies that, to model the uncertainty about the distribution of $R_{s,a}$ we can define a normal distribution having, as parameters, the mean $\mu_{s,a}$ and the precision $\tau_{s,a}$. The precision is simply the inverse of the variance ($\tau_{s,a} = 1/\sigma^2_{s,a}$) and has been introduced to simplify the calculations.
A second reasonable assumption is that the prior distribution over $\mu_{s,a}$ and $\tau_{s,a}$ are all independent when considering different state-action pairs. We will furthermore assume that the distribution over the two parameters is a *normal-gamma*: we say that $\mu, \tau \sim NG(\mu_0, \lambda, \alpha, \beta)$ if the probability distribution function is

$$p(\mu, \tau) \propto \tau^{1/2} e^{-\frac{1}{2}\lambda\tau(\mu-\mu_0)^2} \tau^{\alpha-1} e^{-\beta\tau} \tag{6.1}$$

The parameters μ_0, λ, α and β will have to be determined in advance for the initial priori distribution and will heavily influence the exploration rate and hence the final result.
The choice of this distribution is natural in the framework considered, as it is the conjugate prior distribution for a normal distribution which has unknown mean and precision, such as the one used in this case. This implies that the posterior distribution of the parameters is still a normal-gamma, as proved by Theorem 6.1:

Theorem 6.1. *Let $p(\mu, \tau) \sim NG(\mu_0, \lambda, \alpha, \beta)$ be a prior distribution over the unknown parameters for a normally distributed variable R, and let $r_1, \ldots, r_n$ be n independent samples of R with $M_1 = \frac{1}{n}\sum_i r_i$ and $M_2 = \frac{1}{n}\sum_i r_i^2$. Then*

$$p(\mu, \tau \mid r_1, \ldots, r_n) \sim NG\left(\mu_0', \lambda', \alpha', \beta'\right)$$

where:

- $\mu_0' = \frac{\lambda\mu_0 + nM_1}{\lambda + n}$
- $\lambda' = \lambda + n$
- $\alpha' = \alpha + n/2$
- $\beta' = \beta + \frac{1}{2}n\left(M_2 - M_1^2\right) + \frac{n\lambda(M_1 - \mu_0)^2}{2(\lambda + n)}$

Proof. As widely known, the posterior distribution is proportional to the product between the likelihood of the data given the general parameter θ and the prior distribution. In the case considered, with independent samples from a normal distribution of parameters μ and τ, we have the following proportion:

$$p(\mu, \tau \mid r_1, \ldots, r_n) \propto p(data \mid \mu, \tau)\, p(\mu, \tau)$$

The likelihood of the independent is given by

$$p(r_1 \ldots, r_n \mid \mu, \tau) = \prod_{i=1}^{n} \sqrt{\frac{\tau}{2\pi}} e^{-\frac{\tau}{2}(r_i - \mu)^2} = \frac{\tau^{n/2}}{(2\pi)^{n/2}} e^{-\tau/2 \sum_i (r_i - \mu)^2}$$

The following equality will be used:

$$\begin{aligned} \sum_i (r_i - \mu)^2 &= \sum_i \left(r_i^2 - 2r_i\mu + \mu^2\right) \\ &= \sum_i r_i^2 - 2\mu \sum_i r_i + n\mu^2 \\ &= nM_2 - 2\mu n M_1 + n\mu^2 \end{aligned} \tag{6.2}$$

We will now carry on all the calculations necessary to prove the desired result:

$$
\begin{aligned}
&p\left(\mu,\tau \mid r_1,\ldots,r_n\right) \propto \\
&\qquad \propto \tau^{n/2}\exp\left\{-\tau/2\sum_i (r_i-\mu)^2\right\}\tau^{1/2}\exp\left\{-\frac{1}{2}\lambda\tau(\mu-\mu_0)^2\right\}\tau^{\alpha-1}e^{-\beta\tau} \\
&\qquad \overset{(6.2)}{\propto} \tau^{1/2}\tau^{\alpha+n/2-1}\exp\left\{-\frac{\tau}{2}\Big(nM_2-2\mu nM_1+n\mu^2+\mu^2\lambda- \right. \\
&\qquad\qquad \left. -2\mu\lambda\mu_0+\lambda\mu_0^2+2\beta\Big)\right\} \\
&\qquad \propto \tau^{1/2}\tau^{(\alpha+n/2)-1}\exp\left\{\frac{\tau}{2}\left(\mu^2(n+\lambda)-2\mu\left(\frac{nM_1(n+\lambda)}{n+\lambda}+\right.\right.\right. \\
&\qquad\qquad \left.\left.\left.+\frac{\lambda\mu_0(n+\lambda)}{n+\lambda}\right)+nM_2+\lambda\mu_0^2+2\beta\right)\right\} \\
&\qquad \propto \tau^{1/2}\tau^{(\alpha+n/2)-1}\exp\left\{\frac{\tau}{2}\left((n+\lambda)\left(\mu^2-2\mu\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)+\right.\right.\right. \\
&\qquad\quad \left.\left.\left.+\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2\right)-(n+\lambda)\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2+nM_2+\lambda\mu_0^2+2\beta\right)\right\} \\
&\qquad \propto \tau^{1/2}\exp\left\{-\frac{1}{2}\tau(n+\lambda)\left(\mu-\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2\right)\right\}\tau^{(\alpha+n/2)-1} \\
&\qquad\qquad \exp\left\{\frac{\tau}{2}(n+\lambda)\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2-\frac{\tau}{2}\Big(nM_2+\lambda\mu_0^2+2\beta\Big)\right\}
\end{aligned}
$$

This correspond to a normal-gamma distribution, with new parameters $\mu_0', \lambda', \alpha'$ and β', where

- $\mu_0' = \frac{nM_1+\lambda\mu_0}{n+\lambda}$
- $\lambda' = \lambda + n$
- $\alpha' = \alpha + n/2$
- $\beta' = \frac{\tau}{2}(n+\lambda)\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2 - \frac{\tau}{2}\Big(nM_2+\lambda\mu_0^2+2\beta\Big)$

To show that the equation for β' found above is equivalent to the one proposed in the theorem's statement, it is necessary to carry out some additional calculations:

$$
\beta' = \frac{\tau}{2}(n+\lambda)\left(\frac{nM_1+\lambda\mu_0}{n+\lambda}\right)^2 - \frac{\tau}{2}\Big(nM_2+\lambda\mu_0^2+2\beta\Big)
$$

$$
\begin{aligned}
&= \beta - \frac{1}{2}\frac{1}{n+\lambda}\left(nM_1 + \lambda\mu_0\right)^2 + \frac{n}{2}M_2 + \frac{1}{2}\lambda\mu_0^2 \\
&= \beta - \frac{1}{2}\frac{1}{n+\lambda}\left(n^2M_1^2 + 2n\lambda M_1\mu_0 + \lambda^2\mu_0^2\right) + \frac{n}{2}M_2 + \frac{1}{2}\lambda\mu_0^2 \\
&= \beta - \frac{1}{2}\frac{n^2M_1^2}{n+\lambda} - \frac{n\lambda M_1\mu_0}{n+\lambda} - \frac{1}{2}\frac{\lambda^2\mu_0^2}{n+\lambda} + \frac{n}{2}M_2\frac{1}{2}\lambda\mu_0^2 \\
&= \beta + \frac{n}{2}M_2 - \frac{n}{2}M_1^2 + \frac{n}{2}M_1^2 - \frac{n}{2}M_1^2\frac{n}{n+\lambda} - n\lambda\frac{M_1\mu_0}{n+\lambda} + \frac{1}{2}\lambda\mu_0^2 - \frac{1}{2}\lambda\frac{\mu_0^2}{n+\lambda} \\
&= \beta + \frac{n}{2}\left(M_2 - M_1^2\right) + \frac{n}{2}M_1^2\left(1 - \frac{n}{n+\lambda}\right) - n\lambda\frac{M_1\mu_0}{n+\lambda} + \frac{1}{2}\lambda\mu_0^2\frac{n}{n+\lambda} \\
&= \beta + \frac{n}{2}\left(M_2 - M_1^2\right) + \frac{n}{2}\frac{\lambda}{n+\lambda}\left(M_1^2 - 2M_1\mu_0 + \mu_0^2\right) \\
&= \beta + \frac{n}{2}\left(M_2 - M_1^2\right) + \frac{n\lambda\left(M_1 - \mu_0\right)^2}{2(n+\lambda)}
\end{aligned}
$$

□

The final assumption is rarely verified in the Markov decision processes studied, however it is useful to maintain a compact representation for the method. In fact, we also assume that the posterior distributions over the parameters $\mu_{s,a}$ and $\tau_{s,a}$ are independent: in general they might be strongly correlated, however the advantages given in terms of computational simplification justify the choice. Thanks to all the assumptions done, we can represent the uncertainty on each $R_{s,a}$ only through the 4 hyperparameters that define the corresponding normal-gamma distribution. Instead of storing the estimate of the Q-value, we will just store, for each state-action pair the tuple $(\mu_{0,s,a}, \lambda_{s,a}, \alpha_{s,a}, \beta_{s,a})$.

6.2.2 Action selection

As seen in chapter 5, even in the most trivial cases, different action selection approaches can produce very different results. For this method, three different action selection methods will be proposed: *greedy*, *Q-value sampling* and *Myopic VPI selection*.
The greedy approach naively always chooses the action that maximizes the expected value $\mathbb{E}\left[\mu_{s,a}\right]$. It will be showed in Lemma 6.4 that

$$\mathbb{E}\left[\mu_{s,a}\right] = \mathbb{E}\left[R_{s,a}\right] = \mu_{0,s,a}$$

This approach is the extension of the greedy choice seen in section 5.4: in general, the performances obtained using this method will be worse, as it does not take into account the uncertainty about the Q-value.

The second method presented is *Q-value sampling*: the main idea of this approach is

to select actions stochastically, based on the current probability distribution on each action. In particular, we sample a value from each of the probability distribution functions of all the actions at state s and then select the action with the highest value sampled. Using the assumption on the independence of posterior distributions, we can even compute the probability of selecting action a at state s:

$$\begin{aligned}\mathbb{P}\left(a = \underset{a'}{\operatorname{argmax}}\ \mu_{s,a'}\right) &= \mathbb{P}\left(\forall a \neq a', \mu_{s,a} > \mu_{s,a'}\right) \\ &= \int_{-\infty}^{\infty} \mathbb{P}\left(\mu_{s,a} = x\right) \prod_{a' \neq a} \mathbb{P}\left(\mu_{s,a'} < x\right) dx\end{aligned} \tag{6.3}$$

This expression can be evaluated by computing the posterior distribution of the parameter μ, given by Lemma 6.2.

Lemma 6.2. *If $p(\mu, \tau) \sim NG(\mu_0, \lambda, \alpha, \beta)$, then*

$$p(\mu) = \left(\frac{\lambda}{2\pi}\right)^{1/2} \beta^{\alpha} \frac{\Gamma(\alpha + 1/2)}{\Gamma(\alpha)} \left(\beta + \frac{1}{2}\lambda(\mu - \mu_0)^2\right)^{-(\alpha+1/2)}$$

and

$$\mathbb{P}(\mu < x) = T\left((x - \mu_0)\left(\frac{\lambda\alpha}{\beta}\right)^{1/2} : 2\alpha\right)$$

where $T(x : d)$ is the cumulative t-distribution with d degrees of freedom. Moreover, $\mathbb{E}[\mu] = \mu_0$ and $\mathbb{V}ar(\mu) = \frac{\beta}{\lambda(\alpha-1)}$

Proof. We will just prove the shape of the marginal distribution: the remaining observations are simple properties of the distribution obtained.

$$\begin{aligned}p(\mu) &= \int_0^{\infty} p(\mu, \tau) d\tau \\ &= \int_0^{\infty} p(\mu \mid \tau) p(\tau) d\tau \\ &= \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \int_0^{\infty} \tau^{1/2} \exp\left\{\frac{1}{2}\lambda\tau(\mu - \mu_0)^2\right\} \tau^{\alpha-1} \exp\{\beta\tau\} d\tau \\ &= \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \int_0^{\infty} \tau^{\alpha-\frac{1}{2}} \exp\left\{-\left(\beta + \frac{1}{2}\lambda(\mu - \mu_0)^2\right)\right\} d\tau \\ &\overset{(\star)}{=} \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \left(\frac{1}{\left(\beta + \frac{1}{2}\lambda(mu - \mu_0)^2\right)}\right)^{\alpha+1/2} \int_0^{\infty} \xi^{\alpha-1/2} e^{-\xi} d\xi\end{aligned}$$

$$
\begin{aligned}
&= \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \left(\left(\beta + \frac{1}{2}\lambda(mu - \mu_0)^2 \right) \right)^{-\left(\alpha+1/2\right)} \int_0^{\infty} \xi^{(\alpha+1/2)-1} e^{-\xi} d\xi \\
&= \frac{\beta^{\alpha}\sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \left(\left(\beta + \frac{1}{2}\lambda(mu - \mu_0)^2 \right) \right)^{-\left(\alpha+1/2\right)} \Gamma\left(\alpha + 1/2\right) \\
&= \left(\frac{\lambda}{2\pi} \right)^{1/2} \beta^{\alpha} \frac{\Gamma\left(\alpha + 1/2\right)}{\Gamma\left(\alpha\right)} \left(\beta + \frac{1}{2}\lambda\left(\mu - \mu_0\right)^2 \right)^{-\left(\alpha+1/2\right)}
\end{aligned}
$$

The change of variable used in equality $(\star)$ is the following

$$\xi = \left(\beta + \frac{1}{2}\lambda(mu - \mu_0)^2 \right) \tau$$

which maintains the same integration interval. □

Fortunately, in practical applications, it is not necessary to compute explicitly the probability of choosing each action: if we sample, for each action, a value from $p(\mu_{s,a})$ and then execute the action with the highest sampled value, each action is selected with the probability given by (6.3).
One of the drawbacks of this method is that it does not consider the amount by which choosing a certain action might improve the current policy, but rather simply focuses on computing the probability that a certain action is indeed the best one.

