POLITECNICO DI MILANO
Master of Science Degree in Computer Engineering
Department of Electronics and Information

Boosted Fitted Q-Iteration

AI & R Lab
Artificial Intelligence and Robotics Lab
of Politecnico di Milano

Advisor: Prof. Marcello Restelli
Co-advisors: Ing. Matteo Pirotta, Ing. Carlo D'Eramo

Thesis Author:
Samuele Tosatto 820176

Academic Year 2015-2016

Contents

List of Figures

List of Algorithms

To the beautiful sunlight
who used to play hide and seek
between the falling random leaves
of a golden tree.

Sommario

Recentemente abbiamo assistito ad un consistente avanzamento nell'ambito dell'Intelligenza Artificiale. Sono state sviluppate applicazioni che sono capaci in molti casi di superare le capacità umane in diversi tipi di compiti, come nel caso di GO, un gioco da tavolo [70]. Abbiamo anche assistito ad uno sviluppo di prototipi per la guida autonoma [47], compravendita automatica di azioni nell'ambito finanziario [53], robot industriali e domestici [44]. Queste applicazioni sono spesso troppo complesse per essere risolte con un generico software scritto con l'aiuto di un esperto: è qui che entra in gioco il machine learning, ossia l'apprendimento automatico. Spesso le tecniche di machine learning richiedono enormi risorse computazionali e grandi quantità di dati: il nostro lavoro si propone di fornire un metodo che usi le risorse computazionali e i dati in maniera efficiente. Tale metodo si colloca nella classe di algorithmi di machine learning chiamata Reinforcement Learning [73]. Il reinforcement learning si occupa di risolvere problemi dove vi è un agente immerso in un mondo che può osservare e nel quale può effettuare azioni che ne modificano lo stato. Più precisamente, l'agente deve massimizzare nel tempo un segnale di "reward" che dipende dallo stato e dalle azioni intraprese dall'agente: tale segnale indica in genere se l'agente sta risolvendo o meno il problema proposto. Esistono diverse metodologie per risolvere problemi di RL. Il nostro metodo rientra più precisamente nella classe di algoritmi Approximate Value Iteration [61], dove il cuore dell'algoritmo è quello di stimare, tramite approssimazioni successive, il valore associato ad ogni coppia di stato azione, in modo tale che sia poi possibile scegliere le azioni che massimizzano l'accumulo di reward. Il nostro metodo Boosted Fitted Q-Iteration (B-FQI) poggia su una solida struttura teorica, e adatta un metodo già presente e molto conosciuto in letteratura nel supervised learning al caso del reinforcement learning: il boosting. Il boosting consente di utilizzare dei modelli più semplici che richiedono minori risorse computazionali e che adattano dinamicamente la loro complessità in base al target desiderato. Il nostro algoritmo introduce il boosting in modo efficiente nel

reinforcement learning. Verrà fornita in questa sede una valutazione teorica, affrontando sia come l'errore si propaga iterazione dopo iterazione, sia come l'errore dipenda dalla corposità del dataset e dall'espressività dello spazio funzionale. Infine riportiamo i risultati sperimentali, che supportano le argomentazioni teoriche per quanto riguarda la capacità di B-FQI di usare modelli più semplici rispetto al generico AVI, e fornendo anche un'analisi dal punto di vista dell'efficienza nell'uso dei dati. B-FQI risulta efficiente per quanto riguarda l'uso di risorse computazionali, e merita sicuramente ulteriori studi oltre quelli effettuati nella tesi presente.

Summary

In the recent years we have seen a general technological advance in the field of Artificial Intelligence. Researchers and high-tech companies have developed many applications that are able to beat humans in different kind of tasks, like for the board game GO [70]. We have also seen prototyping autonomous vehicles [47], automatic financial traders [53], industrial and domestic robots [44]. This kind of applications are often too complex to be solved with a generic software wrote with the help of an expert: it is here that machine learning starts to play its role. Very often machine learning requires enormous computational resources and big datasets: our work aims to provide a method that efficiently uses computational resources and data. This method belongs to the class of semi-supervised machine learning algorithms called Reinforcement Learning [73]. The goal of Reinforcement Learning is to provide a technique to solve problems where there is an agent surrounded by an environment that it can both observe and interact by means of actions. More in details the agent wants to maximize a reward signal, which depends by the actual state and performed actions: such signal often indicates how well the agent is behaving. There are several ways to solve reinforcement learning tasks. Our method belongs more precisely to the class of algorithms named Approximate Value Iteration [61], where the core is to estimate the value associated to each state-action pairs, in such way that it will be possible to choose actions that maximize the reward it receives over time. Our method Boosted Fitted Q-Iteration (B-FQI) relies on a solid theoretical foundation, and adapts a preexisting and well known in literature method of supervised learning, to the case of reinforcement learning: boosting [13]. Boosting allows using simpler models that require less computational resources, and that dynamically adapts its complexity to the desired target. Our algorithm introduces efficiently boosting in reinforcement learning. We will provide here a theoretical analysis that will concern both on how the error propagates through iterations, and on how error depends on both the choice of the expressivity of the functional space and

the size of the dataset. We eventually provide the empirical results, which supports our theoretical statements for what concerns the ability of B-FQI to use simpler models with respect to the generic FQI, and we provides also an analysis from the data-efficiency point of view. B-FQI results to be efficient for what concerns the usage of computational resource and deserves undoubtedly further studies beyond the ones developed in this thesis.

Ringraziamenti

Sono passati alcuni anni da quando, uscito dalle scuole superiori, mi sono iscritto in questa università. E' stata un esperienza intensa, che ho potuto condividere con persone che fanno parte della mia vita e che l'hanno resa in qualche modo speciale ed unica al mondo (come la vita di tutti, del resto). Ringrazio innanzitutto chi mi ha seguito nell'ultimo tratto di questo percorso: il mio relatore Marcello Restelli, ed i miei co-relatori Matteo Pirotta e Carlo D'Eramo. Mi hanno guidato nello sviluppo di questo lavoro nel migliore dei modi possibili e sono per questo loro grato. Sono molto contento di poter chiudere con il presente lavoro questo lungo percorso. Ringrazio le persone che mi hanno formato professionalmente, e il mio pensiero va immediatamente a Borgo Lorenza, Fabio Castelli e Guerrini Alberto. Voglio ringraziare i compagni di corso con i quali ho legato di più: Matteo, Daniel, Samuele, Omar, Simone, Luigi e Federico nel triennio; Mohammed, Martin, Camillo e Miguel alla specialistica. Ringrazio di cuore la compagnia valtellinese, con la quale ho condiviso i ricordi migliori: Mich, Vero, Fra, Tina, Zu, Scretch, Rosh (valtellinese acquisito), Amedeo e Spezz (non valtellinesi). Vorrei ringraziare un amico speciale: Andrian, con il quale ho condiviso momenti indimenticabili (vedi gli "spiriti"). Per associazione di idee del tutto casuale, ringrazio anche un'amica speciale "Still Alive Agnese": l'unica persona che abbia rischiato (suo malgrado) la vita a causa mia. Ringrazio il mio amico inventore tutto-fare Efrem, compagno di idee e favolosi progetti puntualmente irrealizzati. I would like to thanks also all those people that I have met during my years abroad, and in particular I would like to thanks my roommate Lada and my italian mates Alberto e Cosimo. Ringrazio col cuore Eleonora, Greta, Monica, Chicca e Guli: quella parte di famiglia, che con me condivide il seme della follia. Un abbraccio grande ai miei nonni che non sono potuti venire alla mia laurea ma che so che avrebbero voluto. In realtà sono felice per loro, che restano in quell'angolo di paradiso in Sardegna, baciati da un sole meraviglioso. Ma sopra ogni cosa ringrazio colei che ha consentito questo percorso, che mi ha sostenuto, e ha creduto ciecamente in me: mia mamma Irene, ed insieme a lei il suo compagno Lanfranco. Mi rincresce pensare che in una così bella giornata probabilmente una persona molto importante per me sarà assente: nonostante i diverbi che abbiamo recentemente avuto, ti voglio bene papà, e ti ringrazio per il tuo sostegno durante questi anni. Ringrazio infine la mia ragazza Silvia, che mi ha supportato (e sopportato) per questi due lunghi anni e mi ha donato il suo bellissimo amore (ed anche molte prelibate cibarie valtellinesi). So bene che la lista è lontana dall'essere esaustiva e mancano ancora tante persone a cui dovrei un grazie, non si sentano offese dalla mancanza di un grazie tra queste righe, che infine altro non è che mero inchiostro su carta: sanno in cuor loro il bene che gli voglio.

Chapter 1

Introduction

"Alea iacta est"

Gaius Iulius Caesar

1.1 Contextualization

In the recent years, we have seen a general technological advance in the field of Artificial Intelligence. There were developed some application that are able to beat humans in different kind of tasks, like game playing [51], [70], but also we have seen the prototyping of autonomous vehicles [47], automatic financial traders [53], industrial and domestic robots [44]. All those successful results were unimaginable a few years ago, and the main technologcal and scientific improvements that allowed to achieve such goals are often in the field of Machine Learning (ML), and more in particular of Reinforcement Learning (RL). In fact, this kind of applications are often too complex to be solved by a generic software programmed by an expert: there is the need that the machine autonomously *learns* how to solve such difficult tasks, from experience or other source of learning (like imitation learning). The field of ML is very huge, and it is living in these years its golden age, with a thousand of articles every year, and an impressively fast technological advancement [40]. ML has become a very appealing way to solve many problems and it has started to be used more and more often in very different kind of applications. On the other hand, also ML has some drawbacks: it often requires a huge amount of data and computational resources. This drawback makes ML expensive, and while on one side it is appealing to conduct research in order to solve complex tasks, on another side it is also interesting to study a way to make ML lighter, requiring less computational resources or data. In this work, we followed the second approach.

1.2 Goal

RL is a branch of ML where the main goal is to make an agent (e.g. a robot) to observe an environment and choose some actions to perform in order to maximize reward collection over a time horizon. The way the agent chooses the actions it is called *policy*, and the main issue in RL is to find the so called *optimal policy* that optimizes the reward function. There are several ways to solve RL tasks, but they all can be divided in two main classes: Value Based or Policy Based [73]. The value based methods are based on a representation of the value function: a function that represents the value of each different situation so that the agent will be able to choose the proper action. The representation of the value function is often obtained by means of a Supervised Learning (SL) method that is able to approximate such function giving a set of data. Policy based methods use instead an internal representation of the policy and they focus to find the optimal policy without the value function. Actor-Critic method are mixed approaches, where the core of the algorithm is to keep track of both the value function and the policy representation. Value based methods are very important in the literature: they have been deeply studied, and they are shown to reach in some applications very impressive results. The goal of our thesis is to provide an efficient way to solve RL task by means of a value based method.

1.3 Motivation

The recent advances in the field of RL are often related both to the tecnological advances in terms of hardware (faster CPUs, more parallel computing, more computational resources in general), and to the progress made in supervised learning. Deep learning [48] is a clear example about the progress made in the former field. There are a number of recent progresses in feature extraction for image processing (like convolutional neural networks) [46], for time dependencies extraction (deep recurrent neural networks) [34], but also in dimensionality reduction with autoencoder [77], just to mention a few. And of course all these progress had an impact in RL, like in Deep Q-Network (DQN) [51] where convolutional networks are used. All these SL methods are often very specific to the problem: we tackle the issue of high dimensionality with ad-hoc techniques, using features extraction relying on our knowledge about the nature of the inputs. We wanted to find an efficient way to solve RL tasks using our knowledge about some properties that are shared among all the RL tasks, and in particulat using those properties that are related to the value function. In fact, all the value based techniques exploit mathematical properties (as we will see in Chapter 2) of the optimal value function in order to find it. What we have found during our research, is a method that exploits some properties of the optimal value function in order to gain time efficiency. The strength of our solution is that it does not depend on the nature of the problem (such as the nature of input), but is instead a very general solution.

1.4 Outline of the Thesis

Boosted Fitted Q-Iteration(for short B-FQI) is a value based algorithm that belongs to the class of Approximate Value Iteration (AVI) algorithms which iteratively build an approximation of the value function. The main advantage of B-FQI with respect to other AVI algorithm is that it should reach a good approximation of the value function with less computation. This result is achieved basically by decomposing the optimal value function as the sum of different terms each one of those can be approximate with simple models, requiring in this way less computing effort. General AVI methods in fact require to approximate each iteration the whole value function, even when the approximation of the value function is good and thus it does almost not change from one iteration to the next one: B-FQI instead of approximating each iteration the value function, it approximates the so called “residual” that consists in the difference between the current approximation of the value function and the previous one. Approximating the residual gives several benefits: the most easy to see is that the residual converges to zero as the value function covnerges to the optimal one. The other advantage is that the residual is easier to approximate even before approaching the convergence. The reader may wonder why approximating the residual is in general a simpler task with respect to approximating the whole value function: after all the residual is a difference between two complex functions, and no-one can guarantee that the difference between two complex functions is a simpler function. The key point is that the residual is the difference of two related functions: in fact it is the difference of two consecutive approximations of the value function. Approximating the residual is not a new idea in machine learning: boosting [14] is a SL technique that provides a general meta algorithm to iteratively approximate the gradient of the loss function, and it could be easily shown that when the error is the l_2-loss, the gradient coincides with the residual [15]. The interesting feature of boosting is that it produces an approximation of a target function by means of several weak lerners where each of those is an approximation of the gradient, or in the l_2-boosting case, is an approximation of the residual. A weak learner is a function approximator that uses a very simple model to approximate the target function, and thus weak learners are in general very efficient. We can show that B-FQI introduces boosting, and thus it approximates the residual at each iteration by means of weak learners, gaining time efficiency. We show in the next chapters the theoretical basis on which our method relies on, and then we also will prove empirically its computing advantage.

In Chapter 2 we introduce the state of the art. This chapter could be logically divided in two main parts: in the first part we introduce the theory behind value based techniques in reinforcement learning, focusing on AVI techniques, while in the second part we describe boosting, which is a supervised learning techniques, giving an idea to the reader on why boosting seems to be so effective. In Chapter 3 we display the theoretical analysis of AVI methods and of B-FQI. In Chapter 4 we describe the experiments that we performed and the empirical results. In Chapter 5 we derive our conclusions and we outline the future research.

Chapter 2

State of the Art

"Tell me and I forget, teach me and I may remember, involve me and I learn."

Benjamin Franklin

This chapter is devoted to the presentation of Reinforcement Learning. We present in Section 2.1 the mathematical framework on which is based, starting from the Markovian decision processes in Section 2.1.1, then showing the main theoretical techniques of dynamic programming in Section 2.1.2 with a special focus on approximate value iteration on which this work is based. We present some value-based algorithms used in Reinforcement Learning with a special focus on an approximate value iteration technique named Fitted Q-Iteration in subsection 2.1.3. In section 2.2 we present a supervised learning technique that, in order to lower down the bias (which will be introduced in Section 2.2.1), uses a linear combination of simple models (weak regressor). In order to present boosting, we introduce in Section 2.2.1 the supervised learning setting, then we do a digression in Section 2.2.1 on the bias-variance trade-off which is one of the main issues in supervised learning. In Section 2.2.2 we show what ensemble methods and boosting methods are, while in Section 2.2.3 we introduce the functional gradient view of boosting techniques. We eventually describe in Section 2.2.4 $L-2$ Boost: a particular case of boosting which is suitable for a theoretical analysis and is the kind of boosting used in B-FQI.

2.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) is one of the most active research areas in artificial intelligence. It has been applied successfully on a number of dif-

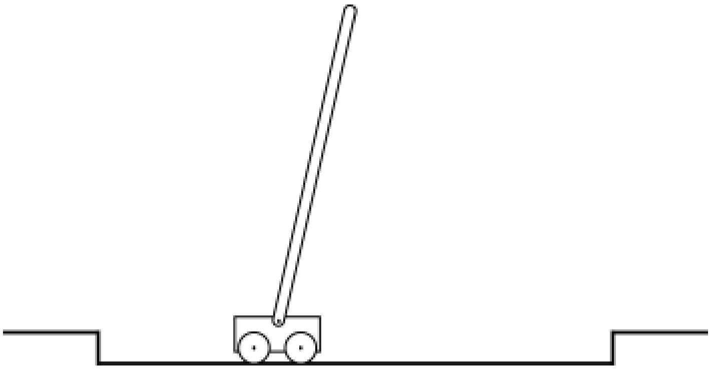

Figure 2.1: Cart-Pole environment.

ferent control problems, such as job-shop scheduling [82], backgammon [74], elevator control [17], machine maintenance [49], dynamic channel allocation [71], airline seat allocation [33], and more recently on playing Atari games [51], mastering the game of Go [70], just to mention a few. RL consists in a computational approach to solve sequential decision problems where an agent tries to maximizes a numerical reward by interacting with an environment through the usage of a set of actions. The goal is to find the so called *optimal policy* which is basically a function that associates to each possible observed state, the best action to choose in order to maximize the numerical reward: in order to do that is not possible to just maximize the immediate reward at each step; but very intuitively sometimes the agent must choose an action that will make receive a low immediate reward, but which allows to get higher future rewards.

Example. To give an example of a practical RL task (that will be also discussed in details in the experimental section), we consider the Cart Pole task, which consists in balancing a pole in upright position by the means of the force to apply to a car where the pole is attached. The reward could consists in a positive reward when the pole is with vertical position, while a zero reward could be given when the pole falls. Thus the state observed by the agent will be the position of the cart, the angle of the pole in respect to the vertical, and the velocities of the cart and the angle of the pole. The set of actions are the possible forces that could be applied to the cart. The optimal policy will be in this case a function that has as input the observed state (position, angle, velocities), and as output, the action that will make the pole stay as long as possible up right, without making it fall.

2.1.1 Markovian Decision Processes

A Markovian Decision Process (MDP) [5, 6] is a mathematical framework that describes a sequential decision making problem in a kind of situation where the outcomes are partially stochastic and partially influenced by the decisions made.