The last action selection method presented is the so called *Myopic-VPI selection*: this algorithm tries to balance the expected gains from exploration (in the form of improved policies) against the expected cost of doing potentially suboptimal action.
To compute the expected gains from an exploration, we need to find the amount of gain we would have if we learned $\mu^{\star}_{s,a}$, the true value of a parameter $\mu_{s,a}$. Knowing the true parameter would change our policy in two scenarios: if the new knowledge shows that an action previously considered sub-optimal is actually the best choice (given the current understanding of the other parameters), or if the true value of the currently considered best action indicates that there exist other optimal choices.
Formally, in the first case, indicating a_1 as the best action, the current knowledge tells us that $\mathbb{E}\left[\mu_{s,a_1}\right] \geq \mathbb{E}\left[\mu_{s,a}\right]$ for every action $a \neq a_1$. If we find that $\mu^{\star}_{s,a'} > \mathbb{E}\left[\mu_{s,a_1}\right]$, so that the true value associated to the action a' is the best one, we expect that the agent gains $\mu_{s,a'} - \mathbb{E}\left[\mu_{s,a_1}\right]$ by performing this action.
On the other hand, if a_1 and a_2 are what the current knowledge indicates as the two best actions, but the true value indicates that $\mu^{\star}_{s,a_1} < \mathbb{E}\left[\mu_{s,a_2}\right]$, the agent should perform a_2 instead of a_1: in that case the expected gain is $\mathbb{E}\left[\mu_{s,a_2}\right] - \mu^{\star}_{s,a_1}$.
We can then define the $Gain_{s,a}$ function, which indicates the amount of gain from learning

the true value $\mu^\star_{s,a}$:

$$Gain_{s,a}(\mu^\star_{s,a}) = \begin{cases} \mathbb{E}\left[\mu_{s,a_2}\right] - \mu^\star_{s,a} & \text{if } a = a_1 \text{ and } \mu^\star_{s,a} < \mathbb{E}\left[\mu_{s,a_2}\right] \\ \mu^\star_{s,a} - \mathbb{E}\left[\mu_{s,a_1}\right] & \text{if } a \neq a_1 \text{ and } \mu^\star_{s,a} > \mathbb{E}\left[\mu_{s,a_1}\right] \\ 0 & \textit{otherwise} \end{cases} \tag{6.4}$$

Clearly the true value of the parameter is not known in advance (otherwise we could simply select the best action), so we need to compute the expected gain given our prior beliefs as follows:

$$VPI(s,a) = \int_{-\infty}^{\infty} Gain_{s,a}(x)\mathbb{P}\left(\mu_{s,a} = x\right) dx$$

Proposition 6.3, already seen in Dearden et al. (1998), gives us a simple way to compute this value without evaluating the integral:

Proposition 6.3. *$VPI(s,a)$ is equal to $c + \left(\mathbb{E}\left[\mu_{s,a_2}\right] - \mathbb{E}\left[\mu_{s,a}\right]\right)\mathbb{P}\left(\mu_{s,a} < \mathbb{E}\left[\mu_{s,a_2}\right]\right)$ when $a = a_1$ and it is equal to $c + \left(\mathbb{E}\left[\mu_{s,a}\right] - \mathbb{E}\left[\mu_{s,a_1}\right]\right)\mathbb{P}\left(\mu_{s,a} > \mathbb{E}\left[\mu_{s,a_1}\right]\right)$ when $a \neq a_1$, where*

$$c = \frac{\Gamma\left(\alpha_{s,a} + 1/2\right)\sqrt{\beta_{s,a}}}{\left(\alpha_{s,a} - 1/2\right)\Gamma\left(\alpha_{s,a}\right)\Gamma\left(1/2\right)\sqrt{2\lambda_{s,a}}}\left(1 + \frac{\mathbb{E}\left[\mu_{s,a}\right]^2}{2\alpha_{s,a}}\right)^{-\alpha_{s,a}+1/2}$$

and a_1 and a_2 indicate, respectively, the best and the second best action.

The quantity computed above gives an upper bound on the value of information for exploring the action a.
The second term we have to take into consideration is the expected cost of an exploration, given by the difference between the expected value of the action considered the best (which corresponds to what we would obtain if we followed the best action) and the action we want to explore, i.e. $\mathbb{E}\left[Q(s,a_1)\right] - \mathbb{E}\left[Q(s,a)\right]$.
This suggests that the action to choose is

$$\begin{aligned} a &= \operatorname*{argmax}_{a'} VPI(s,a') - \mathbb{E}\left[Q(s,a_1)\right] + \mathbb{E}\left[Q(s,a')\right] \\ &= \operatorname*{argmax}_{a'} VPI(s,a') + \mathbb{E}\left[Q(s,a')\right] \end{aligned} \tag{6.5}$$

Note that when the agent is confident of the estimate of the Q-value, the constant c goes to zero, leading to a greedy choice.

6.2.3 Updating Q-values

The last problem to solve in order to define the complete algorithm is the update of the Q-values, or the update of the parameters of each distribution. This is made complicated

by the fact that the distributions of the Q-values are distributions over the *total* reward, while we observe only *local* rewards.
To solve this problem, we can execute the action a considered and then, from the state s' obtained, we do the optimal action $a^\star$. We can therefore sum the reward obtained from action a with the estimate for $R_{s',a^\star}$ to find a new estimate of $R_{s,a}$. This is the main idea of the method called *Moment updating*.
From Theorem 6.1, we know that to find the new values of the parameters it is sufficient to compute the first two moments:

- $M_1 = \mathbb{E}\left[r + \gamma R_{s',a^\star}\right] = r + \gamma \mathbb{E}\left[R_{s',a^\star}\right]$
- $M_2 = \mathbb{E}\left[\left(r + \gamma R_{s',a^\star}\right)^2\right] = r^2 + 2\gamma r \mathbb{E}\left[R_{s',a^\star}\right] + \gamma^2 \mathbb{E}\left[R^2_{s',a^\star}\right]$

Lemma 6.4 gives us the value of the moments for $R_{s,a}$, distributed as a normal with unknown mean and precision.

Lemma 6.4. *Let R be a normally distributed variable with unknown mean μ and precision τ, and let $p(\mu,\tau) \sim NG(\mu_0,\lambda,\alpha,\beta)$. Then $\mathbb{E}[R] = \mu_0$ and $\mathbb{E}\left[R^2\right] = \frac{\lambda+1}{\lambda}\frac{\beta}{\alpha-1} + \mu_0^2$*

Proof. Recall that $R \sim \mathcal{N}(\mu,\tau)$ and let $f(r \mid \mu,\tau)$ be its probability distribution function. Moreover, $(\mu,\tau) \sim \mathcal{NG}(\mu_0,\lambda,\alpha,\beta)$ and let $g(\mu,\tau \mid \mu_0,\lambda,\alpha,\beta)$ be its pdf.
Finally, assume that $h_1(\mu \mid \tau,\mu_0,\lambda)$ is the pdf of a normal distribution of parameters μ_0 and $1/(\lambda\tau)$, and $h_2(\tau \mid \alpha,\beta)$ the pdf of a gamma distribution of parameters α and β. It is known that $h_1(\mu \mid \tau,\mu_0,\lambda)\, h_2(\tau \mid \alpha,\beta) = g(\mu,\tau \mid \mu_0,\lambda,\alpha,\beta)$.
Given all the notation introduced above, we can compute the first two moments of R:

$$\begin{aligned}
\mathbb{E}[R] &= \int_{\mathbb{R}} r f(r,\mu,\tau) dr \\
&= \int_{\mathbb{R}^3} r f(r \mid \mu,\tau) g(\mu,\tau \mid \mu_0,\lambda,\alpha,\beta) dr d\mu d\tau \\
&= \int_{\mathbb{R}^2} g(\mu,\tau \mid \mu_0,\lambda,\alpha,\beta) \int_{\mathbb{R}} r f(r \mid \mu,\tau)\, dr\; d\mu d\tau \\
&= \int_{\mathbb{R}^2} g(\mu,\tau \mid \mu_0,\lambda,\alpha,\beta)\mu\; d\mu d\tau \\
&= \int_{\mathbb{R}^2} \mu h_1(\mu \mid \tau,\mu_0,\lambda)\, h_2(\tau \mid \alpha,\beta)\, d\mu d\tau \\
&= \int_{\mathbb{R}} h_2(\tau \mid \alpha,\beta) \int_{\mathbb{R}} \mu h_1(\mu \mid \tau,\mu_0,\lambda)\, d\mu\; d\tau \\
&= \int_{\mathbb{R}} h_2(\tau \mid \alpha,\beta)\, \mu_0 d\tau \\
&= \mu_0
\end{aligned}$$

With a similar procedure we can also compute the second moment:

$$\begin{aligned}
\mathbb{E}\left[R^2\right] &= \int_{\mathbb{R}} r^2 f\left(r, \mu, \tau\right) dr \\
&= \int_{\mathbb{R}^3} r^2 f(r \mid \mu, \tau) g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) dr d\mu d\tau \\
&= \int_{\mathbb{R}^2} g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) \int_{\mathbb{R}} r^2 f(r \mid \mu, \tau) dr d\mu d\tau \\
&= \int_{\mathbb{R}^2} g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) \left(\mu^2 + \frac{1}{\tau}\right) d\mu d\tau \\
&= \int_{\mathbb{R}^2} \mu^2 g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) d\mu d\tau + \int_{\mathbb{R}^2} g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) \frac{1}{\tau} d\mu d\tau
\end{aligned}$$

The first term can be written as:

$$\begin{aligned}
\int_{\mathbb{R}^2} \mu^2 g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) d\mu d\tau &= \int_{\mathbb{R}^2} \mu^2 h_1\left(\mu \mid \tau, \mu_0, \lambda\right) h_2\left(\tau \mid \alpha, \beta\right) d\mu d\tau \\
&= \int_{\mathbb{R}} h_2\left(\tau \mid \alpha, \beta\right) \int_{\mathbb{R}} \mu^2 h_1\left(\mu \mid \tau, \mu_0, \lambda\right) d\mu \; d\tau \\
&= \int_{\mathbb{R}} h_2\left(\tau \mid \alpha, \beta\right) \left(\frac{1}{\lambda\tau} + \mu_0^2\right) \; d\tau \\
&= \int_{\mathbb{R}} \left(\frac{1}{\lambda\tau}\right) h_2\left(\tau \mid \alpha, \beta\right) \; d\tau + \mu_0^2 \int_{\mathbb{R}} h_2\left(\tau \mid \alpha, \beta\right) \; d\tau \\
&= \frac{1}{\lambda} \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) d\tau + \mu_0^2
\end{aligned}$$

Meanwhile, the second one is equal to

$$\begin{aligned}
\int_{\mathbb{R}^2} \frac{1}{\tau} g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) d\mu d\tau &= \int_{\mathbb{R}^2} \frac{1}{\tau} h_1\left(\mu \mid \tau, \mu_0, \lambda\right) h_2\left(\tau \mid \alpha, \beta\right) d\mu d\tau \\
&= \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) \int_{\mathbb{R}} h_1\left(\mu \mid \tau, \mu_0, \lambda\right) d\mu d\tau \\
&= \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) d\tau
\end{aligned}$$