For a space Σ, with σ-algebra σ_Σ, $\mathcal{M}(\Sigma)$ denotes the set of probability measures over σ_Σ[1]. $\mathcal{B}(\Sigma, B)$ denotes the space of bounded measurable functions w.r.t. σ_Σ with bound B (B is omitted when is unknown). A discounted MDP is a tuple $(\mathcal{X}, \mathcal{A}, P, \mathcal{R}, \gamma, \mu)$, where $\mathcal{X}$ is a measurable state space, $\mathcal{A}$ is a set of actions, $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the transition probability kernel, $\mathcal{R} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function[2], $\gamma \in [0, 1)$ is the discount factor, $\mu : \mathcal{M}(\mathcal{X})$ is the probability of the initial state. Let $r(x, a) = \mathbb{E}\left[\mathcal{R}(\cdot|x, a)\right]$ to be uniformly bounded by $R_{\max}$. A policy $\pi : \mathcal{X} \times \mathcal{B}(\mathcal{A})$ is a mapping from $\mathcal{X}$ to a distribution over $\mathcal{A}$. As a consequence of taking action A_t at X_t we receive a reward signal $R_t \sim \mathcal{R}(\cdot|x, a)$ and the state evolves accordingly to $X_{t+1} \sim P(\cdot|X_t, A_t)$.

The MDPs rely on the Markov assumption which states that the reward function $\mathcal{R}$ and the transition probability P only depend on the actual state, and is conditionally independent from the previous history.

$$P(x_{t+1}|x_t, a_t, x_{t-1}, a_{t-1}, \ldots, x_1, a_1) = P(x_{t+1}|x_t, a_t) \tag{2.1}$$

$$\mathcal{R}(x_t, a_t, x_{t-1}, a_{t-1}, \ldots, x_1, a_1) = \mathcal{R}(x_t, a_t). \tag{2.2}$$

From now on we will always make the assumptions of continuous state-space and action-space and infinite horizon (there is no terminal state), unless differently stated.

The Value Function

In the previous section we explained that the agent's goal is to maximize a numerical reward through its "life". This is an informal definition, because it does not precisely specify what the agent should precisely maximize. There are several different objectives that can be defined. We introduce just two of them: the *Discounted Reward* J_D^π and the *Averaged Reward* J_A^π:

$$J_D^\pi = \mathop{\mathbb{E}}_{x \sim \mu}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | x_0 = x\right] \tag{2.3}$$

[1] We strictly follow here the notation used in [23, 24].

[2] We could have defined $\mathcal{R} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}$ as some authors do, where $\mathcal{R}(x, a, y)$ is the mean reward of the transaction from x to y by means of choosing action a. Actually it is possible to re-define $R(x, a) = \int_{\mathcal{X}} R(x, a, y) P(y|x, a) \mathrm{d}y$.

$$J_A^\pi = \mathop{\mathbb{E}}_{x\sim\mu}\left[\frac{\sum_{t=1}^T r_t}{T}|x_0 = x\right]. \tag{2.4}$$

From now on, we will assume that the discounted reward is used, averaged reward is reported only for sake of completeness. We define the return v_t at time step t as:

$$v_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.5}$$

We can see that γ close to zero will lead to a “myopic evaluation” in the sense that the agent will seek to find a policy that maximizes the close upcoming rewards, while with a γ close to one, the evaluation will be “far-sighted”, leading the agent to find a policy that chooses actions in such a way to also maximize the future rewards.

The *state value function* $V \in \mathcal{V}$ where $\mathcal{V} = \{V|V : \mathcal{X} \to \mathbb{R}\}$ is the expected return from the state $x \in \mathcal{X}$:

$$V(x) = \mathbb{E}\left[v_t|x_t = x\right]. \tag{2.6}$$

This value always depends from the policy followed, because all the next states depend on the policy used too. Let’s suppose that all the states and the rewards are observed by means of a policy π. We indicate a functions that specifies for each state s its expected utility obtained V^π as the state value function by following the policy π. Let’s also define the following

$$\mathcal{R}^\pi(x) = \int_{\mathcal{A}} \mathcal{R}(x,a)\pi(a|x)\mathrm{d}a \tag{2.7}$$

$$P^\pi(y|x) = \int_{\mathcal{A}} P(y|x,a)\pi(a|x)\mathrm{d}a \tag{2.8}$$

$\mathcal{R}^\pi$ and P^π correspond respectively to the *expected reward* and the *expected transition probability* when the policy π is followed. Then

$$\begin{aligned}
V^\pi(x) &= \mathop{\mathbb{E}}_{\pi}\left[v_t|x_t = x\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[r_t + \gamma v_{t+1}|x_t = x\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[r_t + \gamma V^\pi(x_{t+1})|x_t = x\right] \\
&= \int_{\mathcal{A}} \pi(a|x)\left(R(x,a) + \gamma\int_{\mathcal{X}} P(y|x,a)V^\pi(y)\mathrm{d}y\right)\mathrm{d}a \\
&= R^\pi(x) + \gamma\int_{\mathcal{X}} P^\pi(y|x)V^\pi(y)\mathrm{d}y
\end{aligned} \tag{2.9}$$

is the Bellman equation, and its solution $V^\pi(x)$ represents the expected cumulative discounted return by following the policy π from the state x. We also introduce in a similar way the action-state value function $Q^\pi \in \mathcal{Q}$ where $\mathcal{Q} = \{Q | Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}\}$ and where $Q^\pi(x, a)$ represents the expected discounted reward of choosing the action $a \in \mathcal{A}$ over the state $x \in \mathcal{X}$ and then following the policy π. Between Q^π and V^π holds the following:

$$V^\pi(x) = \int_{\mathcal{A}} Q^\pi(x, a)\pi(a|x)\mathrm{d}a. \tag{2.10}$$

It is possible to write Q^π in a recursive way, like we did for V^π:

$$\begin{aligned} Q^\pi(x, a) &= \mathop{\mathbb{E}}_{\pi}[v_t | x_t = x, a_t = a] \\ &= \mathop{\mathbb{E}}_{\pi}[r_t + \gamma v_{t+1} | x_t = x, a_t = a] \\ &= \mathop{\mathbb{E}}_{\pi}[r_t + \gamma Q^\pi(x_{t+1}, a_{t+1}) | x_t = x, a_t = a] \\ &= R(x, a) + \gamma \int_{\mathcal{X}} P(y|x, a)V^\pi(y)\mathrm{d}y \\ &= R(x, a) + \gamma \int_{\mathcal{X}} P(y|x, a) \int_{\mathcal{A}} \pi(a'|y)Q^\pi(y, a')\mathrm{d}a'\mathrm{d}y. \end{aligned} \tag{2.11}$$

Note that if you would like to compute the state-value function in the case that the action space and the state space are discrete, we could rewrite 2.9 as a product of linear matrices where V^π and R^π are column vectors of dimension $|\mathcal{X}|$, while P^π is a square matrix with $|\mathcal{X}|$ columns and rows:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi \tag{2.12}$$

from which derives straight-forward the following solution:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi \tag{2.13}$$

This solution is not very appealing: it requires in fact a model of the MDP, which is however very often unknown and it needs $\gamma < 1$ so that $I - \gamma P^\pi$ is invertible. We will see that V^π and Q^π are very useful not only because they permit to evaluate the performance of a policy over a state, but also because through them we can compute $V^* \in \mathcal{V}$ and $Q^* \in \mathcal{Q}$ that are the state function and respectively the action-state value function computed with the *optimal policy* π^*. In fact the state-value function defines a partial ordering over policies [73]:

$$V^\pi(x) \leq V^{\pi'}(x) \quad \text{with} \quad x \in \mathcal{X} \implies \pi \leq \pi'. \tag{2.14}$$

For any MDP there exists an optimal policy π^* which is better or equal to any other policy π, furthermore all the optimal policies achieve the optimal state-value function and action-state value function:

$$V^{\pi^*} = V^* \tag{2.15}$$

$$Q^{\pi^*} = Q^*. \tag{2.16}$$

We define a *greedy* policy π with respect to Q as $\pi(x) \in \arg\max_{a\in\mathcal{A}} Q(x,a)$, and we can state that $\pi^*(x) \in \arg\max_{a\in\mathcal{A}} Q^*(x,a)$. One of the most interesting properties of MDPs is that there always exists an optimal policy that is deterministic [73]. This restricts a lot the space search of the optimal policy: we can just focus on deterministic policies. To give an intuitive idea on why there always will be a deterministic optimal policy that solves an MDP, we can easily see that if there is an optimal policy it has to be deterministic in those states where the action-state value function assumes different values. On the contrary it is possible that a policy is stochastic on a state where the respective action-state value function evaluated in such state assumes the same value for all different actions: but in this case also the deterministic choice to choose one of those actions will always be an optimal choice. We can also observe that V^* and Q^* satisfy the following Optimality Bellman equations:

$$V^*(x) = \max_{a\in\mathcal{A}} R(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a)V^*(y)\mathrm{d}y \tag{2.17}$$

$$Q^*(x,a) = R(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a) \max_{a'\in\mathcal{A}} Q^*(y,a')\mathrm{d}y. \tag{2.18}$$

Furthermore, we can notice from equations (2.17) and (2.18) that $|V^*|$ and $|Q^*|$ are bounded by $\frac{R_{\max}}{1-\gamma}$, we can show this by contradiction:

Proof. Let us suppose by absurd that there exists $\overline{x}$, $\overline{a}$ and $\varepsilon > 0$ such that $Q^*(x,a) \geq \frac{R_{\max}+\varepsilon}{1-\gamma}$ then for any x, a holds:

$$\begin{aligned} Q^*(x,a) &= R(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a) \max_{a'\in\mathcal{A}} Q^*(y,a')\mathrm{d}y & (2.19) \\ &\leq R_{\max} + \gamma \frac{R_{\max}+\varepsilon}{1-\gamma} & (2.20) \\ &= \frac{R_{\max}+\gamma\varepsilon}{1-\gamma}, & (2.21) \end{aligned}$$

which is clearly a contradiction (we always assume $\gamma < 1$). □

Bellman Operators

Bellman Operators are the main core of all value-based RL techniques. Their properties permit to develop algorithms that infer the optimal policy with nice convergence properties. Bellman Operators are unary operators and they can be defined in $\mathcal{V}$ or $\mathcal{Q}$ with similar meanings.

Definition 1. Bellman Operator in $\mathcal{Q}$
Let $(\mathcal{Q}, d)$ be a non-empty complete metric space with a operator $T^\pi : \mathcal{Q} \to \mathcal{Q}$.

$$(T^\pi Q^\pi)(x,a) = R(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a) \int_{\mathcal{A}} \pi(a'|y) Q^\pi(y,a') \mathrm{d}a' \mathrm{d}y$$

.

Definition 2. Bellman Optimality Operator in $\mathcal{Q}$
Let $(\mathcal{Q}, d)$ be a non-empty complete metric space with a operator $T^* : \mathcal{Q} \to \mathcal{Q}$.
$(T^*Q)(x,a) = R(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a) \max_{a' \in \mathcal{A}} Q(y,a') \mathrm{d}y$.

The Bellman operator in $\mathcal{Q}$ (Definition 1) and the Bellman Optimality operator in $\mathcal{Q}$ (Definition 2) are respectively a compact form of the equations (2.11) (2.18), and equivalent operators in $\mathcal{V}$ could be argued for V^π and V^* by the equations (2.9) and (2.17).

We will show now the properties of the Optimality Bellman Operator in $\mathcal{Q}$, that actually applies also for all the Bellman Operators previously defined:

Property 1. Monotonicity
Given $Q_1, Q_2 \in \mathcal{Q}$ the operator T^* is said to be monotone*:*

$$Q_1 \leq Q_2 \implies T^*Q_1 \leq T^*Q_2$$

.

Property 2. Max-Norm Contraction
Given $Q_1, Q_2 \in \mathcal{Q}$ the operator T^* is said to be a Max-Norm Contraction*:*

$$\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

.

Proof. We show the proof in the case of the Bellman Optimality operator T^* in $\mathcal{Q}$:

$$\begin{aligned}
& \|T^*Q_1 - T^*Q_2\|_\infty \\
= \; & \max_{x\in\mathcal{X},a\in\mathcal{A}} R(x,a) + \gamma \max_{a_1\in\mathcal{A}} \int_{\mathcal{X}} P(y|x,a)Q_1(y,a_1)\mathrm{d}y \\
& -R(x,a) - \gamma \max_{a_2\in\mathcal{A}} \int_{\mathcal{X}} P(y|x,a)Q_2(y,a_2)\mathrm{d}y \\
= \; & \gamma \max_{x\in\mathcal{X},a\in\mathcal{A}} \max_{a_1\in\mathcal{A}} \int_{\mathcal{X}} P(y|x,a)Q_1(y,a_1)\mathrm{d}y \\
& - \max_{a_2\in\mathcal{A}} \int_{\mathcal{X}} P(y|x,a)Q_2(y,a_2)\mathrm{d}y \\
\leq \; & \gamma \max_{x\in\mathcal{X},a\in\mathcal{A},a_1\in\mathcal{A}} \int_{\mathcal{X}} P(y|x,a)\left(Q_1(y,a_1) - Q_2(y,a_1)\right)\mathrm{d}y \\
\leq \; & \gamma \max_{x\in\mathcal{X},y\in\mathcal{X},a\in\mathcal{A},a_1\in\mathcal{A}} Q_1(y,a_1) - Q_2(y,a_1) \qquad (2.22) \\
= \; & \gamma\|Q_1 - Q_2\|_\infty. \qquad (2.23)
\end{aligned}$$

□

The inequality (2.22) holds keeping in account that $P(y|x,a)$ is a probability function, and therefore $\int_{\mathcal{X}} P(y|x,a)\left(Q_1(y,a_1) - Q_2(y,a_1)\right)\mathrm{d}y$ is equivalent to the weighted average of $Q_1(y,a_1) - Q_2(y,a_1)$ and obviously the average of some values can always be upper-bounded by the max of those values.

Theorem 1. Banach Fixed Point
If T^* is a contraction mapping, then it admits a unique fixed-point Q^* in $\mathcal{Q}$. Furthermore, Q^* can be found as follows: start with an arbitrary element $Q_0 \in \mathcal{Q}$ and define a sequence $\{Q_n\}$ by $Q_n = T^*(Q_{n-1})$, then $Q_n \to Q^*$ [38].

What Theorem 1 states is, in more practical words, that if the operator T^* is a contraction, then we can choose from a random point $Q \in T^*$ and apply iteratively the operator T^* to end up in a point (close as desired) to its fixed point Q^*.

Property 3. Unique Fixed Point
As stated in Theorem 1 the operator T^* has a unique fixed point *if $\exists! Q \in \mathcal{Q}$:*

$$T^*Q = Q$$

and

- *the fixed point of T^{π} defined in $\mathcal{V}$ is V^{π};*
- *the fixed point of T^{*} defined in $\mathcal{V}$ is V^{*};*
- *The fixed point of T^{π} defined in $\mathcal{Q}$ is Q^{π};*
- *The fixed point of T^{*} defined in $\mathcal{Q}$ is Q^{*}.*

Each point stated in Property 3 comes respectively from the equations (2.9),(2.17), (2.11) and (2.18) in conjunction with Theorem 1 which ensures that those points are unique.

The Bellman Optimality Equations are non-linear thus they can not be solved in closed form. There are a number of ways to solve them, and we can actually divide such ways in three different branches: *Dynamic Programming*, *Linear Programming* and RL. Dynamic Programming requires the full knowledge of the model in order to solve the MDP, and it is the starting point for developing the theory of RL. That is why we will introduce it in the next section.

2.1.2 Dynamic Programming

Dynamic Programming (DP) [37] [6] is a very broad and general technique to solve complex mathematical problems by breaking down the problem in sub-problems, solving each sub-problem and then recombining the partial solutions. DP can be applied to those problems that satisfy the followings properties:

1. *Optimal Substructure*: the problem could be divided in sub-problems, and the combination of the optimal solutions of such problems is the optimal solution of the whole problem.
2. *Overlapping Sub-problems*: the same sub-problem recurs many times, so we can cache its solution, and reuse whenever it is required. In this way we can speed up the computation.

DP is used for a number of different problems like scheduling algorithms, string algorithms (e.g., sequence alignment), graph algorithms (e.g., shortest path algorithms), graphical models (e.g., Viterbi algorithm), bioinformatics (e.g., lattice models), just to mention a few. DP applies also to MDPs, in fact we can observe that Bellman Equations are recurrent, and the value functions are a way to keep track of the partial solution found.

DP in the case of MDPs assumes the full knowledge of the model, and it can solve two different kinds of problem:

1. *Prediction*: where the input is an MDP $< \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu >$, a policy π and the output is the state value function V^π.
2. *Control*: where the input is an MDP $< \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu >$ and the output is the optimal state value function V^* and the optimal policy π^*.

Policy Iteration

Policy Iteration (PI) [37] [73] is a method to solve MDPs that decomposes the control problem in two different stages: a stage of policy evaluation and a stage of policy improvement.

The policy evaluation is a prediction problem where we compute the value function from an MDP and a policy. To solve this problem we could iterate the Bellman Operator for V^π from a value function V_0, and then $V_{k+1} = T^\pi V_k$, so that, after enough iterations we get $V_k \approx V^\pi$

The policy improvement stage generates a better policy. We can state that:

Theorem 2. Policy Improvement
Let π and π' be a pair of deterministic policies such that:

$$Q^\pi(x, \pi'(x)) \geq V^\pi(x) \quad , \quad \forall x \in \mathcal{X}$$

then

$$V^{\pi'}(x) \geq V^\pi(x)$$

.