We can finally go back to where we left the main equality:

$$\begin{aligned}
\mathbb{E}\left[R^2\right] &= \int_{\mathbb{R}^2} \mu^2 g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) d\mu d\tau + \int_{\mathbb{R}^2} \frac{1}{\tau} g(\mu, \tau \mid \mu_0, \lambda, \alpha, \beta) d\mu d\tau \\
&= \frac{1}{\lambda} \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) d\tau + \mu_0^2 + \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) d\tau \\
&= \left(1 + \frac{1}{\lambda}\right) \int_{\mathbb{R}} \frac{1}{\tau} h_2\left(\tau \mid \alpha, \beta\right) d\tau + \mu_0^2 \\
&= \left(1 + \frac{1}{\lambda}\right) \int_{\mathbb{R}} \frac{1}{\tau} \frac{\beta^\alpha}{\Gamma(\alpha)} \tau^{\alpha-1} e^{-\beta\tau} d\tau + \mu_0^2
\end{aligned}$$

$$
\begin{aligned}
&= \left(1 + \frac{1}{\lambda}\right) \int_{\mathbb{R}} \frac{\beta}{\alpha - 1} \frac{\beta^{\alpha-1}}{\Gamma(\alpha-1)} \tau^{\alpha-2} e^{-\beta\tau} d\tau + \mu_0^2 \\
&= \left(1 + \frac{1}{\lambda}\right) \frac{\beta}{\alpha - 1} \int_{\mathbb{R}} \frac{\beta^{\alpha-1}}{\Gamma(\alpha-1)} \tau^{\alpha-2} e^{-\beta\tau} d\tau + \mu_0^2 \\
&= \frac{\lambda + 1}{\lambda} \frac{\beta}{\alpha - 1} + \mu_0^2
\end{aligned}
$$

The last equality is verified as the argument of the integral is the probability distribution function of a $Gamma(\alpha - 1, \beta)$. □

The results obtained above help us define closed-form equations to compute the new parameters for a state-action pair: the independent samples considered in Theorem 6.1 are going to be only one, $r + \gamma R_{s',a^\star}$. The update rule for the parameters will then take account of the distribution of $R_{s',a^\star}$.
Simple calculations show that, considering $n = 1$, we obtain the following update rules for the parameters of the normal-gamma distribution:

$$
\begin{aligned}
\mu'_{0,s,a} &= \frac{\lambda_{s,a}\mu_{0,s,a} + n\left(r + \gamma \mathbb{E}\left[R_{s',a^\star}\right]\right)}{\lambda_{s,a} + n} = \frac{\lambda_{s,a}\mu_{0,s,a} + r + \gamma\mu_{0,s',a^\star}}{\lambda_{s,a} + 1} \\
\lambda'_{s,a} &= \lambda_{s,a} + n = \lambda_{s,a} + 1 \\
\alpha'_{s,a} &= \alpha_{s,a} + \frac{n}{2} = \alpha_{s,a} + \frac{1}{2} \\
\beta'_{s,a} &= \beta_{s,a} + \frac{1}{2} n \left(r^2 + 2\gamma r \mathbb{E}\left[R_{s',a^\star}\right] + \gamma^2 \mathbb{E}\left[R^2_{s',a^\star}\right]\right) + \\
&\qquad \frac{n\lambda_{s,a}\left(r + \gamma \mathbb{E}\left[R_{s',a^\star}\right] - \mu_{0,s,a}\right)^2}{2\left(\lambda_{s,a} + n\right)} \\
&= \beta_{s,a} + \frac{1}{2}\left(r^2 + 2\gamma r \mu_{0,s',a^\star} + \gamma^2 \left(\frac{\lambda_{s',a^\star} + 1}{\lambda_{s',a^\star}} \frac{\beta_{s',a^\star}}{\alpha_{s',a^\star} - 1}\right)\right) \\
&\qquad + \frac{\lambda_{s,a}\left(r + \gamma\mu_{0,s',a^\star} - \mu_{0,s,a}\right)^2}{2\left(\lambda_{s,a} + 1\right)}
\end{aligned}
$$

Unfortunately this methods tends to become very confident on the value of $\mu_{0,s,a}$ too quickly, sometimes producing poor results. This is due to the fact that our uncertainty about $R_{s',a^\star}$ influences only the estimate of $\beta_{s,a}$, therefore leading to higher estimates of its variance. Unfortunately, this increases quickly the precision on the mean, possibly leading to a convergence on sub-optimal strategies.
As stated in Dearden et al. (1998), it might be possible to mitigate this effect by introducing an *exponential forgetting* term, reducing the impact of previously seen examples.

It has to be noted that, despite the issue just described, the quality of the estimates of Bayesian Q-learning is, at least in the examples considered, much better than the values computed using the methods described in the previous chapters.

It might be possible to eliminate the fast convergence by introducing alternative methods for updating the parameters, such as the *mixture updating* proposed in Dearden et al. (1998): this method is based on computationally heavy numerical calculations and will not be studied.

6.2.4 Convergence

To show the convergence of this method it is sufficient to prove that $\mu_{s,a}$ converges to the corresponding true Q-value and that its variance converges to 0.
As seen in the proof of the convergence of the traditional Q-learning in Theorem 3.3, one of the necessary condition to prove the convergence is to require infinite explorations, meaning that, with infinite episodes, all the states are visited infinitely many times.
Using Q-value sampling and moment updating this is always true, thus giving us the convergence. In the case of Myopic-VPI action selection the infinite explorations' requirement is not satisfied, however, by substituting the greedy selection based on the sum of VPI and the expected value of Q with a Boltzmann exploration over the same values, we can work around the problem and obtain the convergence of the method.

6.2.5 Results in the chain example

The chain experiment is a first example of the improved performances of the method proposed. The following image contains the average results after 50 independent simulations (all with different seeds to perform random choices). The lines represent the moving average (with period 25) of the results, while the semi-transparent lines are the representation of the actual values. This result have been obtained using what are deemed to be the best parameters for this problem: clearly it shows how the performances of the Q-value sampling action selection are far superior to those of the other methods. In particular, it seems that the Myopic-VPI and the greedy selection converge quickly to a certain sub-optimal policy, while the best method continues to improve throughout the whole experiment.
Unfortunately, the results are influenced by an high variance among different simulations: the following table collects, for each method, the average and the standard deviation of the sum of the rewards of the last 5 episodes.

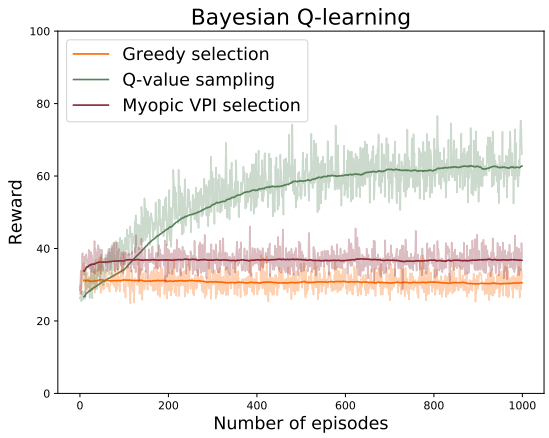


Figure 6.5: Moving average (period = 25) of the rewards obtained using Bayesian Q-learning as the number of episode increases in the chain example. The hyperparameters used are collected in table 6.3.

Action selection method	**Hyperparameters**	**Mean**	**Std dev**
Greedy selection	$\mu_0 = 0, \lambda = 0.001, \alpha = 1.001, \beta = 2$	158.92	91.2
Q-value sampling	$\mu_0 = 0, \lambda = 0.0001, \alpha = 2, \beta = 1.5$	358.4	71.02
Myopic-VPI selection	$\mu_0 = 0, \lambda = 0.01, \alpha = 1.0001, \beta = 2$	186.32	66.03

Table 6.3: Numerical results obtained using Bayesian Q-learning in the chain example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

Comparing these results with those obtained in table 6.1 shows a strong improvement, even though the standard deviation has increased as well.

6.2.6 Results in the maze example

Figure 6.6 contains the results obtained after 10 simulations of 10000 episodes each. even in this case there is a fast convergence towards a non-optimal strategy, as the results indicate that only one flag is collected.

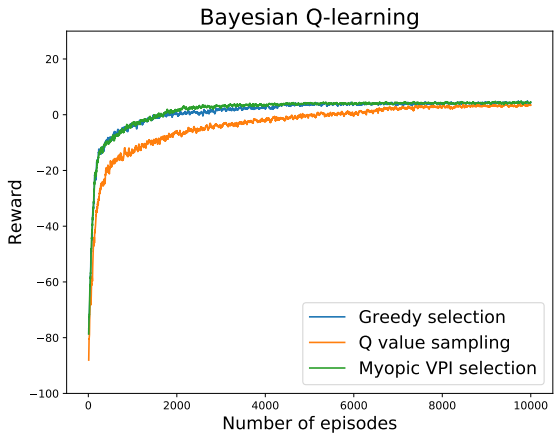


Figure 6.6: Moving average (period = 25) of the rewards obtained using Bayesian Q-learning as the number of episode increases in the maze example. The hyperparameters used are collected in table 6.4.

Differently to what observed in the previous example, in this case the Bayesian method is not much more than the deterministic version: with the parameters found to be optimal among those for which an attempt has been made, only the Q-value sampling action selection method obtained slighty better results than the corresponding deterministic version. The values in table 6.4 are the result of the sum of the numerical outcomes of the last 5 episodes.

Action selection method	**Hyperparameters**	**Mean**	**Std dev**
Greedy selection	$\mu_0 = 0, \lambda = 0.0001, \alpha = 4, \beta = 0.5$	20.6	5.1
Q-value sampling	$\mu_0 = 0, \lambda = 0.25, \alpha = 1.5, \beta = 0.01$	22.5	4.22
Myopic-VPI selection	$\mu_0 = 0, \lambda = 0.0001, \alpha = 4, \beta = 0.75$	25.6	2.97

Table 6.4: Numerical results obtained using Bayesian Q-learning in the maze example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

6.3 Bayesian Expected Sarsa

It might be interesting to define the bayesian version also for the other methods defined in 3: as previously seen, Q-learning is sub-optimal in specific situations, therefore it cannot be excluded that even its bayesian version can be improved.
This part will focus on the definition of an algorithm called *Bayesian Expected Sarsa*, which is a bayesian version of the method studied in section 3.6.

This method will be very similar to Bayesian Q-learning: the only difference is in the update phase: this time we will not consider just the best action, but rather combine the estimates associated to all the actions.

6.3.1 Similarities with Bayesian Q-learning

This method is based on the same assumptions used in section 6.2: we consider a random variable $R_{s,a}$, which indicates the total discounted reward received when we are in state s, execute action a and then follow the optimal policy. We will assume that it is normally distributed, with both the mean $\mu_{s,a}$ and the precision $\tau_{s,a}$ unknown. To carry out the calculations, we will further assume that both the prior and the posterior distribution of $\mu_{s,a}$ and $\tau_{s,a}$ are independent when considering different state-action pairs. Finally, we will assume that the prior distribution on the parameters of the normal distribution is a normal-gamma of parameters $\mu_{0,s,a}, \lambda_{s,a}, \alpha_{s,a}$ and $\beta_{s,a}$. As seen in Theorem 6.1, these assumptions give us a posterior distribution which is still a normal-gamma, with slightly different parameters that will depend on the data observed.

Regarding the action selection algorithms for this method, we might use all the approaches proposed for Bayesian Q-learning. However, for reasons that will become clear in the future pages, the initial focus will just be on the Q-value sampling.
The convergence can be proved in the same way as seen in section 6.2: clearly this time the proof will be based on the one of Theorem 3.4, which is however based on the same principles as Theorem 3.3.

6.3.2 Updating Q-values

As stated above, this part is the only one that differentiates from Bayesian Q-learning: it still remains the problem that the quantities we are trying to estimate are distribution over total rewards, rather than the one we observe which are only local. What we propose to solve it, is to consider all the actions and their associated random variable $R_{s,a}$ and associate to each a weight equal to the probability of selecting such action.
Therefore, the return associated to each sample will be equal to

$$r + \gamma \sum_a \pi\left(a \mid s'\right) R_{s',a} \tag{6.6}$$

where s' indicates the state reached after action a from state s and r the return obtained. Using the *Moment updating* seen in section 6.2, we need to compute the values of M_1 and M_2 to obtain the posterior distribution over $R_{s,a}$.
Using also the results from Lemma 6.4, we obtain the following equalities:

$$M_1 = \mathbb{E}\left[r + \gamma \sum_a \pi\left(a \mid s'\right) R_{s',a}\right]$$

$$= r + \gamma \sum_a \pi\left(a \mid s'\right) \mathbb{E}\left[R_{s',a}\right]$$

$$\begin{aligned}
M_2 &= \mathbb{E}\left[\left(r + \gamma \sum_a \pi\left(a \mid s'\right) R_{s',a}\right)^2\right] \\
&= \mathbb{E}\left[r^2 + 2r\gamma \sum_a \pi\left(a \mid s'\right) + \gamma^2 \left(\pi\left(a \mid s'\right)\right)^2\right] \\
&= r^2 + 2r\gamma \sum_a \pi\left(a \mid s'\right) \mathbb{E}\left[R_{s',a}\right] + \gamma^2 \sum_a \pi\left(a \mid s'\right)^2 \mathbb{E}\left[R^2_{s',a}\right] + \\
&\qquad + 2\gamma^2 \sum_{a_i \neq a_j} \pi\left(a_i \mid s'\right) \pi\left(a_j \mid s'\right) \mathbb{E}\left[R_{s',a_i}\right] \mathbb{E}\left[R_{s',a_j}\right] \\
&= r^2 + 2r\gamma \sum_a \pi\left(a \mid s'\right) \mathbb{E}\left[R_{s',a}\right] + \gamma^2 \sum_a \pi\left(a \mid s'\right)^2 \left(\frac{\lambda_{s',a}+1}{\lambda_{s',a}} \frac{\beta_{s',a}}{\alpha_{s',a}-1} + \mu^2_{0,s',a}\right) + \\
&\qquad + 2\gamma^2 \sum_{a_i \neq a_j} \pi\left(a_i \mid s'\right) \pi\left(a_j \mid s'\right) \mu_{0,s',a_i} \mu_{0,s',a_j}
\end{aligned}$$

The update rule for the parameters can be derived from the one seen in Theorem 6.1, by considering only the return (6.6) and with M_1 and M_2 computed as above.
The only unknown quantity that remains to compute is the probability of choosing a certain action in a state s. This quantity depends on the action selection method: for the Q-value sampling we need to compute the quantity indicated in (6.3). In practical applications, to reduce the execution times, it can be possible to use sampling techniques to approximate this integral.
For each action a, n values of $\mu_{s,a}$ will be sampled from the respective probability distributions. Then, the approximation of the probability associated to a will be equal to the fraction of indexes $i \in \{1, \ldots, n\}$ in which the corresponding sample associated to a is the highest. It can be formally proved that this quantity goes to the real probability as $n \to \infty$. Proposition 6.5 proves it for the case with two actions.

Proposition 6.5. *Let X and Y be two independent random variables, with known distribution f_X and f_Y respectively. Moreover, let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be two groups of n values sampled from f_X and f_Y. For $n \to \infty$, the following equality is true:*

$$\mathbb{P}\left(X > Y\right) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{x_i > y_i}$$

Proof. We can observe that

$$\mathbb{E}\left[\mathbb{1}_{X>Y}\right] = \mathbb{P}\left(X > Y\right)$$

Now consider the sequence of independent and identically distributed random variables $X_1, \ldots, X_n$ with expected value $\mathbb{E}\left[X\right]$, and $Y_1, \ldots, Y_n$, with expected value $\mathbb{E}\left[Y\right]$. We can

therefore define another sequence $\mathbb{1}_{X_i>Y_i}$, for $i \in \{1, \dots, n\}$, such that $\mathbb{E}\left[\mathbb{1}_{X_1>Y_1}\right] = \dots = \mathbb{E}\left[\mathbb{1}_{X_n>Y_n}\right] = \mathbb{E}\left[\mathbb{1}_{X>Y}\right]$.
Using the strong law of large numbers, it is possible to prove that

$$\frac{1}{n}\sum_i \mathbb{1}_{X_i>Y_i} \xrightarrow{a.s.} \mathbb{E}\left[\mathbb{1}_{X>Y}\right] = \mathbb{P}\left(X > Y\right)$$

which concludes the proof. □

Since the distributions of the parameters of all actions are independent, the same can be proved also for the case with k actions, although in that case the calculations might be more complex.
This ideas cannot be repeated for the other action selection methods: in both Greedy and Myopic-VPI selection, we simply select the action based on deterministic values, and therefore it is impossible to retrieve a distribution for those. However, it is possible to introduce an alternative version of the proposed action selection methods which allows us to define the probability of selecting a certain action.
Both methods can be modified introducing the *softmax* function to define the probability of choosing each action based, as seen in the Boltzmann exploration method. For the Greedy action selection, the probability of selecting a will be defined as

$$\begin{aligned}\mathbb{P}\left(a \mid s\right) &= \frac{\exp\left(\frac{\log\left(n_t(s)\right)}{C}\mathbb{E}\left[\mu_{s,a}\right]\right)}{\sum_{b\in\mathcal{A}}\exp\left(\frac{\log\left(n_t(s)\right)}{C}\mathbb{E}\left[\mu_{s,b}\right]\right)} \\ &= \frac{\exp\left(\frac{\log\left(n_t(s)\right)}{C}\mu_{0,s,a}\right)}{\sum_{b\in\mathcal{A}}\exp\left(\frac{\log\left(n_t(s)\right)}{C}\mu_{0,s,b}\right)}\end{aligned} \tag{6.7}$$

In the case of the Myopic-VPI action selection, we will not consider the expected value of the mean but rather the quantity we seek to maximize in the method, (6.5):

$$\begin{aligned}\mathbb{P}\left(a \mid s\right) &= \frac{\exp\left(\frac{\log\left(n_t(s)\right)}{C}\left(\mathbb{E}\left[Q(s,a)\right] + VPI(s,a)\right)\right)}{\sum_{b\in\mathcal{A}}\exp\left(\frac{\log\left(n_t(s)\right)}{C}\left(\mathbb{E}\left[Q(s,b)\right] + VPI(s,b)\right)\right)} \\ &= \frac{\exp\left(\frac{\log\left(n_t(s)\right)}{C}\left(\mu_{0,s,a} + VPI(s,a)\right)\right)}{\sum_{b\in\mathcal{A}}\exp\left(\frac{\log\left(n_t(s)\right)}{C}\left(\mu_{0,s,b} + VPI(s,b)\right)\right)}\end{aligned} \tag{6.8}$$

The choice of the argument of the softmax function has been arbitrary and inspired by the work on the Boltzmann exploration: the quantity $n_t(s)$ is the number of visits to state s up to that point, while C is a constant value, equal to the difference between the highest and the lowest of the values of (6.5) for the state s considered.
Likely a different choice of this last two parameters brings to even better results and further investigations are needed.
Finally, introducing this non-deterministic action choice induces infinite explorations in all states as the number of episodes goes to infinity, proving the convergence of the methods for all the action selection approaches proposed.

6.3.3 Results in the chain example

As showed by Figure 6.7, the Q-value sampling is still the superior action selection choice even in this case. Interestingly, the results of greedy selection get closer to those of the Myopic-VPI selection. Unfortunately, both Greedy and Myopic-VPI action selection seem

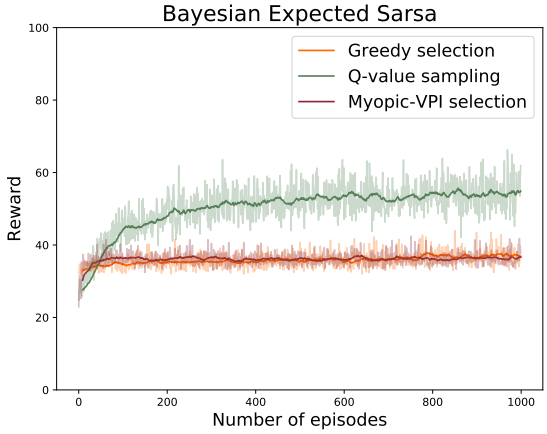


Figure 6.7: Moving average (period = 25) of the rewards obtained using Bayesian Expected Sarsa as the number of episode increases in the chain example. The hyperparameters used are collected in table 6.5.

to converge to a result very quickly, as seen in section 6.2.5.
The results of the final 5 episodes for each method, averaged over 50 independent simulations, are showed in table 6.5:

Action selection method	Hyperparameters	Mean	Std dev
Greedy selection	$\mu_0 = 0, \lambda = 0.001, \alpha = 2, \beta = 0.25$	177.88	36.13
Q-value sampling	$\mu_0 = 0, \lambda = 0.0001, \alpha = 1.1, \beta = 0.1$	291.68	65.6
Myopic-VPI selection	$\mu_0 = 0, \lambda = 0.001, \alpha = 1.0001, \beta = 0.25$	195.08	42.78

Table 6.5: Numerical results obtained using Bayesian Expected Sarsa in the chain example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

The results are particularly interesting for what concerns the greedy and Myopic-VPI selection, as both improve the result obtained with Bayesian Q-learning. In particular, not only the means are higher, but more interestingly, both values of the standard deviation are far inferior.

6.3.4 Results in the maze example

Using these methods, the results are much worse than those found with the Bayesian Q-Learning. It is unclear whether this is due to the smaller amount of unique groups of parameters for which an attempt has been made, or the algorithm is inferior in situations as the one studied.

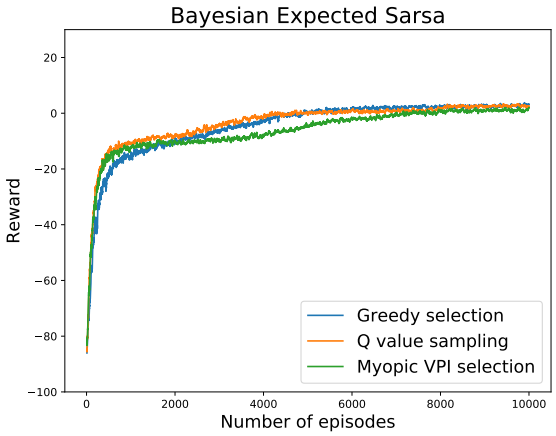


Figure 6.8: Moving average (period = 25) of the rewards obtained using Bayesian Expected Sarsa as the number of episode increases in the maze example. The hyperparameters used are collected in table 6.6.

As seen in table 6.6, Q-value sampling is still the better action selection algorithm in

this case. Moreover, it seems that the methods converge slowly to the final strategy. The numerical results are summarized in table 6.6.

Action selection method	**Hyperparameters**	**Mean**	**Std dev**
Greedy selection	$\mu_0 = 0, \lambda = 0.25, \alpha = 1.0001, \beta = 0.25$	14.7	7.43
Q-value sampling	$\mu_0 = 0, \lambda = 1, \alpha = 5, \beta = 0.01$	21.8	5.17
Myopic-VPI selection	$\mu_0 = 0, \lambda = 0.25, \alpha = 1.25, \beta = 1$	14.8	10.62

Table 6.6: Numerical results obtained using Bayesian Expected Sarsa in the maze example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

6.4 Results with traditional n-step methods

Up to this point, we have seen how the Bayesian version of Q-learning and Expected Sarsa can improve, at least on some of the examples considered, the performances of the traditional 1-step methods seen in chapter 3. This section will show how they compare to the generally better n-step methods seen in chapter 4, in particular to $Q(\sigma)$. That method was able to generalize almost all the methods seen in the chapter by selecting specific parameters.
To find the optimal parameters, all the tuples formed by the following parameters were tried:

- $\sigma \in \{0, 0.25, 0.5, 0.75, 1\}$
- exploration strategy $\in$ {greedy, constant-ϵ-greedy, ϵ-greedy, Boltzmann-exploration}
- $n \in \{1, 5, 10, 20, 50\}$

$\epsilon-$greedy was always performed using the parameter $\epsilon = 0.1$, while for the Boltzmann exploration the parameter chosen was $C = 1.5$. Finally, for each tuple, the result indicated have been obtained with 10 simulations.
Figure 6.9 shows the results of some of the tuples of paraeters that give the best results. The first observation done is that all having $\sigma = 1$ usually gives far superior results, indicating that sampling an action might be the better choice for the problem studied. The blue line is the representation of the best result obtained with $\sigma = 0$ and it is clearly inferior to the others. The orange line represents the best result obtained with $\sigma = 0.75$, while the other 3 are the best results for $\sigma = 1$ for each of the three different undirected exploration strategies.

What Figure 6.9 does not completely show is the high standard deviation of the results, probably due to the large quantity of sampling needed. The following table collects, for each of the 5 tuples of parameters shown in Figure 6.9, the mean and the standard deviation of the results obtained averaging over 10 simulations the values of the last 5 episodes.

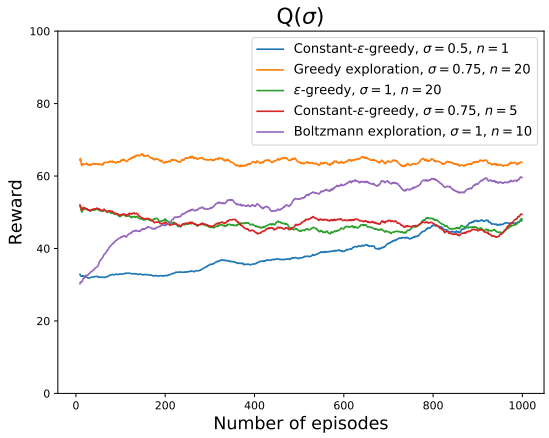


Figure 6.9: Moving average (period = 50) of the rewards obtained using $Q(\sigma)$ as the number of episode increases in the chain example. The hyperparameters used are collected in table 6.7, alongside the precise numerical results.