Proof.

$$\begin{aligned}
V^\pi(x) &\leq Q^\pi(x, \pi'(x)) = \mathop{\mathbb{E}}_{\pi'}[r_{t+1} + \gamma V^\pi(x_{t+1}) | s_t = s] \\
&\leq \mathop{\mathbb{E}}_{\pi'}\left[r_{t+1} + \gamma Q^\pi(x_{t+1}, \pi'(x_{t+1})) | x_t = x\right] \\
&\leq \mathop{\mathbb{E}}_{\pi'}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(x_{t+2}, \pi'(x_{t+2})) | x_t = x\right] \\
&\leq \mathop{\mathbb{E}}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \ldots | x_t = x] = V^{\pi'}(x). \qquad (2.24)
\end{aligned}$$

□

This allows us to write the policy improvement step, that is a *greedy*-policy with respect to Q:

```
Algorithm 1: Policy Iteration
Data: $\mathcal{R}, P$
Result: $\pi$
$k \leftarrow 0$;
initialize $\pi_0$;
$V_0 \leftarrow 0$ ;
for $i = \{1 \dots N\}$ do
    $V_t(x) \leftarrow V^{\pi_{t-1}}(x)$;
    $\pi_t(x) \leftarrow \arg\max_{a \in \mathcal{A}} R(x,a) + \gamma\, \mathbb{E}_{x' \sim P(\cdot|x,a)} [V_t(x')]$;
end
Return $\pi = \pi_N$;
```

$$\pi'(x) = \arg\max_{a \in \mathcal{A}} Q(x,a) = \arg\max_{a \in \mathcal{A}} R^{\pi}(x,a) + \gamma \int_{\mathcal{X}} P(y|x,a)V(y)\mathrm{d}y. \tag{2.25}$$

The PI algorithm so basically iterates a step of policy evaluation and a step of policy improvement:

$$\pi_0 \xrightarrow{\text{PE}} V^{\pi_0} \xrightarrow{\text{PI}} \pi_1 \xrightarrow{\text{PE}} V^{\pi_1} \dots \xrightarrow{\text{PE}} V^*.$$

When we perform the policy evaluation step, a question automatically arises: how many iterations do we have to iterate in order to have a good evaluation of V^{π}? We could stop the number of iteration when $\|V_{k+1} - V_k\|_{\infty}$ becomes lower than a certain threshold, however is this procedure really necessary? Do we really have a good approximation of V^{π}?

Value Iteration

Value Iteration (VI) [5] instead of explicitly representing the policy, it exploits the Bellman Optimality operator, in order to achieve a faster convergence. In fact, at every iteration, the VI algorithm performs the improvement of the value function, without loosing time on estimating the value function associated to an intermediate policy. The core of a Value Iteration algorithm can be represented by an equation $V_{k+1} = T^* V_k$. It will produce a sequence like the following one:

$$V_0 \xrightarrow{\text{VI}} V_1 \xrightarrow{\text{VI}} \dots \xrightarrow{\text{VI}} V^*.$$

VI converges to the optimal V^* thanks to the previously described properties of the Bellman Optimal Operator:

Algorithm 2: Value Iteration

```
Data: $\mathcal{R}, P$
Result: $\pi$
$k \leftarrow 0$;
$V_0 \leftarrow 0$ ;
for $i = \{1 \dots N\}$ do
    $V_t \leftarrow T^* V_{t-1}$;
end
Define as $\pi(x) \leftarrow \arg\max_{a \in \mathcal{A}} R(x,a) + \gamma\, \mathbb{E}_{x' \sim P(\cdot|x,a)} [V_N(x')]$
```

Theorem 3. VI Convergence 1
VI converges to the optimal state-value function V^*:

$$\lim_{k \to \infty} V_k = V^*.$$

Proof.

$$\|V_k - V^*\|_\infty = \|T^* V_{k-1} - T^* V^*\|_\infty \leq \gamma \|V_{k-1} - V^*\|_\infty \leq \dots \leq \gamma^k \|V_0 - V^*\|_\infty. \tag{2.26}$$

□

Theorem 3 shows that VI converges always (since $\gamma < 1$), with a faster convergence rate for small γ.

Theorem 4. VI Convergence 2 [7]
If $\|V_k - V_{k-1}\|_\infty \leq \epsilon$ then $\|V_k - V^*\|_\infty \leq \frac{2\epsilon\gamma}{1-\gamma}$.

Theorem 4 shows instead the convergence rate from the Bellman residual perspective. We know intuitively, from the max-norm contraction property of the optimality Bellman opertor, that if the magnitude of $V_k - V_{k-1}$ is high, then we probably are far away from V^*, and logically when the magnitude of $V_k - V_{k-1}$ is small V_k should be close to V^*. We can see from Theorem 4 that the error $\|V_k - V^*\|_\infty$ can be bounded by $\frac{2\epsilon\gamma}{1-\gamma}$ which suggests that V_k is close to V^* when ϵ or γ are close to zero. The term $\frac{\gamma}{1-\gamma}$ could be seen in fact as a rescaling of the distance between value functions: two value functions that can be considered close in a model where γ is low, could be considered far in a MDP where γ is higher.

Approximate Dynamic Programming

We have assumed state space and action space to be continuous measurable sets. This means that in order to represent functions over such spaces we need to resort to some function approximation technique. In RL this goes under the name of Approximated DP (ADP). ADP is sometimes necessary even when the spaces are discrete but highly dimensional. In this scenario we cannot exploit fully informed tabular representation since the update cost is polynomial in the number of states and actions. This approximator could be seen as an Operator that projects the value function in its functional space. In the case of the state value function we define $\Pi : \mathcal{B}(\mathcal{X}) \rightarrow \mathcal{F}$ where $\mathcal{F}$ is the new functional space.

Norm We define the norm operator: $\|g\|_{p,\mu} = [\int g(x)^p \mathrm{d}\mu(x)]^{1/p}$ in the case that μ is a distribution of probability, or $\|g\|_{p,\mu} = [\sum_{x \in \mu} g(x)^p]^{1/p}$ in the case μ is a set. Note that if $\hat{\mu} = \{z_i \sim \mu\}_{i=1\dots N}$ then $\mathbb{E}\left[\|f\|_{p,\hat{\mu}}\right] = \|f\|_{p,\mu}$.

The operator works by minimizing the p-norm between the function f that belongs to the functional space $\mathcal{F}$ and the target y that is the true state value function:

$$\Pi V = \arg\min_{f \in \mathcal{F}} \|f - y\|_{p,\mu}, \quad y \in \mathcal{B}(\mathcal{X}) \tag{2.27}$$

or very similarly, in the case of action-state value function, we define the approximator $\Pi : \mathcal{B}(\mathcal{X} \times \mathcal{A}) \rightarrow \mathcal{F}$:

$$\Pi Q = \arg\min_{f \in \mathcal{F}} \|f - y\|_{p,\mu}, \quad y \in \mathcal{B}(\mathcal{X} \times \mathcal{A}). \tag{2.28}$$

Asynchronous Dynamic Programming

Until this point we have described *Synchronous DP* in the sense that when we wanted to apply a Bellman Operator to solve a control or a prediction problem we needed two value functions, let's suppose V_k and V_{k-1}, and we compute V_k by applying the Bellman Operator on V_{k-1}, for example in AVI:

$$V_k = T^* V_{k-1}. \tag{2.29}$$

This way to perform computation is called "synchronous" because all states could be updated in parallel:

$$V_k(x) = (T^* V_{k-1})(x). \tag{2.30}$$

We could speed up the computation by applying the Bellman Operator "in place". We keep only one value function V: any time we apply the Bellman operator only for one state at time, and we replace the computed value with the previous value, so the next time we compute an update for another state, the value function will be already improved. This concept is known as *bootstrap*, because it is a kind of speculation: we try to update the value-function as soon as possible. Asynchronous DP is guaranteed to converge if we continue to update all the states. There are mainly two ideas on how to select the states to update:

1. *Prioritized Sweeping*: we compute the Bellman Error $\|(TV)(s) - V(s)\|_\infty$, and we update first the states with the highest Bellman Error.
2. *Real-Time Dynamic Programming*: we can also update the state based on a sampling of the trajectory (i.e., simulating the environment), so that we update mainly the states that are relevant to the agent[6].

Full vs Sample Backup

The methods shown in the previous section describe a way to make DP more efficient with the usage of approximation, or the usage of asynchronous backups. Both the methods can reduce the memory usage and the computations, but still they do not solve the curse of dimensionality when we perform a backup: in case of a high number of states (as we mentioned earlier in the case of discrete states, the number of states grows exponentially with the number of variables). So each time we must perform an update (or backup) we must solve an integral or a very long summation (in all the Bellman operators we have $\int_\mathcal{X}$ or $\sum_\mathcal{X}$). A way to solve also this problem is to not use the reward function or the transition probability, but just a sample of them. For example, let simulate the environment starting from $x, a = \pi(a)$ and observe r, x' where r is the reward gained and x' is the next state reached. We can update the value of state x as follows:

$$V(x) \leftarrow r + \gamma V(x') \tag{2.31}$$

in order to estimate V^π. In this way the cost of a backup is constant independently of the complexity of the underlying MDP, furthermore, in this case, no knowledge of the underlying MDP is required [6].

Approximate Value Iteration

We would like in this section to give a special focus on Approximate Value Iteration (AVI) [63], the branch of algorithms on which is based this work. AVI is an ADP method for VI. The main core of the algorithm is to approximate the Optimality Bellman operator with the projection operator $\Pi : \mathcal{V} \to \mathcal{F}$:

$$V_k = \Pi T^* V_{k-1}. \tag{2.32}$$

Thus $V_k = \arg\min_{f \in \mathcal{F}} \|T^* V_{k-1} - f\|_{p,\mu}$. We know from the theory that we could start from a random V_0 and apply iteratively the Optimality Bellman operator. The same approach is exploited in AVI but the application of the Optimal Bellman operator $T^* V_{k-1}$ is projected onto the functional space $\mathcal{F}$.

In general, this algorithm does not converge, but we can analyze the asymptotic behavior. Let us suppose that the *approximation error* is bounded $\|\varepsilon_k\|_\infty \leq \varepsilon$ where the approximation error is defined as:

$$\varepsilon_k = T^* V_k - V_{k+1} \tag{2.33}$$

then, a bound on the error between the state value function of the policy π_k greedy w.r.t. V_k and the optimal state value function V^* is [7]:

$$\limsup_{k \to \infty} \|V^* - V^{\pi_k}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2}\varepsilon. \tag{2.34}$$

Fitted Value Iteration Actually we do not always have the Optimality Bellman operator, because we do not have full knowledge of the model, thus the only way is to use the approximate Optimality Bellman operator $\tilde{T}^*$. We can construct $\tilde{T}^*$ by sampling $r(x,a) \sim \mathcal{R}(x,a)$ and $x' \sim P(x,a)$ and then:

$$(\tilde{T}^* Q)(x,a) = r(x,a) + \gamma \max_{a' \in A} Q(x', a'). \tag{2.35}$$

We defined the approximated Optimality Bellman operator $\tilde{T}^*$ with the action-value function, because in absence of the model, it is the only way to derive the greedy policy. Several approaches based on the AVI-scheme have been presented in literature, some examples are tree-based Fitted Q-Iteration [20], multilayer perceptron-based Fitted Q-Iteration [66], regularized Fitted Q-Iteration [22] and Fitted Q-iteration by Advantage Weighted Regression [56]. All these approaches belong to the sampling-base fitted value iteration (FVI) class. The main issue of FVI is that it may diverge based on the approximator used [2, 8, 75, 32]. It is shown that the class of averager approximators can be used safely, and no issue has been reported with the usage of non-parametric regressor [58, 20, 66, 61].

2.1.3 Reinforcement Learning

In Section 2.1.2 we studied the problem of solving MDPs with the knowledge of the model (so knowing $\mathcal{R}$ and P). It is not always possible to have $\mathcal{R}$ and P, and even if it is, as we stated in Section 6, a sampling based method can be used to lower down the computation, skipping integrals or long summations. There are also techniques that do not make usage of either the model or a value function, but that are just based on improving the policy. We can summarize RL with the following categorization:

1. *Model-Free vs Model-Based*: For Model-Free are intended all those techniques that do not require the knowledge of $\mathcal{R}$ or P, on the contrary the Model-Based techniques estimate a parametric model of $\mathcal{R}$ and P.

2. *On-Policy vs Off-Policy*: An On-Policy algorithm will try to estimate the value function V^π by sampling from π (like SARSA), while an Off-Policy algorithm will estimate V^π using a different policy π' (like Q-Learning).

3. *Online vs Offline*: This concept is very analogous to the full vs sample backup already seen in sub-subsection 6. Online algorithm (like Q-Learning) try to estimate the value function by sampling from the environment, while Offline methods (like FQI) only use an already collected datasets to find-out the optimal value-function. The advantage of the Online methods is that they can explore and sample the most interesting states, and learn “actively” on the environment, the drawback is that they are sensible to the “exploration-exploitation” tradeoff. The Offline methods are in general applicable on easier problems where a random dataset (a dataset sampled with a random policy) is enough to learn, the advantage of this solution is that is more stable because it does not involve exploration. There is also a half-way between Online and Offline: the batch-sampling techniques (like DQN) where the agent collects a dataset “online”, but it approximates the value-function by sampling batches from this dataset.

4. *Tabular vs Function Approximation*: almost all the techniques that involve discrete MDPs can work with Tabular techniques: we can store the values of the states in a matrix, but in practice these techniques are also used in very simple MDPs where the number of variables involved for the representation of a state is very small. In real applications we will always use Function Approximation, that is more problematic (it

might introduce errors, instability, observability problems, ..) but it is the only way to deal with complex domains.

5. *Value-Based vs Policy-Based vs Actor-Critic*: all the techniques that are based on the value function are called Value-Based. In Value-Based settings we must require that the optimal policy is deterministic, and that the action-space is discrete, although could be difficult to derive the policy from the value-function. Policy-Based techniques make usage of a model of the policy, and they try to maximize the policy performance. They can work also with stochastic policies - making Policy-Based also effective in Partial Observable MDPS, and they have often nice convergency properties. Pure Policy-Based algorithms work only with Monte-Carlo sampling, and this (as we will see later) will introduce a lot of variance. Actor-Critic techniques are a good way to combine the pros of the two kind of algorithms.

Monte-Carlo (in Policy Evaluation)

Monte-Carlo (MC) [73] is a technique that basically samples from the episode and compute the value function by $V(x) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t | x_t = x\right]$. It is a Model-Free technique because it does not require knowledge of R or P; it learns from full episodes, thus it does not bootstrap; and it requires strictly episodic MDPs (all episodes must terminate). Monte-Carlo could be very appealing for its simplicity and it also shows a kind of robustness with non Markovian settings because it does not make assumption of Markovianity. The main drawback is that $\mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t\right]$ has a high variance because it involves the summation of stochastic variables.

Temporal-Difference (in Policy Evaluation)

Temporal Difference (TD) [73] techniques are a collection of Online methods that use the so-called TD-error $r_t + V(x_{t+1}) - V(x_t)$ to update the value function:

$$V(x_t) = V(x_t) + \alpha(r_t + \gamma V(x_{t+1}) - V(x_t)) \tag{2.36}$$

where α is also known as learning rate. It is possible to show that if we uses a dynamic value for α such that $\sum_i \alpha_i = \infty$ and $\sum_i \alpha_i^2 < \infty$ then the equation (2.36) could be expressed as

$$V(x_t) = \mathop{\mathbb{E}}_{\pi} \left[r_t + \gamma V(x_{t+1})\right] = \mathop{\mathbb{E}}_{\pi} \left[R(x_t, \pi(x_t)) + \gamma V(x_{t+1})\right]. \tag{2.37}$$

TD algorithm makes the usage of bootstrap, in fact it backups during the episode. While $\mathbb{E}_{\pi}\left[r_t + \gamma V(x_{t+1})\right]$ suffers less from high variance than MC, unlike MC it has bias, because $r_t + \gamma V(x_{t+1})$ it is an estimation of the expected return, while v_t it is exactly the expected return. A good way to mix MC and TD is to use Eligibility Traces.

SARSA

We saw examples of MC and TD for prediction. SARSA [73] is a TD algorithm for control and its name derives from the fact that it uses samples of shape $< s, a, r, s', a' >$ (where s and s' are the current state and the next state; r, a, a' have always the usual meaning of observed reward and chosen action both for a and a'. SARSA uses action-state value function Q instead of state value function V because in control problems Q contains implicitly also the policy by the following relation:

$$\pi(x) = \arg\max_{a \in \mathcal{A}} Q(x, a). \tag{2.38}$$

SARSA during the running of an episode collects tuples of kind $< x, a, r, x', a' >$ where $x = x_t$, $a = \pi(x_t), x' = x_{t+1}, a' = \pi(x_{t+1})$ and r is the observed reward of choosing a in the state x. SARSA uses the following update:

$$Q(x, a) = Q(x, a) + \alpha(r + \gamma Q(x', a') - Q(x, a)) \tag{2.39}$$

where again α is the learning rate and $r + \gamma Q(x', a') - Q(x, a)$ is the TD-error. The reason we say that SARSA is on-policy is that a' is the action that we choose for the next sampling x', a', r', x'', a''. So far it seems to be a policy evaluation problem: this is not true, because we are not using here a fixed policy, but our policy is defined randomly with probability ϵ and

$$\pi(x) = \arg\max_{a \in \mathcal{A}} Q(x, a) \tag{2.40}$$

with probability $1 - \epsilon$. The policy described is called ϵ-greedy policy. The epsilon parameter is an exploration parameter that allows the policy to visit new states. The ϵ parameter should move to 0, so that SARSA will evaluate the greedy policy, converging so to the optimal one.

Q-learning

SARSA is a very nice algorithm but it is not able to converge to the optimal policy while following a sub-optimal policy. Q-Learning [73] is an Off-Policy version of SARSA thus it is able to work even if it samples with a random policy (provided that a random policy is able to visit all the states that are required in the optimal trajectory). This is because instead of using the action a' chosen by the policy, it always uses $\max_{a' \in \mathcal{A}} Q(x', a')$:

$$Q(x, a) = Q(x, a) + \alpha(r + \max_{a' \in \mathcal{A}} \gamma Q(x', a') - Q(x, a)). \tag{2.41}$$

This time the TD-error is $r + \max_{a' \in \mathcal{A}} \gamma Q(x', a') - Q(x, a)$.