Action selection method	**Hyperparameters**	**Mean**	**Std dev**
constant-ϵ-greedy	$\sigma = 0.5, \epsilon = 0.1, n = 1$	245.75	80.67
Greedy	$\sigma = 0.75, n = 20$	312.32	74.01
ϵ-greedy	$\sigma = 1, \epsilon = 0.1, n = 10$	247.56	75.53
constant-ϵ-greedy	$\sigma = 0.25, \epsilon = 0.1, n = 5$	253.48	79.61
Boltzmann exploration	$\sigma = 0.25, C = 1.5, n = 5$	297.28	87.41

Table 6.7: Numerical results obtained using $Q(\sigma)$ in the chain example, after 50 simulations of 1000 episodes each. The first row contains the best result obtained with $n = 1$. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

We can repeat the same experiment also in the maze example: the results are collected in table 6.8 and show superior performances compared to the 1-step methods.

Action selection method	Hyperparameters	Mean	Std dev
constant-ϵ-greedy	$\sigma = 1, \epsilon = 0.1, n = 1$	25.2	3.42
Greedy	$\sigma = 1, n = 5$	26.0	23.85
ϵ-greedy	$\sigma = 1, \epsilon = 0.1, n = 50$	25.4	3.66
constant-ϵ-greedy	$\sigma = 0.25, \epsilon = 0.1, n = 5$	28.9	4.34
Boltzmann exploration	$\sigma = 0.25, C = 1.5, n = 5$	32.3	1.49

Table 6.8: Numerical results obtained using $Q(\sigma)$ in the maze example, after 10 simulations of 10000 episodes each. The first row contains the best result obtained with $n = 1$. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

It can be noticed that in the chain example, the best results are obtained with high values of σ, while the opposite is verified in the maze example.
As for all the algorithms seen above, it is not excluded that different values of the parameters (in particular $\epsilon = 0.1$ and $C = 1.5$, imposed to reduce computation times on all the methods) might bring to improved results.
Barring the problem of the high standard deviation, it is clear how, in he chain example, the performances are worse than the results that can be obtained using the Bayesian approach with the right parameters and action selection methods. On the other hand, in the Maze example, the n-step methods seems to be better.
Despite the possible improvements that are still possible, the results are better than the deterministic $1-$step version.

6.5 n-STEP BAYESIAN METHODS

The results on the two examples have showed how these methods can improve the performances even in simple Markov Decision processes, with rewards having higher mean values (and also much higher standard deviation). However, what has been showed in the last section raises a question: what would happen if we were able to develop a multi step version of the Bayesian algorithms presented in sections 6.2 and 6.3?
This question does not have a simple answer, and we have not found in the literature n-step extensions of the model-free methods presented in this chapter.
Intuitively, one might try to follow the ideas explored in chapter 4 and compute the improved estimates only after the simulation has gone n steps forward. This method, however, raises the problem that the action following the one considered are not chosen following the optimal estimate, but rather one of the action-selection algorithm considered. As defined in sections 6.2 and 6.3, the quantity $R_{s,a}$ is the total discounted reward received when we are in state s, execute action a and then follow the optimal policy. Clearly, when considering the Q-value sampling or the Myopic-VPI action selection the first n actions after a would be selected according to a non-optimal strategy.

This situation resembles what has been seen, for instance, in section 4.3 and the introduction of the importance sampling ratio might solve the problem. One might even try to implement a bayesian version of Algorithm 16, as that algorithm considers all the actions at every step and therefore might solve this issue. Unfortunately, both this possible solutions to the problem involve computing the probability of selecting a certain actio and this operation, in the bayesian case, can be computationally intensive (when possible). Clearly, one might use a probability defined by an external function, such as the one considered in section 6.3.2, but further investigations are needed to understand whether this approach can lead to good results.
On the other hand, when considering the greedy action selection, these issues disappear and the results can improve considering an higher quantity of steps.

6.5.1 n-step Bayesian Q-learning with greedy action selection

he only n-step version where an improvement has been observed is the one where the action selection method is the greedy one. As noted above, the issue with the other action selection methods is that they do not choose the optimal action, therefore, when considering an n-step return we would not follow the definition of $R_{s,a}$. With the greedy improvement, at each step we select the action associated to the highest value of the mean return ($\mu_{0,s,a}$).
The n-step version of Bayesian Q-learning differs from the one presented in section 6.2 only in the computations of the return: instead of going forward of one step and then use the estimate of Q given by the best action at state S_{t+1}, we first consider the actions and the rewards from S_t up to S_{t+n-1} (weighted by the appropriate the discount factor) and then estimate the last return using the estimate of the best action at time S_{t+n} .
As in the traditional model-free version of the algorithms, for the first $n-1$ steps no changes will be made and all the values will have to be stored. At the same time, an equal number of operations will be made at the end of the episodes, before starting the next episode.
To compute the update rule for the hyperparameters of the distribution, it is sufficient to compute the values of $M1$ and $M2$, the first two moments associated to the return. In this case the final return can be written as

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots \gamma^{n-1} R_{t+n-1} + \gamma^n R_{S_{t+n},a^\star} \tag{6.9}$$

with the last term being the usual random variable, indicating the return expected from S_{t+n} if we followed the optimal strategy. The equation showed in eq. (6.9) is clearly a generalization of what has been seen in the case with $n = 1$.
to compute the value of the moments, we will use the properties showed in Lemma 6.4.
The first moment is computed as

$$M1 = \mathbb{E}\left[G_{t:t+n}\right]$$

$$
\begin{aligned}
&= \mathbb{E}\left[\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i} + \gamma^n R_{S_{t+n},a^\star}\right] \\
&= \sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i} + \gamma^n \mathbb{E}\left[R_{S_{t+n},a^\star}\right] \\
&= \sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i} + \gamma^n \mu_{0,S_{t+n},a^\star}
\end{aligned}
$$

The computations for the second moment are the following.

$$
\begin{aligned}
M2 &= \mathbb{E}\left[G_{t:t+n}^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i} + \gamma^n R_{S_{t+n},a^\star}\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right)^2 + 2\gamma^n \left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right) R_{S_{t+n},a^\star} + \gamma^{2n} R_{S_{t+n},a^\star}^2\right] \\
&= \left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right)^2 + 2\gamma^n \left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right) \mathbb{E}\left[R_{S_{t+n},a^\star}\right] + \gamma^{2n} \mathbb{E}\left[R_{S_{t+n},a^\star}^2\right] \\
&= \left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right)^2 + 2\gamma^n \left(\sum_{i=1}^{n-1} \gamma^{i-1} R_{t+i}\right) \mu_{0,S_{t+n},a^\star} + \\
&\qquad + \gamma^{2n} \left(\frac{\lambda_{S_{t+n},a^\star} + 1}{\lambda_{S_{t+n},a^\star}} \frac{\beta_{S_{t+n},a^\star}}{\alpha_{S_{t+n},a^\star} - 1} + \mu_{0,S_{t+n},a^\star}^2\right)
\end{aligned}
$$

As showed in table 6.9, it is possible to see how increasing the value of the parameter n improves the performances in the *chain* example also considered for the other methods.

Value of n	**Hyperparameters**	**Mean**	**Std dev**
$n = 1$	$\mu_0 = 0, \lambda = 1, \alpha = 2, \beta = 1.5$	158.92	91.2
$n = 2$	$\mu_0 = 0, \lambda = 0.01, \alpha = 1.5, \beta = 0.25$	160.52	70.26
$n = 3$	$\mu_0 = 0, \lambda = 0.01, \alpha = 1.0001, \beta = 0.1$	160.76	76.75
$n = 5$	$\mu_0 = 0, \lambda = 0.0001, \alpha = 2, \beta = 3$	156.88	61.56
$n = 10$	$\mu_0 = 0, \lambda = 0.25, \alpha = 1.25, \beta = 0.1$	164.64	69.63
$n = 20$	$\mu_0 = 0, \lambda = 0.1, \alpha = 1.001, \beta = 2$	178.36	62.51

Table 6.9: Numerical results obtained using different values of n in the greedy n-step algorithm, in the chain example. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

Even though the performances for higher values of n are better, they are still much lower than those observed for alternative action selection strategies. It is not excluded that the performances might improve in alternative environments and further research are needed.
It might even be possible to define a n step version of Bayesian Expected Sarsa.

6.6 Final considerations on bayesian methods

This section introduced a new way of coping with the exploration-expolitation dilemma, by using a distribution instead of a single value to estimate every value of the Q-function. This methods have proved very effective in the examples showed, however they come with some computational drawbacks. The first one regards the pure computational power required to handle this method: due to the action selection methods involved, the time needed to obtain the results in all the tables in this chapter were higher in the case of the Bayesian methods. One additional drawback is the amount of space used to store the table: in the bayesian methods, for every state-action pair, we need to save the 4 hyperparameters μ_0, λ, α and β. This implieas that the space needed is 4 times higher than the one used in the traditional methods. This is clearly not an issue for the examples considered in this chapter, but might cause issues with more complex problems.
Another known problem with this approach is the one related to the fast convergence to a possibly suboptimal strategy using the moment update method; as showed in Dearden et al. (1998), this can be solved introducing an alternative update method, called *mixture updating*, which, however, requires even more computational power and therefore has not been considered here. Future works should focus on finding appropriate methods to reduce the amount of calculations needed to find the optimal strategy.

The examples proposed in this chapter, as well as the results in 7.3.1, show better performances of the Bayesian algorithms in an environment with limited amount of states. It could be interesting to investigate further this observation.
One last remark has to be done on the Bayesian Expected Sarsa method: the results obtained with Q-value sampling action selection are influenced by the number of samples done at every step to compute the approximation of the probability of choosing each action. In the examples above,we always considered $k = 100$ samples: clearly decreasing the value of the parameter k should reduce the execution time, but further investigation are needed to understand if the results obtained can still be considered valid. At the same time, increasing k probably leads to more precise results, but the execution time would increase even more.

7

Introduction to further topics

This chapter will be divided into two parts: the first section will contain a brief analysis of an alternative way of studying tabular problems, while the second section will discuss the main ideas behind the Approximate Methods for Reinforcement Learning. Finally, some examples of the results obtained by the research community using Reinforcement Learning will be showed.

7.1 Model Based approach

All the methods described in the previous follow the *model-free* approach, which means that we tried to learn near optimal policies without explicitly estimating the dynamics of the environment but rather just focusing on the estimating Q by treating each step in the environment as a sample from the underlying dynamics.
As showed in Chapter 8 in Sutton and Barto (2018), there exists an alternative approach, called *model-based*. The methods that fall into this group try to find the expected values of the state-action pairs by computing an estimate of the environment's dynamics. In particular we try to estimate the state-transition probability defined in Definition 1.1, and the *reward model*. They represent, respectively, the probability of reaching state s' after executing action a at state s and the probability of receiving reward r after executing a in s. In general, we will assume that these approximations are correct, we use them to compute the values of the Q-function and then use it to choose the actions to perform.
These methods are capable of exploiting even more the experience available, therefore can be very useful when the amount of experience is limited and it is difficult (or expensive) to obtain even more. On the other hand, model-based model are much more complex than model-free ones and might be affected by the design of the model.

One of the first attempt of using this method is the Dyna algorithm presented in Sutton (1990).

7.1.1 Dyna

This method uses the real experience both to improve the model (in order to make it match the original environment) and to improve the value function and the policy, as in the methods presented in previous chapters.

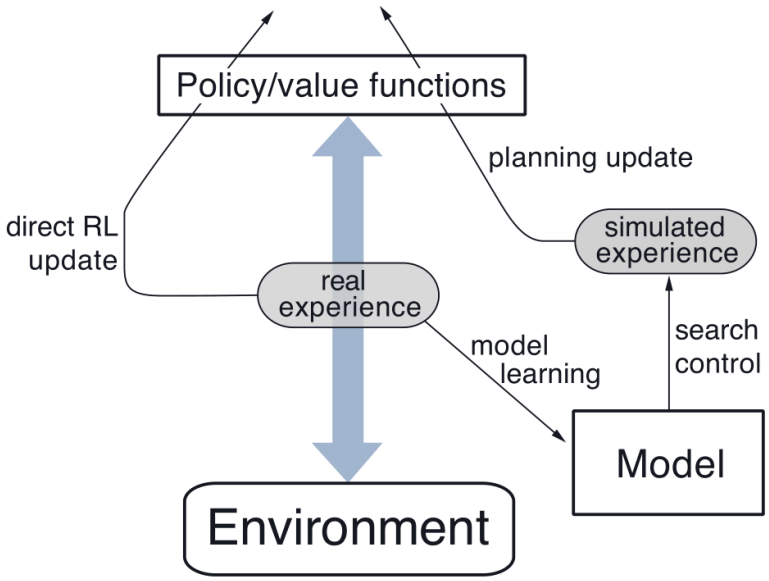


Figure 7.1: General Dyna architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value function in the same way as does simulated exoerience generated by the model environment. Image taken from Sutton and Barto (2018)

As seen in Figure 7.1, this method extends the operations done in model-free methods (indicated as *direct RL update*). The model-learning method assumes a deterministic environment and is still based on a table. After each observed transition, $S_t, A_t \to R_{t+1}, S_{t+1}$ the model records that the prediction R_{t+1}, S_{t+1} will deterministically follow after S_t, A_t. Thus, if we ask to the model a state-action pair that has been previously experienced, it returns the last prediction for next state and next reward that was observed. During the planning phase, the direct RL algorithm samples only from pairs that have been previously experienced and gives rise to the *simulated experience*, which is then used by a reinforcement learning method (typically the same as in the direct RL update) to update the values of the Q function.