Fitted Q-Iteration

Fitted Q-Iteration [20] [66] is an Off-Policy and Offline AVI method. The core of the algorithm, like in the case of Q-Learning or SARSA is to evaluate the optimal Q-value function Q^*. In ideal case, where we have perfect knowledge of the MDP, we could iterate the following:

$$Q_k = T^* Q_{k-1}. \tag{2.42}$$

As we have seen in AVI methods, if we can not use a tabular approach (i.e. in all real applications), we must use an approximator $\Pi : (\mathcal{X} \times \mathcal{A} \to \mathbb{R}) \to \mathcal{F}$ that projects $T^* Q_{k-1}$ onto a functional space $\mathcal{F} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ (2.42) by:

$$Q_k = \Pi T^* Q_{k-1}. \tag{2.43}$$

The projection operator Π normally works by finding the function $f \in \mathcal{F}$ that minimizes the error between such function and a given function g that is the target function to approximate:

$$\Pi(g) = \arg \min_{f \in \mathcal{F}} \|g - f\|^p_{p,\mu} \tag{2.44}$$

(with $\| \cdot \|_{p,\mu}$ as already described in Section 6). In the case of FQI, what we minimize in practice is the Projected Bellman Error (PBE):

$$\Pi(T^* Q_{k-1}) = \arg \min_{f \in \mathcal{F}} \|T^* Q_{k-1} - f\|^p_{p,\mu}. \tag{2.45}$$

We assumed so far to have perfect knowledge of the model, in fact we are using the Optimality Bellman operator. This is not possible in most of the cases, and what we do in practice is to use an approximation of the Optimality Bellman operator $\hat{T}^*_\mu$. We first have to collect a dataset $\mu = \{< x_i, a_i, r_i, x'_i >\}_{i=1...N}$, often sampled from a random policy. The collection is

Algorithm 3: Fitted Q-Iteration

```
Data: $\mu \leftarrow \{x_i, a_i, r_i, x'_i, t_i\}_{i=1 \dots N}$
Result: $\pi$
$k \leftarrow 0$;
$Q_0 \leftarrow 0$ ;
do
    $k \leftarrow k + 1$ ;
    for $i = \{1 \dots N\}$ do
        if $t_i$ is terminal then
            $y_i \leftarrow r_i$;
        else
            $y_i \leftarrow r_i + \gamma \max_a Q_k(x'_i, a)$;
        end
    end
    $Q_k \leftarrow \arg\min_{f \in \mathcal{F}} \|f(\mathbf{x}, \mathbf{a}) - \mathbf{y}\|_2^2$;
while $\|Q_k - Q_{k-1}\|_{\mu,\infty} > \varepsilon$;
Define as $\pi(x) = \arg\max_a(x, a)$
```

fixed and it never changes during the computations of the algorithm. Then we can define $\hat{T}^*_\mu$ as:

$$(\hat{T}^*_\mu Q_{k-1})(x_i, a_i) = r_i + \gamma \max_{a \in \mathcal{A}} Q_{k-1}(x'_i, a) \tag{2.46}$$

where $(x_i, a_i, r_i, x'_i) \in \mu$.

So we can eventually write the core of FQI:

$$Q_k = \Pi \hat{T}^*_\mu Q_{k-1}. \tag{2.47}$$

2.2 Boosting

Boosting was introduced to answer to a question posed by Kearns and Valiant[42] [43]: can a set of *weak learners* create a single strong learner? With weak learner we refer to a learner that is able to perform slightly better than a random guess. The answer is affirmative and it has been proved by Robert Schapire in 1990 [67]. Boosting is a meta-algorithm that (linearly) combines weak learners in a final strong classifier (or regressor). The most famous boosting algorithm is for sure *AdaBoost* [25] which won in 2003 the ambitious Gödel Prize[3]. AdaBoost became so famous for its resistance to over-fitting both for classification and regression problems. Boosting is often compared to Bagging (Boosting Aggregation) [9], which works similarly to Boosting, producing a number of learners and then combining them, but with a great difference: all the learners could learn in parallel, and they do not depend from each other. This is actually not a small difference, because the goal of bagging is to reduce the variance (enhancing the generalization) - thus using complex models, fitting them with a random re-sampling of the dataset, and then averaging them; while boosting normally works with weak (simple) learners and each one depends from the previous one, and then by recombining them linearly enhancing their ability to well fit the dataset. We will see in section 2.2.1 that in order to have a good model, we need to have both a good fitting performance and a good generalization: these two properties are somehow in contradiction, and they must be balanced appropriately. When a model fits too well the data then it learns the dataset used to train it (and the noise associated to the data) and it will be not able to generalize, while if it fits too poorly it simply means that the model is not able to learn the dataset, and thus neither to predict new data. Boosting in general enhance the fitting abilities of the learner, but it is also able to approach "gently" a good generalization.

2.2.1 General Supervised Machine Problem

With Supervised Machine Learning (SML) [52] we generally refer to all those techniques that try to predict a phenomenon by observing it. The phenomenon is often stochastic even if it could be explained in a deterministic way, but the observations are sampled from a noisy source. In general, SML (for regression problem) deserves to find a function g that describes a phenomenon f by means of a finite number of observations which are sampled with an intrinsic source of noise. More formally, we have an input space X that defines our set of possible inputs from which we believe it is possible to predict the *outcomes* belonging to a target space Y, and the phenomenon that we would like to observe and predict could be described by $f \in F$ where $F = \{f | f : X \to Y\}$ with a function $g \in G$ where $G \subset F$. To achieve

[3] The Gödel Prize is an annual prize for the most oustanding papers in the field of theoretical computer science. The prize is assigned jointly by European Association for Theoretical Computer Science (EATCS) and the Association for Computing Machinery Special Interest Group on Algorithms and Computational Theory (ACM SIGACT). The name of the prize is in honour of Kurt Gödel, who was the first to propose the question *P versus NP* which is still one of the most interesting opened question in the area of theoretical computer science.

this goal we use a dataset $D = \{(x_i, y_i)|x_i \sim \mathcal{X}, y_i \sim \mathcal{Y}(x_i)\}_{i=1...N}$ where $\mathcal{X} \in \mathcal{M}(X)$ and $\mathcal{Y} : X \to \mathcal{M}(Y)$. We assume that the generative model could be defined as $\mathcal{Y}(x) = f(x) + \mathcal{E}(x)$ where $f(x)$ is the deterministic explanation and $\mathcal{E}(x)$ is a source of noise with zero mean and $\sigma^2(x)$ variance thus $\mathrm{Var}(Y) = \sigma^2$ and $\mathbb{E}[Y(x)] = f(x)$. To derive g we use a stochastic process $\mathcal{G} : D \to \mathcal{M}(G)$ that infers g from the dataset D. Very intuitively the more complex is the functional space G from which g belongs, the more $\mathcal{G}$ will be stochastic, and of course a simpler G leads a less variable $\mathcal{G}$. We could think about a process that tries to fit a line in a cloud of points: probably the cloud of points could be fitted well by very few lines, and the outcomes of $\mathcal{G}$ will be all very similar. If G is instead very complex, then will be a lot of very different functions g that well fit this cloud of points. In the first case, we say that our model *over-fits* because it has a small variance but a high bias, while in the second case we say that our model *over-fits* because it has a small bias but a high variance. When a model over-fit, it learns too well how to fit the data in the dataset - thus it is too specific, while when it under-fit, it does not learn enough: in both the cases it will leads to a poor prediction. When the variance and the bias are in some way well balanced so that the model does not have a high variance nor a high bias, the model probably will have learned enough from the data but still maintaining a kind of generality and thus a good prediction performance.

Bias-Variance Trade-off

What we explained in the previous section could be more formally expressed as Bias-Variance trade-off [39].

Definition 3. Variance
Let $x \sim \mathcal{X}$ where $\mathcal{X}$ is a stochastic variable, its variance it is defined as:

$$\begin{aligned}\mathrm{Var}[x] &= \mathbb{E}\left[(x - \mathbb{E}[x])^2\right] \\ &= \mathbb{E}\left[x^2 + \mathbb{E}[x]^2 - 2x\,\mathbb{E}[x]\right] \\ &= \mathbb{E}\left[x^2\right] + \mathbb{E}[x]^2 - \mathbb{E}\left[2x\,\mathbb{E}[x]\right] \\ &= \mathbb{E}\left[x^2\right] + \mathbb{E}[x]^2 - 2\,\mathbb{E}[x]^2 \\ &= \mathbb{E}\left[x^2\right] - \mathbb{E}[x]^2 . \end{aligned} \tag{2.48}$$

The variance indicates how much a phenomenon is variable, or in other words how much the samples extract from a distribution are distant in average from their mean value.

Definition 4. Bias
Let y be a target value and $x \sim \mathcal{X}$ where $\mathcal{X}$ is a stochastic variable, the bias of X on the target y is defined as:

$$\mathrm{Bias}[x, y] \quad = \quad \mathbb{E}\left[y - x\right]. \tag{2.49}$$

The bias shows in average the offset between x and the target value y. Very often we use $\mathrm{Bias}^2(x, y)$ to indicate the distance from the mean point $\overline{x} = \mathbb{E}_{x\sim\mathcal{X}}\left[X\right]$ to the target y. Now we would like to decompose the *mean square error* (MSE) of the model g to its target y in terms of bias and variance, omitting for the sake of compactness the observed input x:

$$\begin{aligned}
\mathbb{E}\left[(y-g)^2\right] &= \mathbb{E}\left[y^2 + g^2 - 2yg\right] \\
&= \mathbb{E}\left[y^2\right] + \mathbb{E}\left[g^2\right] - \mathbb{E}\left[2yg\right] \\
&= \mathrm{Var}[y] + \overline{y}^2 + \mathrm{Var}[g] + \mathbb{E}\left[g\right]^2 - 2\overline{y}\,\mathbb{E}\left[g\right] \\
&= \mathrm{Var}[y] + \mathrm{Var}[g] + \left(\overline{y} - \mathbb{E}\left[g\right]\right)^2 \\
&= \mathrm{Var}[y] + \mathrm{Var}[g] + \mathbb{E}\left[\overline{y} - g\right]^2 \\
&= \sigma^2 + \mathrm{Var}[g] + \mathrm{Bias}[\overline{y}, g]^2. \qquad (2.50)
\end{aligned}$$

Equation (2.50) shows that the error on predicting y with the model g is the summation of the intrinsic stochasticity of the phenomenon σ^2, the stochasticity of the model $\mathrm{Var}[g]$ and the square of the bias of the model g in predicting the target $\overline{y}$. We can do nothing about σ because it is a property of the phenomenon that we would like to predict, but we can instead reduce the variability associated to the model by using a simpler model or reducing the bias of the model by over-fitting: the two terms are thus in contrast and while we try to reduce one term, the other will increase. What we have to do is to find a combination that minimizes the sum of the two components.

2.2.2 Ensemble Learning and Boosting

Ensemble Learning [57] is a meta-algorithm that consists in generating a learner that is composed by a set of individual regressors where each of them is trained by re-sampling from the dataset. Boosting [13] is a particular case of ensemble learning, where each learner depends from the previous (the learning process can not be parallelized). More formally boosting is a general procedure that constructs the function g as a linear combination of

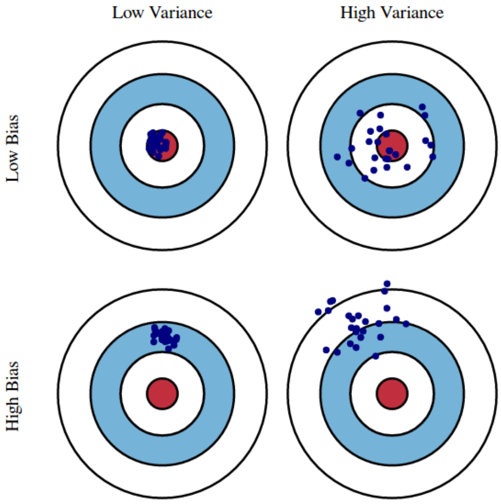

Figure 2.2: This picture shows the difference between variance and bias.

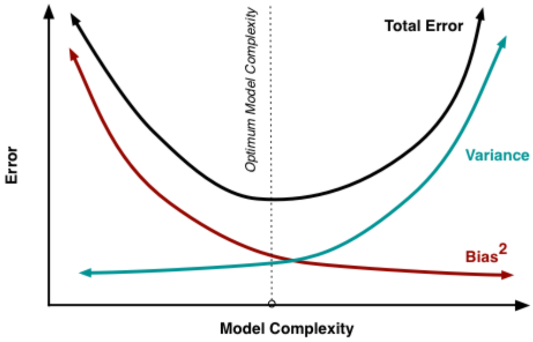

Figure 2.3: This picture shows how bias and variance vary in function of the complexity of the model.

many models g_i. To produce any of those g_i we use a base procedure $\mathcal{G}$ that depends not only on the dataset D but also on the previous models. We could further augment the dataset D with a weighting w_j that dynamically changes during each iteration. Lastly, we combine the function approximator by combining linearly all the intermediates estimates: $\sum_{j=1}^{M} \alpha_j g_j(\cdot)$. An example is the following:

$$\begin{aligned}
\{(w_{1,i}, x_i, y_i)\}_{i=1\ldots N} &\overset{\mathcal{G}}{\to} g_1(\cdot)\\
\{(w_{2,i}, x_i, y_i)\}_{i=1\ldots N} &\overset{\mathcal{G}}{\to} g_2(\cdot)\\
\{(w_{3,i}, x_i, y_i)\}_{i=1\ldots N} &\overset{\mathcal{G}}{\to} g_3(\cdot)\\
&\ldots\\
\{(w_{M,i}, x_i, y_i)\}_{i=1\ldots N} &\overset{\mathcal{G}}{\to} g_M(\cdot)\\
g(\cdot) &= \sum_{j=1}^{M} \alpha_j g_j(\cdot).
\end{aligned}$$

The above description of boosting is too general to be of any direct use. Different specifications of the weighting mechanism as well the specification of different linear coefficients are crucial, and define different ensemble schemes. In most of the cases boosting methods are special cases of sequential ensemble schemes, where the data weights depend on the results of the previous iteration. Very often the weights $w_{j,i}$ are chosen in such a way that a sample becomes more important when it is miss-predicted by the previous model.

2.2.3 Functional Gradient Descent

Breiman showed that AdaBoost can be represented as a steepest descent algorithm in function space which we call functional gradient [11] [10]. Friedman, Hastie and Tibshirani then developed a more general statistical framework which yields a direct interpretation of boosting as a method for function estimation [26]. They refer to “stage-wise additive modeling” where the word additive does not imply that the model boosting methods exploit their SML task by fitting a number of regressor and combining then linearly (i.e. summing up them).

Let us consider a loss function $\rho : Y^N \times Y^N \to \mathbb{R}$ which defines in our case a measure of the error between our model g evaluated in all the points x_i (which we denote by the column vector $\mathbf{x}$) and all the targets y_i (which we denote by the column vector $\mathbf{y}$). Let us consider a parametric function

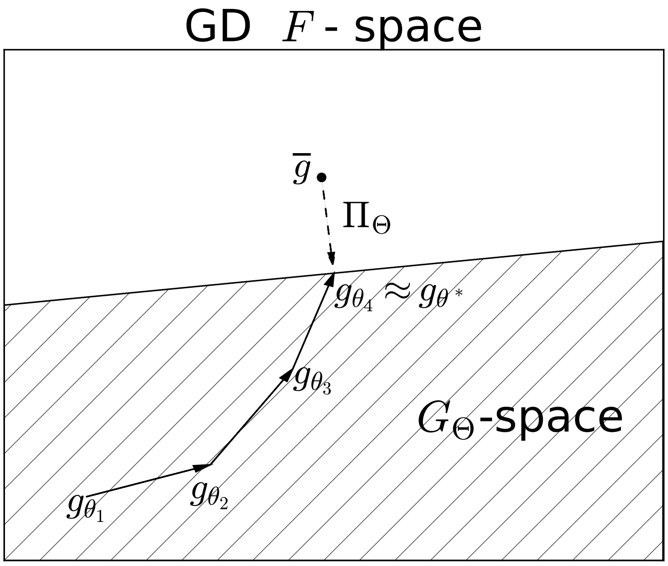

Figure 2.4: The pictures show the GD algorithm, starting from a configuration θ_1. Each arrow shows the gradient step, which brings from θ_i to θ_{i+1}. The space represented is the F space, but the gradient moves only in the G_Θ space: GD aims to recover the parameter θ^* that represents the projection in G_Θ of the true minimizer g of the loss ρ.

$g : \Theta \times X \to Y$ which for a fixed set of parameters $\theta \in \Theta$ could be denoted as $g_\theta \in G_\Theta$ where $G_\Theta \subset F = \{f | f : X \to Y\}$. Our goal is to find a projection of the function $\overline{g} \in F$ that minimizes (locally) the loss function ρ for a fixed $\mathbf{y}$ in the functional space G_Θ. A generic Gradient Descent (GD) [72] method starts from a random set of parameters θ_1, and then iterates the following:

$$\theta_i = \theta_{i-1} - \alpha \frac{\mathrm{d}}{\mathrm{d}\theta} \rho(g_\theta(\mathbf{x}), \mathbf{y}) \bigg|_{\theta=\theta_{i-1}} . \tag{2.51}$$

Equation (2.51) shows that the parameters of the function are updated iteratively following the opposite direction of the gradient of the loss function ρ. This makes sense because in this way we will find some value for θ that will locally minimize the loss function ρ. As we can see, GD finds a sequence $\theta_1, \theta_2, \ldots, \theta_n \approx \theta^*$ where θ^* is the configuration that minimizes locally $\rho(\mathbf{y}, g_{\theta^*}(\mathbf{x}))$. The function g_{θ^*} belongs to the functional space $G_\Theta \subset F$, thus g_{θ^*} could be seen as a projection $\Pi_\Theta F \to G_\Theta$. Figure 2.4 shows how GD works.

The principle behind Functional Gradient Descent (FGD) [13] is very similar, the goal is to find a function g by exploiting the gradient of the loss, but in a very different way. Instead of finding a parameter θ for which

Algorithm 4: Functional Gradient Descent

```
Data: $\mathbf{x}$, $\mathbf{y}$, $\rho$, $G$, $M$
Result: $\hat{g}_M$
$j \leftarrow 0$;
$\hat{g}_0 \leftarrow 0$ ;
for $i = \{0 \dots M-1\}$ do
    $\mathbf{u} \leftarrow -\frac{\partial}{\partial \mathbf{g}} \rho(\mathbf{y}, \mathbf{g})\Big|_{\mathbf{g}=\hat{g}_j(\mathbf{x})}$;
    $g_j \leftarrow \arg\min_{g \in G} \|g(\mathbf{x}) - \mathbf{u}\|$;
    $\hat{g}_{j+1} \leftarrow \hat{g}_j + g_j$;
end
return $\hat{g}_M$;
```

$g_\theta \approx \overline{g}$, we are interested in finding a non parametric function $g \approx \overline{g}$ obtained as the sum of individual gradients. We know that by truncating the first order of the Taylor series that we can approximate a function by using its derivative (or gradient):

$$g(x_0 + \epsilon) \approx g(x_0) + \epsilon \frac{\mathrm{d}}{\mathrm{d}x} g(x)\Big|_{x=x_0} . \tag{2.52}$$

The FGD starts from a function $\hat{g}_0 \in G$ (where $G \subset F$) which we can suppose to be equal to zero (but it could be initialized randomly). At each iteration FGD computes the gradient descent direction from $\hat{g}_j$ in order to minimize the error $\rho(\mathbf{y}, \hat{g}_j(\mathbf{x}))$:

$$\mathbf{u} = -\frac{\partial}{\partial \mathbf{g}} \rho(\mathbf{y}, \mathbf{g})\Big|_{\mathbf{g}=\hat{g}_j(\mathbf{x})} . \tag{2.53}$$

We approximate the gradient with a function g_j such that $g_j(\mathbf{x}) \approx \mathbf{u}$. Note that this step is performed by a base-procedure (a SL algorithm) $\mathcal{G}$ which could use either a parametric or a non-parametric learner.

As we see, in each iteration $q_j \approx -\frac{\partial}{\partial \mathbf{g}} \rho(\mathbf{y}, \mathbf{g})\Big|_{\mathbf{g}=\hat{g}_j(\mathbf{x})}$ thus $\hat{g}_{j+1} = \hat{g}_j - \frac{\partial}{\partial \mathbf{g}} \rho(\mathbf{y}, g)\Big|_{g=\hat{g}_j(\mathbf{x})}$ which for (2.52) corresponds a function closer to the target function $\overline{g}$.

In the method described a certain point represents a function $\hat{g}_j$, and to approximate the function $\hat{g}_{j+1}$ it approximates the direction of the gradient descent with g_j and then $\hat{g}_{j+1} = \hat{g}_j + g_j$. Thanks to (2.52) we can see that by approaching $j \to \infty$, $\hat{g}_j$ approximates the target model $\overline{z}$ as shown in Figure 2.5.

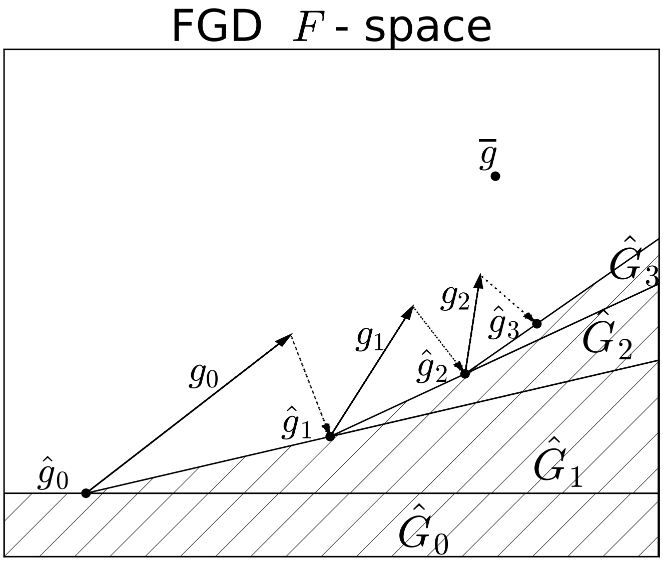

Figure 2.5: The picture is a representation of the FGD algorithm. Here the full arrows represent the gradients, while the dashed ones are the projection from the F space to the space $\hat{G}_j = \{\hat{g}_j\}$. The $\hat{G}_j$ space augment at each iteration, because every dot is the summation by the dot at the step $j - 1$ and represents the function $\hat{g}_j$, while the function g_j are represented by the difference $\hat{g}_j - \hat{g}_{j-1}$.

What it is really interesting is that FGD works directly in the F space (that is also equivalent to say that works in the Y space because X is fixed), and then projects the gradient found in the space G of the function approximator. GD works in the space of parameters Θ, without projecting the gradient. Performing the gradient in Y is better, because, in theory, it will lead to find the real optimum, and it is a "natural choice", while performing the gradient in Θ makes the gradient move in a less natural space, and by the way, the optimum g_{θ^*} that we found with GD is not the target $\overline{g}$.

The last advantage of FGD with respect to GD is that every time that we iterate, the functional space of $\hat{g}_j$ increases, so that $\hat{g}_j$ can grow in the direction of $\overline{g}$. We refer to the functional space of $\hat{g_j}$ by $\hat{G}_j = \{\hat{g}_j\}$. Figure 2.5 actually shows how FGD works.

It comes with no surprise that Boosting is a good way to lower down the bias, because it actually tries to always improve the miss prediction, moving always closer to the target function $\overline{g}$, and increasing at each iteration the expressivity of the functional space, giving thus to the learner more chance to approach g. Actually it is also interesting to see that the construction of the final learner, is done by following a "smart" direction: the one of the gradient. So, we approach a low bias by starting from a learner with a small variance too, and this is actually very different with respect to just picking up a very complex model and trying to over-fit. We will see in the next section that actually, the fact that the model is additive does not imply that also the variance of the model sums up linearly: intuitively, the fact that each learner depends by the previous ones combined also by the fact that the gradient will become smooth and close to zero approaching the target function $\overline{g}$, is equivalent somehow to subtract some degrees of freedom in the learning process.

2.2.4 L-2 Boost

The FGD algorithm described above is quite general and it allows different loss functions. We would like to study the case of the L_2 norm also called square error. In the case of L_2 norm, FGD results particularly simple and intuitive, showing that at each iteration of the algorithm we are approximating the *residual* between the target and the function f_m. Let define $\rho_{L_2} = |f - y|^2/2$ [15], it is easy to show that:

$$\frac{\mathrm{d}}{\mathrm{d}f}\rho_{L_2}(f, y) = f - y \tag{2.54}$$

which let us derive $\mathbf{u}$ from Equation (2.53):

$$\mathbf{u} = \mathbf{y} - \hat{g}_j(\mathbf{x}). \tag{2.55}$$

The difference $f - y$ could be seen as the error between the estimation of y and f. The FGD in the case of L_2-boost we know that at each step we use the base-procedure $\mathcal{G}$ in order to find a function $g_j(\mathbf{x}) \approx \mathbf{u}$, where u is previously defined in Equation (2.55).

L_2-Boost iterates is an algorithm that refits multiple times the residual [27]. When we sum up all the models, their variance does not sum up linearly, because each model depends in some way on the previous one. Very intuitively, the residual after some iterations will be smoother and simpler. The complexity of the new combined model increases with respect the singular models, and approaching $j \to \infty$, the variance of the j-th model is bounded by the noise.

We would like to formally define the bias and the variation of the model. To do this, we have to introduce some simplification. The first one, is to hide somehow the variable $\mathbf{x}$, because during the learning process $\mathbf{x}$ is fixed. In fact, when we compute the gradient of $\rho(\mathbf{y}, \mathbf{g})$, the term $\mathbf{g}$ belongs to the target space Y. We could so assume that the function g_j does not belong to the space G (previously defined as a subset of F) but to the space $Y \to Y$ because g_j could be seen as a translation of the output of $\hat{g}_j$. We can now define the boosting operator at iteration j as $\mathcal{B}_j : Y \to Y$, and the operator $S : Y \to Y$ mapping the responses $\mathbf{y}$ to some fitted values in Y.

Definition 5. *Boosting operator [15]*
The boosting operator $\mathcal{B}_j$ is defined as

$$\mathcal{B}_j = I - (I - S)^{j+1}. \tag{2.56}$$

An easy way to show the effectiveness of L_2-boost it to restrict the space the learner S just in the case of linear and symmetric learner with enginevalues $\{\lambda_k; k = 1, \ldots, n\}$, based on the deterministic input set $x_1, \ldots, x_n$, then the enginevalues of the L_2-Boost operator $\mathcal{B}_j$ are $\{(1 - (1 - \lambda_k)^{j+1}) : k = 1, \ldots, n\}$.

Now we would like to write the MSE as a combination of bias and variance.

$$\mathrm{MSE}(j, S; f, \sigma^2) = \sum_{i}^{n} (x_i - f(x_i))^2 = \mathrm{Bias}^2(j, S; f) + \mathrm{Var}(j, S; \sigma^2)$$

and we can show that:

$$\begin{aligned} \mathrm{Bias}^2(j, S; f) &= \frac{1}{n} \sum_{i=1}^{n} (\mathbb{E}[\hat{g}_j(x_i)] - f(x_i))^2 \\ &= \frac{1}{n} f^T U \mathrm{diag}((1 - \lambda_k)^{2j+2}) U^T f, \end{aligned} \tag{2.57}$$

$$\begin{aligned} \mathrm{Var}(j, S; \sigma^2) &= \frac{1}{n} \sum_{i}^{n} (\hat{g}_j(x_i)) \\ &= \sigma^2 \frac{1}{n} \sum_{k=1}^{n} (1 - (1 - \lambda_k)^{j+1})^2. \end{aligned} \tag{2.58}$$

We can argue, under the assumptions made, that the bias component of the error decays exponentially fast with respect to j, and the variance increases exponentially slower with respect to j, and when j approaches to ∞ then the overall MSE converges to σ^2, furthermore if there exists $\lambda_k < 1$ then there is a j such that at that iteration the MSE is strictly lower than σ^2. Moreover, let $\beta = U^T f = (\beta_1, \ldots, \beta_n)^T$, if $\beta_k^2 / \sigma^2 > 1/(1 - \lambda_k)^2 - 1$ for all k with $\lambda_k < 1$ then MSE improves over the linear learner S.

In other words, while the bias decreases exponentially fast, the variance increases with exponentially diminishing terms[15].

Algorithm 5: L_2-Boost

```
Data: $\mathbf{x}$, $\mathbf{y}$, $\mathcal{G}$, $M$
Result: $\hat{g}_M$
$j \leftarrow 0$;
$\hat{g}_0 \leftarrow 0$ ;
for $i = \{0 \ldots M-1\}$ do
    $\mathbf{u} \leftarrow \mathbf{y} - \hat{g}_j(\mathbf{x})$;
    $g_j \leftarrow \arg\ \min_{g \in G} \|g(\mathbf{x}) - \mathbf{u}\|$;
    $\hat{g}_{j+1} \leftarrow \hat{g}_j + g_j$;
end
return $\hat{g}_M$;
```

Chapter 3

Theoretical Analysis

Well, the way of paradoxes is the way of truth. To test reality we must see it on the tight rope. When the verities become acrobats, we can judge them.

Oscar Wilde - The Picture of Dorian Gray

B-FQI is an algorithm that belongs to the class of AVI algorithms. As explained in Chapter 2, AVI is a meta-algorithm that is based on an approximation built at each iteration of T^*Q_{k-1}. The difference among AVI algorithms reside in how this approximation is performed and on how the Optimality Bellman operator is exploited. In the class of FVI algorithms we use the empirical Optimality Bellman operator which is based on samples. FQI is a family of batch algorithms that combine the approximation inducted by projecting at each iteration T^*Q_{k-1} and the empirical Bellman operator which is an approximation of the Optimality Bellman operator. There are many FQI algorithms, and the main difference between those algorithms is on how T^*Q_{k-1} is approximated. In Section 3.1 we introduce the general AVI procedure and the mathematical concepts that are exploited in the theoretical analysis of AVI methods, while in Section 3.2 we will present some theoretical results of general AVI algorithm. We introduce in Section 3.4 Regularized FQI (R-FQI): an algorithm that introduces regularization in the loss function relying on the idea that we can use a very complex approximator, and avoid the overfitting by dynamically setting the penalization coefficient which in practice restrict the functional space. We introduce in Section 3.3 Value Pursuit Iteration (VPI) which is an approximate value iteration algorithm that finds a policy close to optimal one for problems with large state space: it finds a good sparse approximation of the optimal value function being almost insensitive to the number of irrelevant features, and each iteration it enrich the functional space based on the currently learned value function. Boosted FQI (B-FQI), introduced in Section 3.5, is an FQI algorithm which approximate the Bellman residual instead of approximate directly the value function, so that at each iteration can use a weak learner and then it constructs an additive model from all the approximations of the bellman residuals. The former technique could be seen as a boosting method, with the only difference that here the target changes (but when B-FQI converges, approaching $k \to \infty$, the target not changes anymore) thus B-FQI uses weak learners with the advantage of requiring less time in the approximation phase, and plays the counterpart with respect to R-FQI which do substantially the opposite since it uses complex functional spaces. What R-FQI, PVI, and B-FQI share is the idea to dynamically adapt the complexity of the functional space, overcoming the issue given by the fact that the complexity of the value function is unknown and it changes iteraton by iteration.

3.1 Introduction to the Mathematical Framework

In this section we introduce the notations and mathematical formalisms required in the rest of the chapter. Since the algorithms have a lot in common (they all belong to the AVI family), they share a lot of mathematical objects, like how the dataset is generated, the use of the Optimality Bellman operator or the empirical Optimality Bellman operator; and in order to exploit their theoretical analysis they all uses same mathematical concepts like the concentrability. We try here to use for them the same mathematical notation. We use the same notation as in Farahmand 2011 [21].

The MDP is the same as described in Chapter 2 with discrete action space $\mathcal{A}$. The dataset that the following technique use is not fixed but it changes each iteration k (this is a general assumption used to derive theoretical results), and each state-action pair is drawn from a distribution $\mu \in \mathcal{M}(\mathcal{X} \times \mathcal{A})$ which could be seen as a distribution over the states $\mu_{\mathcal{X}} \in \mathcal{M}(\mathcal{X})$ where the action is sampled by a policy $\pi_b : \mathcal{X} \to \mathcal{M}(\mathcal{A})$. The collection of action-state pairs is defined as $H_N^{(k)} = \{(X_i^{(k)}, A_i^{(k)}) | (X_i^{(k)}, A_i^{(k)}) \sim \mu\}$, and the overall dataset is defined as $D_{N_k}^{(k)} = \{(X_i^{(k)}, A_i^{(k)}, R_i^{(k)}, X{\prime}_i^{(k)}) | (X_i^{(k)}, A_i^{(k)}) \in H_i^{(k)}, R_i^{(k)} \sim \mathcal{R}(X_i^{(k)}, A_i^{(k)}), X{\prime}_i^{(k)} \sim P(\cdot | X_i^{(k)}, A_i^{(k)})\}$[1].

As stated in Chapter 2, AVI is a meta-algorithm which produces at each iteration an approximation of the application of the Bellman optimal operator $Q_k \approx T^* Q_{k-1}$. Basically from how $T^* Q_{k-1}$ is approximated we will have different FQI algorithm. To give some example *Regularized FQI*

[1] Sometimes we will omit k when it is not necessary or implicit.

uses regularization in the loss function, *Pursuit Value Iteration* uses kernel functions that are enriched iteration after iteration to represent Q_k, and *B-FQI* uses an additive model by approximating the Bellman Residual. In the theoretical analysis of AVI, we assume to not know how T^*Q_{k-1} is approximate, all we need to know is the approximation error between Q_k and T^*Q_{k-1} at each iteration k.

Definition 6. Approximation Error
Consider the AVI procedure and the sequence of action-value function estimates $Q_0, Q_1, \ldots, Q_K$, in which Q_{k+1} is the result of approximately applying the Optimality Bellman operator to the previous estimate Q_k , i.e., $Q_{k+1} \approx T^*Q_k$. Denote the approximation error caused at each iteration by

$$\epsilon_k = T^*Q_k - Q_{k+1} \quad (k \geq 0); \quad \epsilon_{-1} = Q^* - Q_0. \tag{3.1}$$

Note that $\epsilon_{-1} = Q^* - Q_0$ is introduced here for notational simplification (it will be used in 3.4).

Concentrability. Very commonly in AVI we want to estimate the error of approximation between two function (e.g., between Q^* and Q_k). Since $Q^* - Q_k$ is not a real value, we generally want to compute the p-norm of the difference between the two functions computed on a set of points or with respect to a distribution. The error $\|Q^* - Q_k\|_{p,\mu}$ assumes different values depending on the choiche of p and μ. Since we want to upper-bound the error, we would like to give the appropriate error bounds with respect to the norm and the distribution that we are using. In order to do that, we use the concentrability measures. There are different concentrability measures, we present here the ones that will be needed in the rest of this document.

Definition 7. One step concentrability (Definition 5.2 Farahmand 2011 [21]) *Let μ be a distribution over the state-action pairs, $(X, A) \sim \mu$, $\mu_{\mathcal{X}}$ be the marginal distribution of $\mathcal{X}$, and $\pi_b(\cdot|\cdot)$ be the conditional probability of A given X. Further, let P be a transition probability kernel $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ and $P_{x,a} = P(\cdot|x, a)$. Define the* one-step concentrability coefficient *w.r.t. μ as*

$$C_{\mu\to\infty} = \left(\mathbb{E}\left[\sup_{(y,a')\in\mathcal{X}\times\mathcal{A}} \left| \frac{1}{\pi_b(a'|y)} \frac{\mathrm{d}P_{X,A}}{\mathrm{d}\mu_{\mathcal{X}}}(y) \right| \right] \right)^{\frac{1}{2}}, \tag{3.2}$$

where $C_{\mu\to\infty} = \infty$ if $P_{x,a}$ is not absolutely continuous w.r.t. $\mu_{\mathcal{X}}$ for some $(x, a) \in \mathcal{X} \times \mathcal{A}$, or if $\pi_b(a'|y) = 0$ for some $(y, a') \in \mathcal{X} \times \mathcal{A}$.