Assuming that we use a single-step Q-learning as the direct RL method, the pseudocode of the Dyna algorithm is showed in Algorithm 18.

Algorithm 18: Tabular Dyna-Q

input: The parameter n

1 **Initialize** $Q(s,a) \in \mathbb{R} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

2 **Initialize** $Model(s,a) \in \mathbb{R} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$

3 **for** *each episode* **do**

4 $S \leftarrow$ current non-terminal state

5 $A \leftarrow \epsilon - greedy$ action, according to S and Q

6 Take action A, observe the reward R and next state S'

7 $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A) \right]$

8 $Model(S,A) \leftarrow R, S'$

9 **for** n *times* **do**

10 $S \leftarrow$random previously observed state

11 $A \leftarrow$ random action previously taken in $\mathcal{S}$

12 $R, S' \leftarrow Model(S,A)$

13 $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A) \right]$

14 **end for**

15 **end for**

In Algorithm 18, the quantity $Model(s,a)$ represents the predicted next state and reward for the state-action pair (s,a). The traditional 1-step Q-learning algorithm is achieved considering only up to line 7.

An alternative method which follows this approach is *Prioritized Sweeping*: first described in Moore and Atkeson (1993), it tries to estimate to what extent states would change their value as a consequence of new knowledge of the MDP dynamics. states are then assigned priorities according to the expected size of changes in their values and states to which we assign an higher priority are those for which we perform value propagation.

7.1.2 Bayesian Model based approach

In the previous methods, we kept only a *point-estimate* of the dynamics of the environment. Following the approach seen in sections 6.2 and 6.3, it is possible to extend the methods described introducing a Bayesian approach to model-based reinforcement learning. This approach is explored in Dearden et al. (2013). Another notable work that applies the ideas of Bayesian statistics to model-based reinforcement learning is Strens (2000).

7.2 Approximate methods for reinforcement learning

The problems studied in previous chapters are all characterized by a limited number of states and actions: Reinforcement Learning, however, can be extended to much more complex problems.
These can be still seen as Markov Decision Processes with a very high number of states: for example the game of *backgammon* can be seen as an MDP with 10^{20} states, while chess has approximately 10^{40} states (as stated in Steinerberger (2015)). Moreover some of the problems consider a continuous state space, such as most of those related to robotics.
If we wanted to solve these problems with a very high number of states using the methods described upto this point we would have to compute, for each state an estimate of the Q-function and at the same time store the table. Certain problems, such as playing the real time strategy video game Starcraft II (as seen in Vinyals et al. (2019)) have an estimated number of states that goes well beyond the estimated number of atoms in the universe, hence it would be impossible to store all the values of the Q-function in any existing storage device.
To solve this issue we need to compute an approximation of the Q-function that we will represent as a parameterized functional, with a vector of weights $\boldsymbol{w} \in \mathbb{R}^d$: the approximate state-value function will be $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$, while the action-value function will be approximated by $\hat{q}(s, a, \boldsymbol{w})$. Typically, the dimension of $\boldsymbol{w}$ is going to be much smaller than the number of states, hence updating one weight will change the estimated value of a large number of states (or state-action pairs). This can make the update phase more complex, as the changes to a parameter that improve the estimate for a state can worsen the estimate for other states.

In the tabular case studied in previous chapters the update of the Q-function for a state s was obtained as the result of a shift of the estimate towards an *update target* u. In this case, moreover, each update was done solely on the state s involved. In the approximate framework, where the update of a single state s influences other values and the update target can be computed with more complex methods, we make use of supervised learning techniques to estimate the value function. In particular, we use *function approximation* techniques such as artificial neural networks, decision trees, linear approximators and multivariate regressors. The inputs are going to be pairs *(state, update target)* and we will choose the actual updates as training examples: the approximate function produced will be then seen as the estimated value function. Training data will be provided using the simulations of events on the environment.
To compute the update target and the actual update, several approaches are available: one example is the *Stochastic-gradient method*, studied in Sutton and Barto (2018). In that case, the target output u might be computed using one of the methods seen in previous

chapter. The update at state S_t, assuming an update target U_t, will be then given by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}\left(S_t, \boldsymbol{w}_t\right)\right] \nabla \hat{v}\left(S_t, \boldsymbol{w}_t\right) \tag{7.1}$$

As stated above, the high number of states, alongside the dimension of $\boldsymbol{w}$, implies that each slight change in the weight vector influences an higher number of states. One possible solution to this issue is defining a distribution μ on the states, that, intuitively, describes how much we care about each state s and therefore how important it is to have a good estimate on that state. Usually, $\mu(s)$ is chosen to be the fraction of time spent in s.
This idea leads to a new definition of the objective function to minimize:

$$\overline{VE}(\boldsymbol{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}\left(s, \boldsymbol{w}\right)\right]^2 \tag{7.2}$$

An analogous function will have to be defined for the action value function.
The square root of (7.2) will give an estimate of how much the approximate values differ from the true values.
Our goal will be to find the optimal weight vector $\boldsymbol{w}^\star$, the one for which $\overline{VE}(\boldsymbol{w}^\star) \leq \overline{VE}(\boldsymbol{w})$ for all possible $\boldsymbol{w}$.
Clearly, due to the complexity of the problems involved, it will not always be possible to find the global optimum and we may only look for a *local optimum*. This is usually the best result that can be obtained, in particular when dealing with non-linear function approximators.
An introductory study. that includes some of the several approximation methods developed by the research community, is found in Sutton and Barto (2018).

Bayesian methods similar to those studied in chapter 6 are also used in the approximate framework, as they provide a mechanism to include prior knowledge into the algorithms as well as an elegant (and effective) approach to the exploration-exploitation dilemma. Moreover, these methods have proved to be very effective in handling parameter uncertainty. Clearly, as in the tabular case, these methods require more computational power and the choice of the initial representation of the initial knowledge might not be trivial. A survey of some of the bayesian methods developed in recent years, is found in Ghavamzadeh et al. (2016).

7.3 Applications of reinforcement learning

Reinforcement learning is still a very active field of research and recently many progresses have been done in several fields, that include autonomous driving (see Kiran et al. (2021) for a recent survey), healtcare (see Yu et al. (2019) for a recent survey of the methods) or

robotics(see Kober et al. (2013)). Other fields in which Reinforcement Learning algorithms proved to be very effective include complex video games, as in the already mentioned Vinyals et al. (2019) where a Deep Reinforcement Learning algorithm was able to defeat the best players in the world, finance and news recommendation.
The section is concluded with some insights about some of the most remarkable results obtained using Reinforcement Learning, as well as an introduction to a Python environment often used as a benchmark for Reinforcement Learning algorithms.

7.3.1 Gym

Introduced in 2016 with the goal of defining a groups of environments to be used as a benchmark for Reinforcement Learning algorithms, OpenAI Gym is a toolkit for research in the field.
The environments vary from easy examples such as the one seen in previous chapter (as the Blackjack environment studied in section 2.7) or classic control problems, such as the mountain car one that will be studied, in a simplified version, in section 7.3.1.
This toolkit contains also more complex environments, such as those related to robotics, which can be used for example, for training a robotic hand to perform certain tasks only, or the famous Atari games. It is, in fact, possible to train an agent to play to some of the most famous video games of the past: in that case the input is a representation of the screen, preprocessed by a Convolutional Neural Network, and a well-trained agent might be capable of playing at the same level as the best human players, or even better as recently seen in Ecoffet et al. (2021).
See Brockman et al. (2016) for more details.

Example: mountain car

One of the environments available with the Gym package is *Mountain Car*. In this problem, a car has to learn how to climb a hill as the one in Figure 7.2. At every time step, the agent can perform three different actions: accelerate on the right, accelerate on the left ar do nothing. The state is determined by the position and the speed.

In order to reduce the number of states, and therefore allow the use of the tabular methods studied in previous chapters, the environment has been simplified. In particular, the values of the position (between -1.2 and 0.6) have been rounded down to the second decimal number and an analogous operation have been done on the value of the speed, a decimal value having an absolute value smaller or equal than 0.15. The value of the speed can be either positive or negative, with the former indicating a movement towards the right. With the simplifications aforementioned, the total number of states is 5611, given by 181 possible positions and 31 possible values of the velocity.
Every time an action is executed, the value of the speed increases according to a formula that takes into consideration the force applied by the engine due the acceleration caused

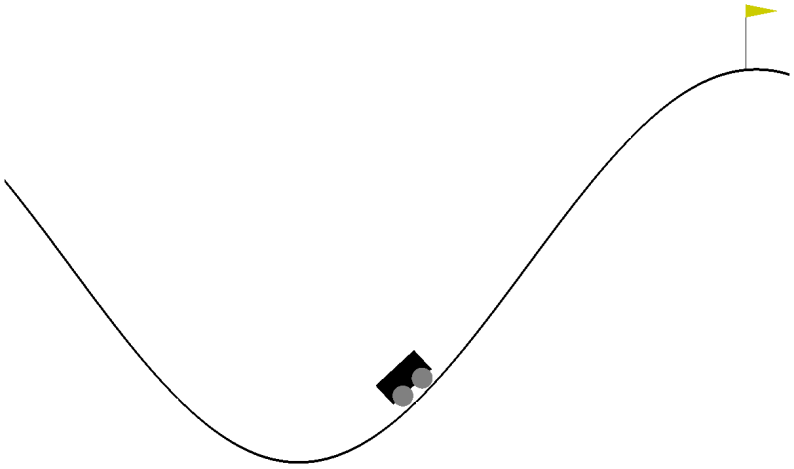
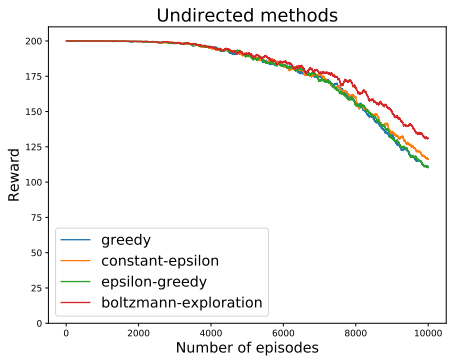


Figure 7.2: Representation of the environment Mountain Car seen in Gym.

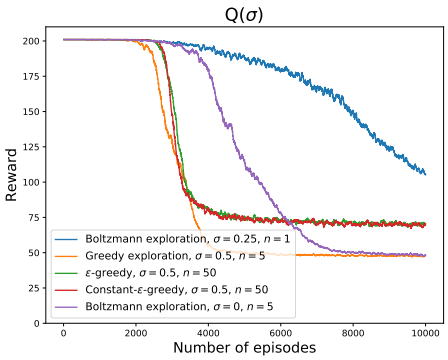
by the action selected, the force of gravity and the slope of the hill. In particular, the updated value of the speed is

$$v_{t+1} = v_t + A_t * F_e - \cos(3x) * F_g$$

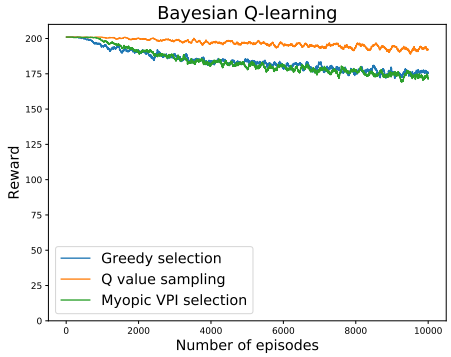
with $A_t = -1, 0, 1$, indicating, respectively, an acceleration on the left, doing nothing and an acceleration on the right. Moreover, F_e is the force given by the engine, $F_e = 0.0075$, F_g indicates the force of gravity, $F_g = 0.025$ and x is the position. The component with the cosine function is obtained given by the shape of the hill.
To each action, a reward of -1 is applied, with the exception of those steps which result in a position across the goal line, positioned at $x = 0.5$.
Finally, every episode starts from a random position between -0.6 and -0.4, so around the lowest point.