Definition 8. Expected Concentrability of the Future State-Action Distribution (Definition 3.1 Farahmand 2011 [21])

Given $\rho, v \in \mathcal{M}(\mathcal{X} \times \mathcal{A}), m \geq 0$ and an arbitrary sequence of stationary policies $(\pi_m)_{m \geq 1}$, let $\rho P^{\pi_1} P^{\pi_2} \dots P^{\pi_m} \in \mathcal{M}(\mathcal{X} \times \mathcal{A})$ denote the future state-action distribution obtained when the first state-action is distributed according to ρ and then we follow the sequence of policies $(\pi_k)_{k=1}^{m}$. For integers $m_1, m_2 \geq 1$ and policies π, π_1, π_2 define the following concentrability coefficients, which are used in the analysis of AVI:

$$c_{VI_1,\rho,v}(m_1, m_2; \pi) \triangleq \left(\mathbb{E}\left[\left| \frac{\mathrm{d}(\rho(P^{\pi})^{m_1}(P^{\pi^*})^{m_2}}{\mathrm{d}v}(X, A) \right|^2 \right] \right)^{\frac{1}{2}} \tag{3.3}$$

$$c_{VI_2,\rho,v}(m_1; \pi_1, \dots, \pi_k) \triangleq \left(\mathbb{E}\left[\left| \frac{\mathrm{d}(\rho(P^{\pi}_k)^{m_1} P^{\pi_{k-1}} P^{\pi_{k-2}} \dots P^{\pi_1}}{\mathrm{d}v}(X, A) \right|^2 \right] \right)^{\frac{1}{2}} \tag{3.4}$$

with $(X, A) \sim v$. If the future state-action distribution $\rho(P^{\pi})^{m_1}(P^{\pi^*})^{m_2}$ (or likewise $\rho(P^{\pi}_k)^{m_1} P^{\pi_{k-1}} P^{\pi_{k-2}} \dots P^{\pi_1}$) is not absolutely continuous with respect to v, then take $c_{VI_1,\rho,v}(m_1, m_2; \pi) = \infty$ and similarly $c_{VI_2,\rho,v}(m_1; \pi_1, \dots, \pi_k) = \infty$.

Empirical Bellman Operator Until now we have seen the error given by the approximation of Q_k, and basically this error is caused by the projection of $T^* Q_k$ onto a functional space. There is another source of error that is strictly connected by the used of a finite dataset. In fact, in our application, we do not know what is the precise definition of the Bellman operator, because we do not have access to the analytical definition of the model. What we can do is to approximate the Bellman operator by using samples generated from the model. With the distribution already defined μ, we construct $H_{N_k}^{(k)}$ which contains state-action pairs, and then we generate (assuming to have a generative model), a dataset $D_{N_k}^{(k)}$ which contains also the next states and the rewards observed.

Definition 9. Empirical Optimality Bellman operator (Definition 5.2 [21]) *Let $D_{N_k}^{(k)} = \{X_i^{(k)}, A_i^{(k)}, R_i^{(k)}, X'^{(k)}_i\}_{i=1}^{N_k}$ be a set of transitions such that $(X_i^{(k)}, A_i^{(k)}) \sim \mu, R_i^{(k)} \sim \mathcal{R}(\cdot | X_i^{(k)}, A_i^{(k)})$ and $X'^{(k)}_i \sim P(\cdot | X_i^{(k)}, A_i^{(k)})$ and define $H_{N_k}^{(k)} = \{(X_1^{(k)}, A_1^{(k)}), \dots, X_{N_k}^{(k)}, A_{N_k}^{(k)})\}$. The empirical Optimality Bellman operator $\hat{T}_k^* : H_{N_k} \to \mathbb{R}^{N_k}$ is defined as*

$$(\hat{T}^* Q)(X_i^{(k)}, A_i^{(k)}) \triangleq R_i^{(k)} + \gamma \max_{a' \in \mathcal{A}} Q(X'^{(k)}_i, a'). \tag{3.5}$$

We introduce here also the empirical Bellman residual*:*

$$\tilde{\varrho}_k \triangleq \hat{T}^* Q_k - Q_k. \tag{3.6}$$

The whole class of Fitted Q-Iteration (FQI) algorithms [20, 66, 22, 24] is based on using the empirical optimal Bellman operator in $\mathcal{F}$. The correctness of this procedure is guaranteed by $\mathbb{E}\left[\hat{T}^* Q_k(X_i^{(k)}, A_i^{(k)}) | X_i^{(k)}, A_i^{(k)}\right] = T^* Q_k(X_i^{(k)}, A_i^{(k)})$. Note that the same result holds for the Bellman residual $\mathbb{E}\left[\tilde{\varrho}_k(X_i^{(k)}, A_i^{(k)}) | X_i^{(k)}, A_i^{(k)}\right] = \varrho_k(X_i^{(k)}, A_i^{(k)})$.

3.2 Theoretical Analysis of AVI

The theoretical analysis is important both to understand the general properties and error bounds of AVI algorithms, and also to support new bounds for each specific AVI algorithm. We report here all the results that are important for our analysis and to compare the AVI results with the algorithms presented. The first result that we introduce is an important property of the Optimality Bellman operator which states that the Optimality Bellman operator is $\gamma C_{\mu\to\infty}$-Lipschitz with respect to the Banach space of Q-functions equipped with $\|\cdot\|_\mu$ [21]. This property could be explicitly wrote as:

$$\|T^* Q_1 - T^* Q_2\|_\mu \leq \gamma C_{\mu\to\infty} \|Q_1 - Q_2\|_\mu. \tag{3.7}$$

This property shows that the bellman operator is $\gamma C_{\mu\to\infty}$ contractive in a normed space $\|\cdot\|_\mu$. This property will be extensively used in the error propagation analysis of B-FQI.

3.2.1 Error Propagation

With the error propagation analysis we want to upperbound the error at the iteration k with respect to the approximation error committed in the previous iterations. This is done assuming to know the Optimality Bellman operation T^* and to derive Q_k with an approximation error as defined in Equation (3.1). Note that the definition given by Equation (3.1) is general and does not take in account on how the approximation between Q_k and $T^* Q_{k-1}$ is performed. Another assumption that we do is that the norm operators works directly with the distributions μ without accounting for the dataset generation and thus without the presence of an empirical Optimality Bellman operator. Here we present the upperbound of the propagation error in the case of AVI.

A first results is given by the following theorem:

Theorem 5. Bound on the Approximation error (Theorem 2 Farahmand 2012 [24])
Let $(Q_k)_{k=0}^K$ *be a sequence of state-action value functions and* ϵ_k *as defined*

by Equation (3.1). *Let* $\mathcal{F}^{|A|} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^{|A|}$ *be a subset of vector-valued measurable functions. Then,*

$$\inf_{Q' \in \mathcal{F}^{|\mathcal{A}|}} \|Q' - T^* Q_k\|_\mu \leq \inf_{Q' \in \mathcal{F}^{|A|}} \|Q' - (T^*)^{k+1} Q_0\|_\mu + \sum_{i=0}^{k-1} (\gamma C_{v\to\infty})^{k-i} \|\varepsilon_i\|_\mu. \tag{3.8}$$

Theorem 5 shows that the minimum error in approximating $T^* Q_k$ with the functional space $\mathcal{F}^{|\mathcal{A}|}$ is bounded by the minimum error that will results in projecting $(T^*)^{k+1} Q_0$ and a linear combination of all the previous approximation error $\|\epsilon_i\|_\mu$. This result is very interesting because upperbounds the projection only thanks to the previous approximation errors, and the error that would be committed by approximating $(T^*)^{k+1} Q_0$.

We now introduce an interesting result on how the error propagates in a general AVI procedure:

Theorem 6. Error Propagation for AVI (Theorem 3.4 Farahmand 2001 [21])
Let k *be a positive integer,* $Q_{\max} \leq \frac{R_{\max}}{1-\gamma}$, *and* ρ *an initial state-action distribution. Then for any sequence* $(Q_i)_{i=0}^{k-1} \subset B(\mathcal{X} \times \mathcal{A}, Q_{\max})$ *and the corresponding sequence* $(\epsilon_k)_{i=0}^{k-1}$ *defined in* (3.1), *we have*

$$\|Q^* - Q^{\pi_k}\|_{1,\rho} \leq \frac{2\gamma}{(1-\gamma)^2} \left[2\gamma^k Q_{\max} + \inf_{g \in [0,1]} C^{1/2}_{VI,\rho,\mu}(k; g) \mathcal{E}^{1/2}(\epsilon_0, \dots, \epsilon_{k-1}; g) \right], \tag{3.9}$$

where

$$\begin{aligned} C^{1/2}_{VI,\rho,\mu}(k;g) &= \left(\frac{1-\gamma}{2}\right)^2 \sup_{\pi'_1,\dots,\pi'_k} \sum_{i=0}^{k-1} \alpha_k^{2(1-g)} \Bigg[\sum_{m \geq 0} \gamma^m \big(c_{VI_1,\rho,\mu}(m, k-i; \pi'_k) \\ &\qquad + c_{VI_2,\rho,\mu}(m+1; \pi'_{i+1}, \dots, \pi'_k) \big) \Bigg]^2 \end{aligned} \tag{3.10}$$

and $\mathcal{E}(\epsilon_0, \dots, \epsilon_{k-1}; g) = \sum_{i=0}^{k-1} \alpha_i^{2g} \|\epsilon_i\|^2_\mu$. *Where*

$$\alpha_i = \begin{cases} \frac{(1-\gamma)\gamma^{k-i-1}}{1-\gamma^{k+1}} & 0 \leq i < k, \\ \frac{(1-\gamma)\gamma^k}{1-\gamma^{k+1}} & i = k. \end{cases} \tag{3.11}$$

The result given by Theorem 6 shows that $\|Q^* - Q^{\pi_k}\|_{1,p}$ is upperbounded by two main terms: the first one depends by $Q_{\max}$ while the second depends by the concentrability and the previous approximation error committed.

3.2.2 Finite-Samples Error Analysis

The finite samples error analysis uses the empirical Optimality Bellman operator introduced in Section 3.1 and deals with the fact that we have a dataset composed by a finite number of samples, and thus here we keep account of both the source of error derived by the approximation caused by the fact that Q_k is an approximation of Q_{k-1} and the fact that we use here the empirical Optimality Bellman operator, which is an approximation of the Optimality Bellman operator. We report from a work of Munos et al. [55] an example of finite-sample bound for a generic AVI algorithm. In order to define such bound we must introduce some concepts:

Definition 10. $(\mathcal{E}, q)$-covering number
Let $\mathcal{F}$ be an infinite functional space. Let $\mathcal{E} > 0$, $q \geq 1$, $x^{1:N} \triangleq (x_1, \dots, x_N) \in \mathcal{X}^N$. The $(\mathcal{E}, q)$-covering number is the smallest number m such that $\mathcal{F}(x^{1:N})$ can be covered my m balls of the normed-space $(\mathbb{R}^N, \|\cdot\|_q)$ with centers in $\mathcal{F}(x^{1:N})$ and radius $N^{1/q}\mathcal{E}$. The $(\mathcal{E}, q)$-covering number of $\mathcal{F}(x^{1:N})$ is denoted by $\mathcal{N}_q(\mathcal{E}, \mathcal{F}(x^{1:N}))$.

When $X^{1:N}$ are i.i.d with common underlying distribution μ then $\mathbb{E}\left[\mathcal{N}_q(\mathcal{E}, \mathcal{F}(X^{1:N}))\right]$ shall be denoted by $\mathcal{N}_q(\mathcal{E}, \mathcal{F}, N, \mu)$. The logarithm of $\mathcal{N}_q$ is called the q-norm metric entropy of $\mathcal{F}$. The idea underlying the covering numbers is that what really matters when bounding maximal deviations is how much the functions in the function space vary *at the actual samples.* Of course, without imposing any conditions on the function space, covering numbers can grow as a function of the sample size.

Theorem 7. Finite-Sample Bound for AVI *Let $V_{\max} = R_{\max}/(1-\gamma)$, fix a real number $p \geq 1$, integer $N \geq 1$, $\mu \in \mathcal{M}(\mathcal{X})$ and $\mathcal{F} \subset \mathcal{B}(\mathcal{X}; V_{\max})$. Pick any $V \in \mathcal{B}(\mathcal{X}; V_{\max})$ and let $V'(X_i) = \arg\min_{f\in\mathcal{F}} \sum_{i=1}^{N} |f(X_i) - \max_{a\in\mathcal{A}} R^{X_i,a} + \gamma V(Y_i^{X_i,a})|^p$ where $X_i \sim \mu$, $R^{X_i,a} \sim \mathcal{R}(\cdot|X_i, a)$ and $Y^{X_i,a} \sim P(\cdot|X_i, a)$ for $i = 1, \dots, N$. Let $\mathcal{N}_0(\frac{1}{8}(\frac{\mathcal{E}}{4})^p, \mathcal{F}, N, \mu)$. Then for any $\mathcal{E}, \delta > 0$,*

$$\|V' - TV\|_{p,\mu} \leq \inf_{f\in\mathcal{F}} \|f - TV\|_{p,\mu} + \mathcal{E} \tag{3.12}$$

holds with probability at least $1-\delta$ provided that

$$128\left(\frac{8V_{\max}}{\mathcal{E}}\right)^{2p} (\log(1/\delta) + \log(32\mathcal{N}_0(N))) \quad < \quad N, \tag{3.13}$$

$$\frac{8(R_{\max} + \gamma V_{\max})^2}{\mathcal{E}^2} (\log(1/\delta) \log(8N|\mathcal{A}|)) \quad < \quad 1, \tag{3.14}$$

hold simultaneously.

Theorem 7 shows clearly that the error between V' and T^*V can be bounded with probability at least $1 - \delta$ by two terms: the first term is the approximation error, while the second term $\mathcal{E}$ depends by the number of samples in the dataset. The theorem also shows that, without taking account of a logarithmic term, if δ is fixed then $\mathcal{E}$ scales linearly with $N^{\frac{1}{2p}}$.

3.3 Value Pursuit Iteration

Value Pursuit Iteration (VPI) [24] is an approximate value iteration algorithm that finds a policy close to the optimal one, for large state-space RL tasks. It works with a dictionary of features and finds a good sparse approximation of the optimal value function. At each iteration the set of features is augmented both with features that represent the value functions found at the previous step and by those features that best approximate the bellman residual. The goal is to find a sparse representation of the optimal value function by using “smart features”, and to grow the functional space in a convenient way. The sparsity in literature is commonly tackled using regularization such as the L_1-norm of the weights [45], [41] and [29]. Another approach is based on greedly adding atoms to the representation of the target function. Examples of this approaches in SL are Matching Pursuit and Orthogonal Matching Pursuit (OMP) [59] [18]. VPI has been designed to address the problem of approximating the value function when not much a priori knowledge is available. The core of VPI is the sparse feature selection provided by OMP: the algorithm works in theory even with a countably infinite set of features. In practice we define a dictionary of features $\mathcal{D} = \{g_1, g_2, \dots\}$ where every g_i is a real valued function defined in the state-action domain. The ideal situation is achieved when $Q^* = \sum_{i \geq 1} w_i g_i$ where only one of the weights w_i assumes value one and the others are set at zero. Of course we are not usually so lucky to have the optimal value function in our dictionary, but in exact VI algorithms there is k such that Q_k explains a large part of T^*Q_k, so we could use the other features to explain the bellman residual.

Dictionaries. Let $\mathcal{H}$ be a Hilbert space equipped with an inner product norm $\|\cdot\|$. We call $\mathcal{D} = g_1, g_2, \dots$ with $g_i \in \mathcal{H}$ a *dictionary.* The class $\mathcal{L}_1(\mathcal{D}) = \mathcal{L}(\mathcal{D}, \|\cdot\|)$ consists of those functions $f \in \mathcal{H}$ that admit an expansion $f = \sum_{g \in \mathcal{D}} |c_g g$ with c_g being absolutely summable [3]. The norm operator is defined compactly as $\|f\|_{\mathcal{L}_1(\mathcal{D};D_n)} \triangleq \inf\{\sum_{g \in \mathcal{D}} |c_g| : f = \sum_{g \in \mathcal{D}} c_g g\}$ and we may use $\|f\|_{\mathcal{L}_1(\mathcal{D};v)}$ with a similar meaning. When we denote $\mathcal{D}_m$, where m represents the cardinality of $\mathcal{D}$ and with $\mathcal{D}_{m,k}$ we denote the k-th element

Algorithm 6: Value Pursuit Iteration [24]

```
Input: Initial dictionary $\mathcal{D}_0$, number of dictionary atoms $m$, link function $\{\sigma_i\}_{i=1}^{m'}$, state-action distribution $v$ and number of iterations $K$.
Return: $Q_k$;
$Q_0 \leftarrow 0$;
$\mathcal{D}'_0 \leftarrow \emptyset$;
for $k = 0, \dots, K-1$ do
    Construct the dataset $D^{(k)}_{N_k}$
    $\hat{Q}^{(0)}_{k+1} \leftarrow 0$;
    //Orthogonal Matching Pursuit loop
    Normalize $\mathcal{D}_{0,m}$ and $\mathcal{D}'_k$ according to $\|\cdot\|_{D^{(k)}_{N_k}}$ and then call them $\mathcal{B}_k$ and $\mathcal{D}'_k$;
    for $i = 1, \dots, c_1 n$ do
        $r^{(i-1)} \leftarrow \hat{T} Q_k - \hat{Q}^{i-1}_{k+1}$;
        $g^{(i)} \leftarrow \arg\max_{g \in \mathcal{D}_k \cup \mathcal{D}'_k} \left|\left\langle r^{(i-1)}, g \right\rangle\right|$;
        $\hat{Q}^{(i)}_{k+1} \leftarrow \Pi^{(i)} \hat{T}^* Q_k$;
    end for
    $i^* \leftarrow \arg\min_{i \geq 1} \left\{ \left|\left| \beta_{Q_{\max}} \hat{Q}^{(i)}_{k+1} - \hat{T}^* Q_k \right|\right|^2_{D^{(k)}_{N_k}} + c_2 (Q_{\max})^{i \ln(n)/n} \right\}$;
    $Q_{k+1} \leftarrow \hat{Q}^{(i)}_{k+1}$;
    $\mathcal{D}'_{k+1} \leftarrow \mathcal{D}'_k \cup \{\sigma_i(\beta_{Q_{\max}} Q_{k+1}, \mathcal{D}_k \cup \mathcal{D}'_k)\}_{i=1}^{m'}$;
end for
```

of $\mathcal{D}_m$. Furthermore $\mathcal{L}_{1,\alpha} = \{h : \|h\|_{\mathcal{L}_1(\mathcal{D};\|\cdot\|)} \leq C \quad \text{and} \quad \|f - h\| \leq C m^{-\alpha}\}$. We finally define the truncation operator $\beta_L : \mathcal{F} \to \mathcal{B}(\mathcal{X} \times \mathcal{A}; L)$.

OMP algorithm works with two inner cycles: the outer one chooses the function that best approximates Q_{k+1} among different candidates $\beta_{Q_{\max}} \tilde{Q}_{k+1}$ and extends the dictionary $\mathcal{D}'_{\|+\infty} \leftarrow \mathcal{D}'_{\|} \cup \{\sigma_i(\beta_{Q_{\max}} Q_{k+1}; \mathcal{D}_{\|} \cup \mathcal{D}'_{\|})\}_{i=1}^{m'}$. The inner cycle generates iteratively $\hat{Q}^{(i)}_k$ by computing the residual $r^{(i-1)} = \hat{T} Q_k - \hat{Q}^{(i-1)}_{k+1}$ and then finding the feature $g^{(i)}$ that minimizes the scalar product with respect to $r^{(i-1)}$. Since every g_i has been normalized at the beginning of the iteration, the feature g_i that best minimizes the scalar product is the one that is more aligned to $r^{(i-1)}$ and $\hat{Q}^{(i)}_{k+1}$ is the best linear combination of $\{g^{(1)}, \dots, g^{(i)}\}$ that describes $\hat{T}^* Q_k$. All the procedure is formally explained in Algorithm 6.

3.3.1 Error Propagation

The actual error propagation bounds (at the best of our knowledge) are the same bounds for AVI, thus they are not really specific for the VPI case. In VPI article [24] is reported Theorem 6 which we already mentioned in Section 3.2. We report here Lemma 1 from that article that actually is a generic AVI bound that will be used in Theorem 8.

Lemma 1. *Let $(Q_i)_{i=0}^{k} \subset \mathcal{B}(\mathcal{X} \times \mathcal{A}; Q_{\max})$ be a $Q_{\max}$-bounded sequence of measurable state-action value functions. Define ϵ_i for $0 \leq i \leq k-1$ as in Equation 3.1. Then:*

$$\|Q_k - T^* Q_k\|_v^2 \leq \frac{(1+\gamma C^2_{v\to\infty})}{1-\gamma}\left[\sum_{i=0}^{k-1} \gamma^{k-1-i}\|\epsilon_i\|_v^2 + \gamma^k (2Q_{\max})^2\right]. \quad (3.15)$$

where $C_v(m) \triangleq \|\frac{\mathrm{d}(v(P^{\pi^*}))^m}{\mathrm{d}v}\|_\infty$.

Lemma 1 shows that the bellman residual in the normed space $\|\cdot\|_v^2$ is bounded by two terms both scaled by $\frac{(1+\gamma C^2_{v\to\infty})}{1-\gamma}$: the first term is a linear combination of the previous approximation errors, where each of them is weighted by concentrability $C_v(k-1-i)$ and by γ^{k-1-i}. We can notice that the approximation errors in the previous iteration are more important while the earlier ones decreases exponentially fast. The second term also decreases exponentially fast and depends by $Q_{\max}$.

3.3.2 Finite-Samples Analysis

We provide here an upper-bound of the performance loss $\|Q^* - Q^{\pi_K}\|_{1,\rho}$. It is important to notice that this error indicates the regret of following the policy π_K instead the optimal one when the initial state-action is distributed according to ρ. In order to introduce the theoretical result provided in "Value Pursuit Iteration" Farahmand et al. 2012 [24] we must enumerate some assumptions:

1. The dataset $D_n^{(k)}$ is defined as in Section 3.1 except for the size of the dataset that is always fixed to n ($N_k = n$).
2. For any couple $k \neq k'$ where $k, k' \in \{1, \ldots K-1\}$, datasets $D_n^{(k)}$ and $D_n^{(k')}$ are independent.
3. There exists a constant $Q_{\max}$ such that for any $Q \in \mathcal{B}(\mathcal{X} \times \mathcal{A}; Q_{\max})$, $|\hat{T}^* Q(X, A)| \leq Q_{\max}$ for any $X \in \mathcal{X}, A \in \mathcal{A}$ with probability 1.
4. For all $g \in \mathcal{D}_0$, $\|g\|_\infty \leq L < \infty$.

5. The number of atoms m used from the dictionary $\mathcal{D}_0$ is $m = \lceil n^a \rceil$ for some finite $a > 0$. The numer of link functions m' used at each iteration is at most m/K.

6. At iteration k, each of the link functions $\{\sigma_i\}_{i=1}^{m'}$ maps $\beta_{Q_{\max}} Q_{k+1}$ and the dictionary $\mathcal{D}_k \cup \mathcal{D'}_k$ to an element of the space of vector-valued $Q_{\max}$-bounded measurable functions $\mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}^{|\mathcal{A}|}$. At least one of the mapping returns $\beta_{Q_{\max}} Q_{k+1}$.

Theorem 8. Finite-Sample Bound for VPI [24]
Consider the sequence $(Q_k)_{k=0}^K$ generated by VPI Algorithm 6. Let the assumptions stated above hold. For any fixed $0 < \delta < 1$, recursively define the sequence $(b_i)_{i=0}^K$ as follows:

$$b_0^2 \triangleq c_1 Q_{\max}^3 \sqrt{\frac{\log(\frac{nK}{\delta})}{n}} + 3 \inf_{Q' \in \mathcal{B}_{Q_{\max}}(\mathcal{L}_1(\mathcal{D}_{0,m};v))} \|Q' - T^* Q_0\|_n^2,$$

$$\begin{aligned} b_0^2 \triangleq\; & c_1 Q_{\max}^3 \sqrt{\frac{\log(\frac{nK}{\delta})}{n}} + \\ & c_2 \min\Big\{ \inf_{Q' \in \mathcal{B}_{Q_{\max}}(\mathcal{L}_1(\mathcal{D}_{0,m};v))} \|Q' - (T^*)^{k+1} Q_0\|_v^2 + C_1(k) \sum_{i=0}^{k-1} \gamma^{k-1} b_i^2, \\ & C_2 \left(\sum_{i=0}^{k-1} c_v(k-1-i) b_i^2 + \gamma^k (2Q_{\max})^2 \right) \Big\}, \quad (k \geq 1) \end{aligned}$$

for some $c_1, c_2, c_3 > 0$ that are only functions of $Q_{\max}$ and L. Then for any $k = 0, \ldots, K-1$, it holds that $\|Q_{k+1} - T^* Q_k\|_v^2 \leq b_k^2$, with probability at least $1 - \frac{k\delta}{K}$. Furthermore, define the discounted sum of errors as $\mathcal{E}(s) \triangleq \sum_{k=0}^{K-1} \alpha^{2s} b_k$ (for $s \in [0,1]$). Choose $\rho \in \mathcal{M}(\mathcal{X} \times \mathcal{A})$. The ρ-weighted performance loss of π_K is the upper-bounded as

$$\|Q^* - Q^{\pi_K}\|_{1,\rho} \leq \frac{2\gamma}{(1-\gamma)^2} \left[\inf_{s \in [0,1]} C_{VI,\rho,v}^{1/2}(K;s) \mathcal{E}^{1/2}(s) + 2\gamma^K Q_{\max} \right], \quad (3.16)$$

holds with probability $1 - \delta$.

Theorem 8 derives directly by the Theorem 6 where we redefine $\mathcal{E}$ as a linear combination of the approximation error committed at each iteration. Each approximation error is defined by b_k. The terms b_k are described with two terms: the first term $\sqrt{\frac{\log(\frac{nK}{\delta})}{n}}$ is the estimation error, while the second term describes the approximation error. The first term inside $\min\{\cdot,\cdot\}$ describes the error when we use the predefined dictionary $\mathcal{D}_{0,m}$ to approximate $T^* Q_k$ (this could be seen thanks to Theorem 5), the second term is given by Lemma 1 refers to the error approximation in the earlier iterations: in this way we consider only earlier Q_k to approximate T^*Q_k while VPI uses possibly other features that might lead a smaller function approximation error.

3.4 Regularized Fitted Q-Iteration

Regularized Fitted Q-Iteration (R-FQI) [22][2] is a variation of FQI that introduces regularization in the loss function. Regularization [68] is a term that applies to the loss function in the case parametric regression, penalizing parameters with high values:

$$\theta = \arg\min_{\theta \in \Theta} \rho(f_\theta, \mathbf{y}) + \lambda \|\theta\|_p^p \tag{3.17}$$

where θ is a vector of parameters. This is done in order to “restrict” the functional space, and thus to reducing the overfitting. The main idea in R-FQI it to use a very complex model (in theory a universal functional approximator), and to find a regularization coefficient that could well balance overfitting and bias in such way that during each iteration we have a good approximation of the action-state value function. The advantage is that, while with standard FQI we must choose the right complexity of the functional space in order that it is enough expressive to represent the action-state value function and enough simple to do not overfit, in Regularized FQI we must choose a functional space enough expressive, but without restrictions. The question now is on how to choose a regularization coefficient in order to acheive a good approximation of the action-state value function.

We define here the loss function in the case of R-FQI:

$$\rho_k(Q) = \frac{1}{N_j} \sum_{i=1}^{N} [r_i^{(k)} + \gamma \max_{a' \in \mathcal{A}} Q_k(x'^{(k)}_i, a') - Q(x, a)]^2. \tag{3.18}$$

Let us suppose that $Q \in \mathcal{H}$ where $\mathcal{H}$ is an Hilbert space. Let the norm of Q in $\mathcal{H}$ be the penalization and ker the Mercer kernel function. At iteration each iteration k we must find:

$$Q_{k+1} = \arg\min_{Q \in \mathcal{H}} \left[\rho_k(Q) + \lambda \|Q\|^2_{\mathcal{H}}\right]. \tag{3.19}$$

[2] The reader who wish to deepen regularization in RL, could read Farahmand’s Phd dissertation “Regularization in Reinforcement Learning” [21]

According to the Representer Theorem (for further information read Scholkopf et al. [69]) we represent Q_k as:

$$Q_k(x,a) = \sum_{i=1}^{N} \alpha_i^{(k)} \mathrm{ker}((x_i^{(k)}, a_i^{(k)}), (x,a)) \tag{3.20}$$

where the vector $\alpha^k \in \mathbb{R}^{N_k}$ is the vector of the parameters at the k-th iteration and $\alpha_i^{(k)}$ is the i-th item of such vector. We can now rewrite Equation 3.19:

$$\boldsymbol{\alpha}^{k+1} = \arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^{N_k}} \frac{1}{N_k} \left\| \mathbf{r}_j + \gamma \mathbf{K}^+ \boldsymbol{\alpha}^{(k)} - \mathbf{K}\boldsymbol{\alpha} \right\| + \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \tag{3.21}$$

with:

$$\begin{aligned} [\mathbf{K}]_{h,l} &= \mathrm{ker}(Z_{j,h}, Z_{j,l}), && (3.22) \\ [\mathbf{K}^+]_{h,l} &= \mathrm{ker}(Z_{j,h}^{(j)}, Z_{j,l}), && (3.23) \end{aligned}$$

where $Z_{j,i} = (X_{i^{(k)}}, A_i^{(k)})$, $Z_{j,i}^{(j)} = (X'^{(k)}_i, A_i^{(k)})$,

$$A_{k,i}^{(k)} = \arg\max_{A \in \mathcal{A}} Q_k(X'^{(k)}_i, A)$$

and

$$\mathbf{r} = (R_1^{(k)}, \ldots, R_{N_k}^{(k)})^T.$$

3.4.1 Error Propagation in R-FQI

Like we have done previously in Section 3.2 we derive here the upperbound of the error propagation. We use the same definition of approximation error ϵ_k in Equation (3.1). For the sake of flexibility, we allow the user to choose another distribution, $v \in \mathcal{M}(\mathcal{X})$, that is used in assessing the procedure's performance, e.g. the stationary distribution induced by the optimal policy.

Theorem 9. L^p-Bound in R-FQI [22]
Consider a discounted MDP with a finite number of actions. Let $p > 1$. Assume that both Q_k and ϵ_k satisfy Equation (3.1) *and that π_k is a greedy policy with respect to Q_k. Fix $\overline{k} > 0$. Define $E_0 = \|\epsilon_{-1}\|_\infty$ and $\overline{\epsilon}_{\overline{k}} = \max_{0 \le k \le \overline{k}} \|\epsilon_k\|_{p,v}$. Then there exist constants $C_{\mu,v}^{(1,1)}$ and $C_{\mu,v}^{(2,1)}$ that only depend on μ, v, γ and the MDP dynamics such that*

$$\|V^* - V^{\pi_{\overline{k}}}\|_{p,\mu} \le \frac{2}{1-\gamma} \left[\gamma^{\frac{\overline{k}}{p}} E_0 + \left((1-\gamma) C_{\mu,v}^{(1,1)\,\frac{1}{p}} + \gamma C_{\mu,v}^{(2,1)\,\frac{1}{p}} \right) \overline{\epsilon}_{\overline{k}} \right]. \tag{3.24}$$

Theorem 9 shows that the difference between V^* and $V^{\pi_{\overline{k}}}$ is bounded by two terms. The first term $\gamma^{\frac{\overline{k}}{p}} E_0$ says that the error is bounded by the magnitude of the first error between Q^* and Q_0 but it exponentially decreases in the number of iteration. The second term says that the error both depends by the magnitude of the worst approximation error and by some constant that depends by the distributions μ and v. Theorem 9 suggests that we should minimize $\overline{\epsilon}_{\overline{k}}$, and thus we must choose an appropriate λ (see Equation (3.19)).

3.4.2 Finite-Samples Analysis

The following finite-samples bound can be obtained generalizing Theorem 21.1 of [35] to an arbitrary reproducing kernel Hilbert space combining it with Proposition 3 of [83]. In order to simplify the proof of the theorem Farahmand et al., we made assumption of $\mathcal{X} = [0,1]^d$ and to have access of a generative model. However, those assumptions are not essential and the theorem could be extended to more general cases.

Theorem 10. R-FQI Approximation Error [22]
Assume $\mathcal{X} = [0,1]^d, \mathrm{ker} \in \mathrm{Lip}^*(s, C(\mathcal{X}, \mathcal{X}))$, $s > d$ and Q_k is such that $TQ_k \in \mathcal{H}$. Furthermore, (for the sake of simplicity) we assume that all the functions involved in the regression problem (the reward function, Q_k, and the result of the optimization problem Q_{k+1}) are bounded by some constant $L > 0$. Let Q_{k+1} be a solution of (3.19) with some $\lambda > 0$. Then

$$\|Q_{k+1} - TQ_k\|_v^2 \leq 2\lambda \|TQ_k\|_{\mathcal{H}}^2 + \frac{c_1 L^4}{N_k \lambda^{d/s}} + \frac{c_2 \log(1/\delta)}{N_k L^4} \tag{3.25}$$

holds with probability at least $1 - \delta$, for some $c_1, c_2 > 0$.

We can see that as λ increases, the first term increases too but the second decreases (as d is positive and $s > d$). We can see that with $\lambda = N_k^{-1/(1+d/s)}$ the rate convergence is $O(N_k^{-1/(1+d/s)})$ and this is the optimal rate.

3.5 Introduction to B-FQI

Boosted Fitted Q-Iteration (B-FQI) is an iterative offline algorithm that, given a dataset of transitions, builds an approximation of the optimal action-state value function by summing approximations of the Bellman residuals at each iteration. The main advantage of such algorithm with respect to ordinary AVI methods is that the Bellman residual is an "easy" function to approximate, and this could be indirectly shown by showing that B-FQI is a boosting procedure. Introducing boosting in FQI gives us the typical advantage of Boosting methods which consists in dynamically increasing the functional expressivity following the descent direction of the gradient of the loss. Another advantage is given by the combination of FQI and boosting. In classical supervised learning setting there is no clear advantage in terms of time in using boosting because even if it can exploit weak regressors, it also introduces *iterations*. B-FQI instead gives a clear advantage in terms of time because every AVI method has to perform *iterations* by definition, thus using a weak regressor instead a more complex regressor must be advantageous in terms of time. The advantage is particularly clear when B-FQI approaches to the optimal value function: the bellman residual tends to be zero, while when FQI as described in [20] approaches to the optimal value function, it must approximate at each iteration the whole action-state value function. It is interesting that B-FQI can be considered the counter-side of R-FQI since that B-FQI increase its functional space complexity, while R-FQI starts from a complex functional space, and restricts it by means of regularization.

3.6 Theoretical Analysis of B-FQI

Like in Section 3.4 we need to formalize how Q_{k+1} approximates T^*Q_k. In order to do that we need to introduce a *non-linear* operator $S : \mathcal{B}(\mathcal{X} \times \mathcal{A}) \to \mathcal{F}$ which projects an action-state value function onto a functional space $\mathcal{F} \subset \mathcal{B}(\mathcal{X} \times \mathcal{A})$. The projection operator is defined as:

$$Sy = \arg\inf_{f \in \mathcal{F}} \|f - y\|_\mu^2, \quad y \in \mathcal{B}(\mathcal{X} \times \mathcal{A}). \tag{3.26}$$

The projection error is required to be bounded linearly with the magnitude of the target:

$$\exists \chi > 0 : \|(I - S)y\|_\mu \leq \chi \|y\|_\mu \quad \forall y \in \mathcal{B}(\mathcal{X} \times \mathcal{A}). \tag{3.27}$$

Let ϱ_k be the Bellman Residual:

$$\varrho_k \triangleq T^*Q_k - Q_k. \tag{3.28}$$

The estimate Q_{k+1} built by B-FQI is a generalized additive model [36]:

$$Q_{k+1} = Q_k + S\varrho_k = \sum_{i=0}^{k} S\varrho_i, \tag{3.29}$$

Algorithm 7: Boosted Fitted Q-Iteration

```
Input: $(\mathcal{D}_n^{(i)})_{i=0}^K$, $(\beta_{B_i})_{i=0}^K$, $Q_0 = 0$;
for $k = 0, \dots, K$ do
    $\tilde{\varrho}_k \leftarrow \hat{T}Q_k - Q_k$ (w.r.t. $\mathcal{D}_n^{(k)}$);
    $Q_{k+1} \leftarrow Q_k + \beta_{B_k} \arg\inf_{f \in \mathcal{F}} \|f - \tilde{\varrho}_k\|_{\mathcal{D}_n^{(k)}}$;
end for
return $\bar{\pi}(x) = \arg\max_a Q_{K+1}(x, a) \quad \forall x \in \mathcal{X}$;
```

obtained by fitting the Bellman residual at each iteration. Without loss of generality we assume $Q_0(x,a) = 0$ for any $(x,a) \in \mathcal{X} \times \mathcal{A}$.