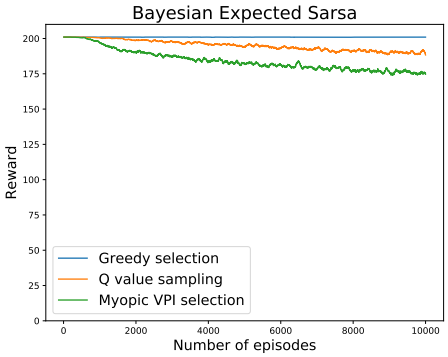
(a) Undirected methods



(b) $Q(\sigma)$



(c) Bayesian Q-Learning



(d) Bayesian Expected Sarsa

Figure 7.3: Graphical comparison of the results obtained in the Mountain Car example using the different methods seen in the entire work.

Table 7.1 contains, for each method, the results obtained using what are deemed to be the optimal parameters. Each result will contain the sum of the values obtained in the last 5 episodes and is composed by the average and the standard deviation of the corresponding results of all the simulations. The optimal parameters have been found after attempting 10 simulations of 10000 episodes (of less than 200 steps), each for every unique group of hyperparameters. Likely, the *real* optimal parameters have not been found.
What showed in table 7.1 shows a different behavior to those observed in the previous examples: the Bayesian methods seems to struggle with this environment and it remains to understand whether that is due to the non-optimal parameter choice or to the properties of the method, and in particular the poor convergence results described in section 6.6. On the other hand, the $Q(\sigma)$ algorithm proves once more an optimal choice.

	Action selection	Parameters	Mean	Std dev
Undirected methods	Greedy selection	-	531.8	59.17
	Constant-ϵ-greedy	$\epsilon = 0.1$	611.0	94.38
	ϵ-greedy	$\epsilon = 0.25$	561.9	41.98
	Boltzmann exploration	$C = 1.5$	626.0	43.73
Q(σ)	Boltzmann exploration	$\sigma = 0.25,\ C = 1.5, n = 1$	537.4	62.48
	Greedy selection	$\sigma = 0.5,\ n = 5$	231.0	8.5
	ϵ-greedy	$\sigma = 0.5,\ \epsilon = 0.1,\ n = 50$	354.2	40.12
	Constant-ϵ-greedy	$\sigma = 0.5,\ \epsilon = 0.1,\ n = 50$	359.9	45.03
	Boltzmann exploration	$\sigma = 0,\ C = 1.5,\ n = 5$	233.0	10.19
Bayesian Q-learning	Greedy selection	$\mu_0 = 0,\ \lambda = 0.25,\ \alpha = 1.01,\ \beta = 2$	781.2	21.56
	Q-value sampling	$\mu_0 = 0,\ \lambda = 0.25,\ \alpha = 2,\ \beta = 0.05$	896.4	75.59
	Myopic-VPI selection	$\mu_0 = 0,\ \lambda = 0.0001,\ \alpha = 2,\ \beta = 1$	768.4	57.18
Bayesian Exp. Sarsa	Greedy selection	$\mu_0 = 0,\ \lambda = 0.25,\ \alpha = 1.01,\ \beta = 2$	1005	0
	Q-value sampling	$\mu_0 = 0,\ \lambda = 0.1,\ \alpha = 1.5,\ \beta = 0.1$	899.0	66.9
	Myopic-VPI selection	$\mu_0 = 0,\ \lambda = 0.01,\ \alpha = 1.5,\ \beta = 0.25$	769.2	60.02

Table 7.1: Results obtained in the Mountain Car example using all the methods studied. The values refer to the number of steps taken to get to the goal line: in this case, lower values indicate a better result. The results have been obtained assuming $\gamma = 0.99$ and (for the methods that required it) $\alpha = 0.1$. The values indicated in the *Mean* and *Std dev* columns have been obtained by summing the results of the last 5 episodes.

7.3.2 Chess

The study of applications of computer chess is one of the oldest in computer science and dates back to the first half of the 20^{th} century. An important result in this field is the one obtained by Deep Blue, which in 1997 defeated the human world chess champion using an algorithm that evaluated the positions of the pieces on the board using handcrafted

features and carefully thought weights, constructed with the help of great human players. In the following years, the research in this field has rapidly progressed and nowadays the most modern artificial intelligence algorithm, AlphaZero (Silver et al., 2018), reaches ability levels much higher than those of the best humans.

One important feature of this algorithm, is that it is based on a more general-purpose idea, and builds its knowledge of the game by repeatedly playing against itself. At the beginning it only knows the rules of the game, it uses a deep neural network (with parameters $\boldsymbol{\theta}$) that receives as input (s) the position of the pieces and gives, as output, the probability of selecting each move (a vector $\boldsymbol{p} = (p_a)_{a \in \mathcal{A}} = \mathbb{P}(a \mid s)$) and a scalar value v that stands for the expected outcome, with defeat equal to -1, draw to 0 and 1 that stands for a victory. AlphaZero learns the moves probabilities and the value estimate entirely by playing against itself: these values are then used to guide the search in future games.

Each search consists of a series of simulated games against itself that explores a tree from root (represented by state s) until a leaf is reached. In each simulation, in each state s we select an action a having low visit count, high move probability and high value (computed as the average over the leaf states), with values computed according to the neural network's weights $\boldsymbol{\theta}$. Each search returns a vector π, representing a probability distribution over actions, $\pi_a = \mathbb{P}(a \mid s)$.

Each game is played by running the search described above (*MCTS search*) from the current position s_t at time t and then selecting an action $a_t \sim \pi_{s_t}$, either proportionally, in the initial exploratory phase, or greedily for exploitation, with regard to the visit counts at root state.

At the end of each game, the terminal position, s_T, induces a reward (-1, 0 or 1, depending on the outcome of the game). Then the parameters of the neural network are updated to minimize the error between the predicted outcome (v_t) and the actual outcome and to maximize the similarity between the policy vector $\boldsymbol{p}_t$ and the search probabilities π_t. The updated parameters are then used in the subsequent games.

AlphaZero uses a convolutional neural network to encode the board and the position of the pieces on it.

One last interesting result regarding AlphaZero, is that it requires minimal modification in order to make it suitable for playing other complex board games such as Go and Shogi. The remarkable results obtained by this algorithm, with a comparison in matches against another strong AI algorithm that plays chess (Stockfish), are found in Silver et al. (2018).

8

Conclusions

Reinforcement Learning methods try to find an optimal strategy in order to solve a problem: the basic idea behind all the methods in this field is to interact repeatedly with the environment and learn, through trial and error, which are the best actions to do in certain situations. The mathematical framework used to describe the problems studied is the Markov Decision process: as seen in chapter 1, we can use states to represent a particular configuration of the environment and action to describe the interactions of the agent with it. Finally, to each interaction will be associated a reward, the quantity we seek to maximize throughout the whole process.
In this work, we only considered simple environments for which, to find the optimal strategy, we can define a function, called *action-value function*, or *q-function*, formally defined as the expected value of the accumulated return after executing action a in state s. The optimal strategy will then be the one that chooses, at each step, the action associated to the maximum expected return.
Clearly we do not know in advance the exact value of the q-function and appropriate estimates are required. To compute those values, several methods have been introduced and are the subjects of chapters 2 to 4: intuitively, they differentiate for the number of steps (and therefore exact rewards) we consider when computing an update of the estimate of the q-function associated to a specific state-action pair. For every method, the update rule is

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[G_t - Q(S_t, A_t) \right]$$

where G_t is the return observed with the interaction between the environment and the agent, α (*learning rate*) and represents the weight we associate to the return when computing the updated estimate and S_t, A_t are, respectively, the state and the action we want

to study at time t.
All the methods described will differ for the way G_t is computed and can be divided into three main classes: Monte Carlo methods, 1-step Temporal Difference and n-step Temporal Difference.
The first class, described in chapter 2, computes the return G_t as the (weighted) sum of all the returns obtained up to the end of the episode; these methods are characterized by a low variance, but are rarely used in practical applications, as reaching the final step before updating the estimate can take a long time in complex environments, or might even be impossible. The second and the third approach are, as their names suggest, closely related, with the latter being a generalization of the other: the idea is that we do not consider all the returns, but rather just a limited number (respectively 1 and a generic n). Considering only a a fixed amount of steps would result in a truncated reward, therefore, to complete the estimate, we add a term that depends on the estimates of the q-function in the state S_{t+n}. Different methods can be used to compute this last estimate: among the methods considered, there are Q-learning, which takes account only of the best action (the one with highest estimate), Sarsa, which samples an action according to the action selection method considered, and Expected Sarsa, which computes a weighted average of all the estimates, with weights given by the probability of choosing the respective action. In the n-step case, we also presented the $Q(\sigma)$ algorithm, a powerful method capable of generalizing almost all the methods previously introduced. In particular, using this algorithm, it is possible to choose, at each step, to consider a sampled action, the weighted average of all the estimates, the best action or a combination of those.
An important property of n-step Temporal Difference methods is that they are also a generalization of Monte Carlo methods: if we consider environments whose simulation end in a finite number of steps (the only one in which it makes sense to use Monte Carlo methods), by considering a very high value of n and a Temporal Difference method we get the same results.
Several examples throughout the whole work showed how the n-step methods can obtain better results, due to their flexibility. On the other hand, the lack of a general method to find the optimal value of the number of steps, induces additional computations.

One of the most relevant issues in Reinforcement Learning is the exploitation-exploration dilemma, presented in chapter 5 it describes the problem of understanding when it is more convenient to explore the environment and when, instead, it is more favorable to exploit the current information and simply follow what is considered the best action. Intuitively, at the beginning it might be more convenient to explore, while after some time it might be better to simply follow the optimal policy. This involves the choice of the action selection method (or equivalently, the exploration strategy): the constant-ϵ-greedy method, which heavily favors the best action, discussed in the first chapter is a good choice, but it can be showed how alternative methods can give much better results. A first overview of these

methods is seen in chapter 5, applied to the simple case of the Bernoullian bandit. Alternative strategies, such as the Boltzmann exploration and the ϵ-greedy and can be more effective in several examples. All the exploration methods described above, build their strategy by looking only at the punctual estimates of the values. It can also be possible to define alternative and more complex approaches: in particular, the Upper Confidence Bound method (UCB) builds, for each value of the q-function, a confidence interval and then selects the action having the highest possible value, which intuitively is the one that potentially can obtain the best results. An alternative class is the one of Bayesian methods: these define, for each state-action pair, an appropriate distribution. In the case of the Bernoullian bandit, this distribution is a simple Beta, having as initial parameters $\alpha = 0.5, \beta = 0.5$. These last two methods, Bayesian and UCB, are united by the idea of using, not only the value of the estimate, but also the amount of confidence in it. By using both information, it is then possible to take more informed decision, that proved to be better in some of the examples studied.

All the exploration methods described, can be extended to more general Markov Decision process. In this case the Bayesian methods make use of a normal distribution, with unknown mean and precision to describe the distribution of the expected reward.

The Bayesian methods are capable of obtaining very interesting results in several examples: their ability to describe the value using a distribution, which gives us an idea on the precision on the estimate, is definitely interesting and can help when deciding which action to take. Moreover, with the appropriate assumptions, it is possible to reduce the calculations required to describe the distributions in a Bayesian framework. However they still require more computational power, in particular when defining the initial parameters of the distribution.

Research on Reinforcement Learning does not stop here: in particular, two main topics have not been touch (if not briefly) in this work. In particular, there exist the *model-based* methods, that try also to estimate the model of the environments and not only the q-function. However, the main focus of today's research is on approximate methods, that are capable of studying more complex environments and therefore can solve problems more related to the real world. As seen in chapter 7, researchers are now capable of applying the methods studied in this work to healthcare, robotics and finance, among others.

Naturally, an eventual future work could focus on the aforementioned approximate methods and on the applications of Bayesian methods to that framework. In particular, in that case one should find a way to approximate the parameters that define the distribution associated to each state-action pair, without losing relevant information.

One might also try to improve the methods described in this work: of great interest might be a n-step version of the Bayesian methods: it has been observed how the traditional n-step methods are a much better solution than their 1-step version, hence it is natural to wonder what might be the results of a generalized version of the 1-step Bayesian methods studied in chapter 6. Regarding this topic, it might be interesting to see if the work done

on the n-step version with greedy action selection can be verified even on other examples. Other future research might also focus on possible improvements on the 1-step algorithms: as described in the dedicated chapter, the performances of these methods are limited by poor parameter update algorithms. In particular, the *moment update* studied here tends to converge quickly to a potentially sub-optimal strategy (as probably observed in the Mountain Car example), while the *mixture update* requires heavy numerical computations. At the same time, it is possible that alternative action selection approaches to the one proposed might give better results.

A

Appendix

A.1 Incremental implementation of the importance sampling formulas

The computations carried out in this section give us a simpler representation of the update of the Importance Sampling ratio, usually computed in the context of *off-policy* methods.

A.1.1 Ordinary importance sampling

$$
\begin{aligned}
V_n(s) &= \frac{\sum_{t \in \tau_n(s)} \rho_{t:T(t)-1} G_t}{|\tau_n(s)|} \\
&= \frac{\sum_{t \in \tau_n(s)} \rho_{t:T(t)-1} G_t}{n} \\
&= \frac{\sum_{t \in \tau_n(s)} A_i}{n} \quad \text{with } A_i = \rho_{t_i:T(t_i)-1} G_{t_i} \\
&= \frac{1}{n} \left(A_n + \sum_{i=1}^{n-1} a_i \right) \\
&= \frac{1}{n} \left(A_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} A_i \right) \\
&= \frac{1}{n} \left(A_n + (n-1) V_{n-1}(s) \right) \\
&= \frac{1}{n} \left(A_n + n V_{n-1}(s) - V_{n-1}(s) \right)
\end{aligned}
$$

$$
\begin{aligned}
&= V_{n-1}(s) + \frac{1}{n}\left(A_n - V_{n-1}(s)\right) \\
&= \frac{\sum_{t\in\tau_{n-1}(s)} \rho_{t:T(t)-1} G_t}{n-1} + \frac{1}{n}\left(\rho_{t_n:T(T_n)-1} G_{t_n} - \frac{\sum_{t\in\tau_{n-1}(s)} \rho_{t:T(t)-1} G_t}{n-1}\right)
\end{aligned}
$$

A.1.2 Weighted importance sampling

$$
\begin{aligned}
V_{n+1}(s) &= \frac{\sum_{i=1}^{n} W_k\ G_k}{C_n} \\
&= \frac{\sum_{i=1}^{n-1} W_k\ G_k + W_n\ G_n}{C_n} \\
&= \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_n}\frac{C_n - W_n}{C_{n-1}} + \frac{W_n\ G_n}{C_n} \\
&= \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_{n-1}}\left(1 - \frac{W_n}{C_n}\right) + \frac{W_n\ G_n}{C_n} \\
&= \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_{n-1}} - \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_{n-1}}\frac{W_n}{C_n} + \frac{W_n\ G_n}{C_n} \\
&= \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_{n-1}} + \frac{W_n}{C_n}\left(G_n - \frac{\sum_{i=1}^{n-1} W_k\ G_k}{C_{n-1}}\right) \\
&= V_n(s) + \frac{W_n}{C_n}\left(G_n - V_n(s)\right)
\end{aligned}
$$

A.2 Contraction

Definition A.1. (Contraction mapping)
Given a metric space (X, d), a function f from X to itself is said a contraction if there exists some nonnegative number $0 \leq k \leq 1$ such that the following is true:

$$d\left(f(x), f(y)\right) \leq k\ d\left(x, y\right) \quad \forall x, y \in X$$

An important property of contractions is that each contraction map has at most one fixed point. Moreover can also be proved the following theorem:

Theorem A.1. ***(Banach fixed point theorem)***
Let (X, d) be a non-empty complete metric space, with a contraction mapping $T : X \to X$. Then T admits a fixed point $x^\star$, a point such that $T\left(x^\star\right) = x^\star$.

Given these two important results of analysis, we can define the following map.

Definition A.2. (Map H)
Let q be a generic function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, then we can define a map H such that

$$\begin{aligned}\left(Hq\right)(s, a) &= \left[r(s, a, s') + \gamma \max_{b \in \mathcal{A}} q(s', b)\right] \\ &= \mathbb{E}\left[r(s, a, s') \mid s, a\right]\end{aligned} \tag{A.1}$$

where $P_a(s, s')$ represents the probability of getting from state s to s' after doing the action a.

The mapping H, defined in the space of functions such as q, is a contraction if the space is endowed with the distance based on the infinite norm.

Proposition A.2. *Let M be the space of functions $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and d_∞ the distance given by the infinite norm, such that $d_\infty(x, y) = \|x - y\|_\infty$. (M, d_∞) is a metric space.*

We can finally prove that the map $H : M \to M$ is indeed a contraction on the metric space (M, d_∞):

Proposition A.3. *The map H as defined in eq.* (A.1) *is a contraction on the metric space (M, d_∞).*

Proof. Let q_1 and q_2 be two functions in M

$$\begin{aligned}
\|Hq_1 - Hq_2\|_\infty &= \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P_a(s,s') \left[r(s,a,s') + \gamma \max_{b \in \mathcal{A}} q_1(s',b) \right] - \right. \\
&\qquad \left. - \left[r(s,a,s') + \gamma \max_{b \in \mathcal{A}} q_2(s',b) \right] \right| \\
&= \gamma \max_{s,a} \left| \sum_{s' \in \mathcal{S}} P_a(s,s') \left[\max_{b \in \mathcal{A}} q_1(s',b) - \max_{b \in \mathcal{A}} q_2(s',b) \right] \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P_a(s,s') \left| \max_{b \in \mathcal{A}} q_1(s',b) - \max_{b \in \mathcal{A}} q_2(s',b) \right| \\
&\leq \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P_a(s,s') \max_{b \in \mathcal{A}} \left| q_1(s',b) - q_2(s',b) \right| \\
&= \gamma \max_{s,a} \sum_{s' \in \mathcal{S}} P_a(s,s') \, \|q_1 - q_2\|_\infty \\
&\leq \gamma \, \|q_1 - q_2\|_\infty
\end{aligned}$$

□

Corollary A.4. *There exists a fixed point for the map H defined in eq.* (A.1)*. Moreover the fixed point is the optimal Q function associated to the environment with the transition probabilities defined by $P_a(s,s')$.*

Proof. The first is a consequence of Theorem A.1 and the fact that H is a contraction. The second part is a consequence of the Bellmann equations 1.6. □

A.3 Proof that $Q(\sigma)$ is a generalization of the n-step methods

This chapter will contain all the calculations that prove that, for $\sigma_t = 0 \ \forall t$ the $Q(\sigma)$ algorithm corresponds to the Tree Backup algorithm presented in Algorithm 16, while for $\sigma_t = 1 \ \forall t$ we have Algorithm 15.

A.3.1 $\sigma_t = 0 \ \forall t$

In this case the modified importance sampling ratio, whose formula is given in (4.14) is always equal to 1, $\rho_{t+1}^{t+n} = 1$.
Moreover, $\delta_t^{\sigma=0} = R_{t+1} + \gamma \tilde{V}_t(S_{t+1}) - Q_{t-1}(S_t, A_t)$. Given these quantities, we have

$$\begin{aligned}
G_{t:t+n} = Q_{t-1}(S_t, A_t) \quad & + \sum_{k=t}^{\tau} \delta_k^0 \prod_{i=t+1}^{k} \gamma \Big[(1-\sigma_i)\pi(A_i \mid S_i) + \sigma_i\Big] \\
= Q_{t-1}(S_t, A_t) \quad & + \sum_{k=t}^{\tau} \Big(R_{k+1} + \gamma \tilde{V}_k(S_{k+1}) - Q_{k-1}(S_k A_k)\Big) \prod_{i=t+1}^{k} \gamma \pi(A_i \mid S_i) \\
= Q_{t-1}(S_t, A_t) \quad & + R_{t+1} + \gamma \tilde{V}_t(S_{t+1}) - Q_{t-1}(S_t, A_t) + \\
& + \Big(R_{t+2} + \gamma \tilde{V}_{t+1}(S_{t+2}) - Q_t(S_{t+1}, A_{t+1})\Big)\gamma \cdot \\
& \qquad \cdot \pi(A_{t+1} \mid S_{t+1}) + \\
& + \Big(R_{t+3} + \gamma \tilde{V}_{t+2}(S_{t+3}) - Q_{t+1}(S_{t+2}, A_{t+2})\Big)\gamma \cdot \\
& \qquad \cdot \pi(A_{t+1} \mid S_{t+1})\, \pi(A_{t+2} \mid S_{t+2})
\end{aligned}$$

$$\begin{aligned}
&= R_{t+1} + \gamma \Big(\tilde{V}_t(S_{t+1}) - \pi(A_{t+1} \mid S_{t+1})\, Q_t(S_{t+1}, A_{t+1})\Big) + \\
&\qquad + \gamma \pi(A_{t+1} \mid S_{t+1}) \Big(R_{t+2} + \gamma \tilde{V}_{t+1}(S_{t+2}) + \ldots\Big) \\
&= R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a \mid S_{t+1})\, Q_t(S_{t+1}, a) + \gamma \pi(A_{t+1} \mid S_{t+1})\, G_{t+1:t+n}
\end{aligned}$$

This return is exactly the one in (4.12).

A.3.2 $\sigma_t = 1 \ \forall t$

In this second case the modified importance sampling ratio, whose formula is given in (4.14) corresponds to the traditional one, (2.5). Moreover, the error is given by

$$\delta_t^{\sigma=1} = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t)$$

Given this quantities, we have:

$$
\begin{aligned}
G_{t:t+n} = Q_{t-1}(S_t, A_t) \quad & + \sum_{k=t}^{\tau} \delta_k^0 \prod_{i=t+1}^{k} \gamma \left[(1-\sigma_i)\pi(A_i \mid S_i) + \sigma_i \right] \\
= Q_{t-1}(S_t, A_t) \quad & + \sum_{k=t}^{\tau} \delta_k^0 \gamma^{k-t} \\
= Q_{t-1}(S_t, A_t) \quad & + \delta_t^1 + \delta_{t+1}^1 \gamma + \cdots + \delta_\tau^{\tau-t} \\
= Q_{t-1}(S_t, A_t) \quad & + R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t) + \\
& + \gamma \left(R_{t+2} + \gamma Q_{t+1}(S_{t+2}, A_{t+2}) - Q_t(S_{t+1}, A_{t+1}) \right) + \\
& + \gamma^2 \left(R_{t+3} + \gamma Q_{t+2}(S_{t+3}, A_{t+3}) - Q_{t+1}(S_{t+2}, A_{t+2}) \right) + \\
& + \dots \\
= R_{t+1} + \gamma R_{t+2} \quad & + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})
\end{aligned}
$$

Bibliography

Andrew G Barto and Richard S Sutton. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4(3): 221–235, 1982.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.

Donald A Berry and Bert Fristedt. Bandit problems: sequential allocation of experiments (monographs on statistics and applied probability). *London: Chapman and Hall*, 5 (71-87):7–7, 1985.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume II*, volume 2. Athena scientific, 2012.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Pawel Cichosz. Truncating temporal differences: On the efficient implementation of td (lambda) for reinforcement learning. *Journal of Artificial Intelligence Research*, 2: 287–318, 1994.

Kristopher De Asis, J Fernando Hernandez-Garcia, G Zacharias Holland, and Richard S Sutton. Multi-step reinforcement learning: A unifying algorithm. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *Aaai/iaai*, pages 761–768, 1998.

Richard Dearden, Nir Friedman, and David Andre. Model-based bayesian exploration. *arXiv preprint arXiv:1301.6690*, 2013.

Markus Dumke. Double q(σ) and q(σ, λ): Unifying reinforcement learning control algorithms. *arXiv preprint arXiv:1711.01569*, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*, 2016.

Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

George H John. When the best move isn't optimal: Q-learning with exploration. In *AAAI*, volume 1464. Citeseer, 1994.

JL Kelly. A new interpretation of information rate. *Bell System Technical*, 1956.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

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

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

Tom Mitchell. Solving known MDPs. `https://www.andrew.cmu.edu/course/10-703/slides/lecture4_valuePolicyDP-9-10-2018.pdf`, 2018. [Online slides].

Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

Ivan Petrovitch Pavlov and William Gantt. Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals. 1928.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

Thomas Ross. Machines that think. *Scientific American*, 148(4):206–208, 1933.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1):123–158, 1996.

Dennis Soemers. How do we prove the n-step return error reduction property? `https://ai.stackexchange.com/a/9426`, 2019. [Online; accessed 17-August-2021].

Stefan Steinerberger. On the number of positions in chess without promotion. *International Journal of Game Theory*, 44(3):761–767, 2015.

Malcolm Strens. A bayesian framework for reinforcement learning. In *ICML*, volume 2000, pages 943–950, 2000.

Richard S Sutton. Learning theory support for a single channel theory of the brain, 1978a. Unpublished report.

Richard S Sutton. Single channel theory: A neuronal theory of learning. *Brain Theory Newsletter*, 4:72–75, 1978b.

Richard S Sutton. A unified theory of expectation in classical and instrumental conditioning. 1978c.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

Richard S Sutton and Andrew G Barto. Toward a modern theory of adaptive networks: expectation and prediction. *Psychological review*, 88(2):135, 1981.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

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

Edward O Thorp. *Beat the Dealer: a winning strategy for the game of twenty one*, volume 310. Vintage, 1966.

Sebastian B Thrun. Efficient exploration in reinforcement learning. 1992.

Hado van Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

Hado Philip van Hasselt. *Insights in reinforcement rearning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. Utrecht University, 2011.

Harm van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv preprint arXiv:1608.05151*, 2016.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *2009 ieee symposium on adaptive dynamic programming and reinforcement learning*, pages 177–184. IEEE, 2009.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement learning in healthcare: A survey. *arXiv preprint arXiv:1908.08796*, 2019.