3.6.1 Error Propagation

Like we have done previously for AVI R-FQI and for VPI, we bound the error $\|Q_{k+1} - Q^*\|_\infty$ in terms of the previous error $\|Q_k - Q^*\|_\infty$. This error bound is done very easily considering Equation (3.29) and the bound on the projection operator S (Equation (3.27)), moreover we keep account of the max-norm contraction property of the Optimality Bellman operator already introduced in Chapter 2.

Theorem 11. Propagation Error
Let $(Q_i)_{i=0}^k$ be a sequence of measurable action-value functions obtained following the boosted procedure described in Equation (3.29) *and let $L = \gamma C_{\mu\to\infty}$. Then, under assumption made in Equation* (3.27)

$$\|Q_k - Q^*\|_\mu \le ((1+L)\chi + L)\,\|Q_{k-1} - Q^*\|_\mu .$$

Proof.

$$\begin{aligned}
\|Q_k - Q^*\|_\mu &= \|Q_k - T^*Q_{k-1} + T^*Q_{k-1} - Q^*\|_\mu \\
&\le \|Q_{k-1} + S\varrho_{k-1} - T^*Q_{k-1}\|_\mu + \|T^*Q_{k-1} - Q^*\|_\mu \\
&\le \chi\,\|\varrho_{k-1}\|_\mu + L\,\|Q_{k-1} - Q^*\|_\mu \qquad (3.30)\\
&\le (1+L)\chi\,\|Q_{k-1} - Q^*\|_L + L\,\|Q_{k-1} - Q^*\|_\mu \qquad (3.31)
\end{aligned}$$

where (3.30) follows (3.27) and (3.7) while inequality (3.31) is a consequence of the fact that

$$\begin{aligned}
\|\varrho_k\|_\mu &\le \|T^*Q_k - T^*Q^*\|_\mu + \|T^*Q^* - Q_k\|_\mu \\
&\le (1+L)\,\|Q_k - Q^*\|_\mu \qquad (3.32)
\end{aligned}$$

that again is a consequence of (3.7). □

First of all, notice that when $S = I$ (i.e., $\chi = 0$) we correctly obtain the usual convergence rate of value iteration. On the other cases, similarly to SL [14], we can still converge to the target (here Q^*) given that the operator $I - S$ is sufficiently contractive.

Corollary 1. *Given the settings of Theorem 11, the sequence $(Q_i)_{i=0}^k$ converges to Q^* when*

$$\chi < \frac{1 - \gamma C_{\mu\to\infty}}{1 + \gamma C_{\mu\to\infty}}.$$

and

$$\gamma C_{\mu\to\infty} < 1.$$

Corollary 1 shows us that if the approximator S has a relative error $\chi < \frac{1-\gamma C_{\mu\to\infty}}{1+\gamma C_{\mu\to\infty}}$ and combined with a concentrability measure $C_{\mu\to\infty} < 1/\gamma$, then B-FQI converges.

3.6.2 $\tau-$Greedy Policies

Recently, Munos et al. [54] have analyzed the use of τ-greedy polices for control purposes in off-policy learning. Inspired by such paper, by exploiting their definition in l_∞-norm, we show that is possible to use τ-greedy policies in AVI. In our work we provided error propagation bounds both for general AVI and for B-FQI. We introduce the non-greedy policy π_k which is τ-far away from the greedy policy with respect to Q_k:

Definition 11. τ-greedy policy [54]
Consider a non greedy policy π_k which is τ far away from the greedy policy with respect to Q_k:

$$T^{\pi_k} Q_k \geq T^* Q_k - \tau_k \|Q_k\|_\infty \boldsymbol{e} \tag{3.33}$$

where $\boldsymbol{e}$ is the vector with 1-components.

τ-greedy policies. We show here a bound for non-greedy policies in the general case of AVI. As far as we know this bound is not present yet in the literature:

Theorem 12. *Consider $\xi_k = T^{\pi_k} Q_k - Q_{k+1}$ as the approximation error with respect to the bellman operator π_k where π_k is τ-greedy (Equation (3.33)). The error $Q_{k+1} - Q^*$ is bounded by:*

$$\|Q_{k+1} - Q^*\|_\infty \leq \|\xi_k\|_\infty + \tau_k \|Q_k\|_\infty + \gamma \|Q_k - Q^*\|_\infty. \tag{3.34}$$

Proof. The proof could be split in the upperbound and lowerbound derivation, which are completely symmetric. For sake of completeness we report here both.

Upper bound

$$\begin{aligned}
Q_{k+1} - Q^* &= Q_{k+1} - T^{\pi_k} Q_k + T^{\pi_k} Q_k - Q^* \\
&\leq (\|Q_{k+1} - T^{\pi_k} Q_k\|_\infty \; + \|T^{\pi_k} Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&= (\|\xi_k\|_\infty + \|T^{\pi_k} Q_k - T^* Q_k + T^* Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&\leq (\|\xi_k\|_\infty + \|T^{\pi_k} Q_k - T^* Q_k\|_\infty + \|T^* Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&\leq (\|\xi_k\|_\infty + \tau_k \|Q_k\|_\infty + \gamma \|Q_k - Q^*\|_\infty)\boldsymbol{e}. \qquad (3.35)
\end{aligned}$$

Lower bound

$$\begin{aligned}
Q_{k+1} - Q^* &= Q_{k+1} - T^{\pi_k} Q_k + T^{\pi_k} Q_k - Q^* \\
&\geq -(\|Q_{k+1} - T^{\pi_k} Q_k\|_\infty + \|T^{\pi_k} Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&= -(\|\xi_k\|_\infty + \|T^{\pi_k} Q_k - T^* Q_k + T^* Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&\geq -(\|\xi_k\|_\infty + \|T^{\pi_k} Q_k - T^* Q_k\|_\infty + \|T^* Q_k - Q^*\|_\infty)\boldsymbol{e} \\
&\geq -(\|\xi_k\|_\infty + \tau_k \|Q_k\|_\infty + \gamma \|Q_k - Q^*\|_\infty)\boldsymbol{e}. \qquad (3.36)
\end{aligned}$$

The last passages Equations (3.35), (3.36) are derived by the max-norm contraction property of the bellman operator, and thanks to Equation (3.33). Now we can see that both the bounds derive the same bounds point-wisely. We can thus same that:

$$\|Q_{k+1} - Q^*\|_\infty \leq \|\xi_k\|_\infty + \tau_k \|Q_k\|_\infty + \gamma \|Q_k - Q^*\|_\infty \qquad (3.37)$$

which is the statement of the theorem. □

Theorem 12 shows that the error between Q_{k+1} and Q^* could be bounded the composition of three terms: the approximation error $\|\xi_k\|_\infty$, the τ-distance of the non-greedy policy, and the previous error $\|Q_k - Q^*\|_\infty$.

We can derive also the relative bound for B-FQI:

Theorem 13. Propagation Error with non-greedy policies for B-FQI *Consider a sequence of policies $(\pi_i)_{i=0}^k$ that are non-greedy w.r.t. the sequence $(Q_k)_{i=0}^k$ of Q-functions obtained following the boosting procedure*

$$Q_k = Q_{k-1} + S\eta_{k-1} \qquad (3.38)$$

where $\eta_k \triangleq T^{\pi_k} Q_k - Q_k$ is the residual computed with the non-greedy Bellman operator. Assume the policies π_k are τ_k-away from the greedy policy w.r.t.

Q_k, so that $T^{\pi_k}Q_k \geq T^*Q_k - \tau_k \left\| Q_k \right\|_\infty \boldsymbol{e}$, where $\boldsymbol{e}$ is the vector with 1-components. Then for any $k > 0$

$$\left\| Q_k - Q^* \right\|_\infty \leq \left\| (S - I)\eta_{k-1} \right\|_\infty + \gamma \left\| Q_{k-1} - Q^* \right\|_\infty + \tau_k \left\| Q_{k-1} \right\|_\infty . \quad (3.39)$$

Proof. We split here the proof in two parts: the first part is devoted to upperbound $Q_k - Q^*$ and the second one to lowerbound the same quantity. **Lower bound.** Notice that

$$\begin{aligned} Q_k - Q^* &= Q_k \pm T^{\pi_{k-1}}Q_{k-1} \pm T^{\pi^*}Q_{k-1} - Q^* \\ &\geq Q_k - T^{\pi_{k-1}}Q_{k-1} - \tau_{k-1} \left\| Q_{k-1} \right\|_\infty \boldsymbol{e} \\ &\quad + \gamma P^{\pi^*}(Q_{k-1} - Q^*) \qquad (3.40) \\ &= (S - I)(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) - \tau_{k-1} \left\| Q_{k-1} \right\|_\infty \boldsymbol{e} \\ &\quad + \gamma P^{\pi^*}(Q_{k-1} - Q^*), \qquad (3.41) \end{aligned}$$

where (3.40) follows from the definition of T^{π^*} and τ-away from the greedy policy

$$T^{\pi_k}Q_k \geq T^*Q_k - \tau_k \left\| Q_k \right\|_\infty \boldsymbol{e} \geq T^{\pi^*}Q_k - \tau_k \left\| Q_k \right\|_\infty \boldsymbol{e},$$

while (3.41) is proved from

$$\begin{aligned} Q_{k+1} - T^{\pi_k}Q_k &= Q_k + S(T^{\pi_k}Q_k - Q_k) - T^{\pi_k}Q_k \\ &= (S - I)(T^{\pi_k}Q_k - Q_k). \end{aligned}$$

Upper bound. Let us derive an upper bound of the same quantity. Let $\bar{\pi}_k$ be the greedy policy associated to Q_k, then $T^{\bar{\pi}_k}Q_k \geq T^{\pi}Q_k$, for any policy π.

$$\begin{aligned} Q_k - Q^* &= Q_{k-1} + S(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) - Q^* \\ &= Q_{k-1} \pm T^{\pi_{k-1}}Q_{k-1} + S(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) - Q^* \\ &= (S - I)(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) + T^{\pi_{k-1}}Q_k - Q^* \\ &\leq (S - I)(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) \\ &\quad + T^{\bar{\pi}_{k-1}}Q_{k-1} - T^{\bar{\pi}_{k-1}}Q^* \\ &= (S - I)(T^{\pi_{k-1}}Q_{k-1} - Q_{k-1}) \\ &\quad + \gamma P^{\bar{\pi}_{k-1}}(Q_{k-1} - Q^*) \qquad (3.42) \end{aligned}$$

since $T^{\pi}Q^* \leq Q^*$ for any π. Combining the above (3.42) with (3.41) we derive the result. □

We can compare here the results of Theorem 13 with the relative Theorem 12. The results is coherent if wee keep in account that:

$$\begin{aligned}
\|\xi_{k-1}\|_\infty &= \|T^{\pi_{k-1}}Q_{k-1} - Q_k\|_\infty \\
&= \|T^{\pi_{k-1}}Q_{k-1} - (Q_{k-1} + S(T^{\pi_k}Q_{k-1} - Q_{k-1}))\|_\infty \quad (3.43) \\
&= \|\eta_{k-1} - S\eta_{k-1}\|_\infty \\
&= \|(S - I)\eta_{k-1}\|_\infty .
\end{aligned}$$

Passage (3.43) is the application of the definition of Q_k in this contest (Equation (3.38)). This result plays the same role of Theorem 11 and shows that by behaving τ-greedly we have a linear additive cost proportional to τ. The advantage respect the general AVI procedure is that η_k should tend to zero when B-FQI converges, thus, also $(S - I)\eta_k$ should converge for how operator S is defined and bounded. ξ_k could be defined, for a general FQI procedure, as $\xi_k = T^{\pi_k}Q_k - ST^{\pi_k}Q_k$, and for how S is define, there is no guarantee that ξ_k converges to zero.

Approximation of the residual. While previous results were somehow expected to hold as a consequence of the results in SL, we now show how the approximation error due to the fitting of the Bellman residual propagates. Until now we used S to represent the projection of the Bellman Residual onto the functional space $\mathcal{F}$ by minimizing a norm computed under the distribution μ. This operator is somehow theoretical, in reality we must use an operator that minimizes the norm computed on samples. We introduce here the *nonlinear* operator $\hat{S} : \mathcal{B}(\mathcal{X} \times \mathcal{A}) \to \mathcal{F}$ which represents the regression step, and the truncation operator β_{B_k} which is a truncation operator:

$$\hat{S}y = \arg\inf_{f\in\mathcal{F}} \|f - y\|^2_{D^{(k)}_{N_k}} . \tag{3.44}$$

We define here $\hat{\varrho}$ as the truncated projected empirical Bellman residual (as defined in Equation 3.6):

$$\begin{aligned}
\hat{\varrho} &= \beta_{B_k}\hat{S}\tilde{\varrho}_k \\
&= \beta_{B_k} \arg\inf_{f\in\mathcal{F}} \|f - \tilde{\varrho}_k\|^2_{D^{(k)}_{N_k}} \\
&= \beta_{B_k} \arg\inf_{f\in\mathcal{F}} \sum_{i=1}^{N_k} \frac{1}{N_k} \left| f(X_i^{(k)}, A_i^{(k)}) - \tilde{\varrho}_k(X_i^{(k)}, A_i^{(k)}) \right|^2 , \quad (3.45)
\end{aligned}$$

which is used to update the approximation of T^*Q_k:

$$Q_{k+1} = Q_k + \beta_{B_k} \hat{S} \tilde{\varrho}_k = \sum_{i=0}^{k} \hat{\varrho}_i, \quad Q_{k+1} \in \mathcal{H}_{k+1}. \tag{3.46}$$

Now we have a complete definition of B-FQI algorithm, taking in account all the sources of error (Algorithm 7). Note that the introduction of the truncated projected Bellman residual $\hat{\varrho}_k \in \mathcal{B}(\mathcal{X} \times \mathcal{A}, B_k)$ is required for theoretical guarantees, while $\mathcal{H} \subset \mathcal{B}(\mathcal{X} \times \mathcal{A})$ is a function space that possibly increases its complexity over time. This increment can be seen as a procedure of altering the underlying function space and, potentially, increasing the richness of $\mathcal{H}_k$ at each iteration. Now suppose that our function space $\mathcal{F}$ is Glivenko-Cantelli, i.e., the error due to the empirical process goes to zero at least asymptotically. The preservation theorem [76] states that, under mild assumptions, the space obtained by the sum of Glivenko-Cantelli functions is still Glivenko-Cantelli. This means that if we start from a sufficiently powerful functional space, the boosting procedure at least preserves its properties. Although this does not provide any insight about the "increased" complexity of $\mathcal{H}_k$, it shows the soundness of boosting. In practice this means that B-FQI is able to learn complex, nonparametric approximations of Q^* over time. Additionally, the update procedure is computationally efficient since it can rely on specialized batch algorithms available for several regression techniques. In SL the boosting procedure comes at an increased computational cost since it should estimate $k > 1$ regressors. Even if regression tasks become simpler at each successive iteration, the complexity is proportional to the number of steps [14]. In our settings, we enjoy the benefits of exploiting a richer approximation space, without paying any additional cost, since the number of regression tasks is the same as in the other Fitted Value Iteration methods. In particular, we can see B-FQI as a single boosting procedure with time-varying target: $Y_{k+1} = T^* Q_k$ (while in SL the target is fixed). This aspect prevents to directly reuse results from SL. We want to show that every iteration of B-FQI is equivalent to L_2-boost procedure:

Theorem 14. Each iteration of B-FQI is equivalent to one L_2-Boost procedure
Let

$$Q_{k+1} = Q_k + \beta_{B_k} \hat{S} \varrho_k \tag{3.47}$$

where at each iteration k, $\hat{T}^* Q_k$ is the target, *Q_k is the actual approximation, and $\beta_{B_k} \hat{S}$ is the base procedure; then Q_{k+1} is equivalent to one-step L_2-boost.*

Proof. Let us define the L_2-loss as

$$\rho(y,f) = \frac{1}{2}(y-f)^2 \tag{3.48}$$

Since our target is $\hat{T}^*Q_k(\mathbf{x},\mathbf{a})$ where $\mathbf{x} = (X_1^{(k)},\dots,X_{N_k}^{(k)})$ and $\mathbf{a} = (A_1^{(k)},\dots,A_{N_k}^{(k)})$, The Functional Gradient Descent described by Algorithm 4 requires to compute

$$\mathbf{u} = -\frac{\partial}{\partial f}\frac{1}{2}(\hat{T}^*Q_k(\mathbf{x},\mathbf{a}) - f)^2|_{f=Q_k(\mathbf{x},\mathbf{a})} \tag{3.49}$$

yields

$$\mathbf{u} = \hat{T}^*Q_k(\mathbf{x},\mathbf{a}) - Q_k. \tag{3.50}$$

The boosting algorithm requires to approximate the residual $\mathbf{u}$ with a base procedure $\hat{S}$, and then add the approximation to the actual model:

$$Q_{k+1} = Q_k + \beta_{B_k}\hat{S}\mathbf{u}. \tag{3.51}$$

As we can notice $\mathbf{u} = \varrho_k$ and Equation (3.51) is equivalent to Equation (3.47). As our loss was a L_2 loss, our approximation is equivalent to one step of L_2-Boost. □

Theorem 14 shows that B-FQI performs at every iteration a step L_2-Boost, which does not means that the overall algorithm perform a L_2-Boost procedure, because at each iteration the target $\hat{T}^*Q_k$ changes. We can also notice that when B-FQI converges, $\hat{T}^*Q_k \approx \hat{T}^*Q_{k+1}$ which means that the target is almost always the same, thus, approaching the convergence, B-FQI behaves approximately like a L_2-Boost algorithm.

For a sequence $(Q_i)_{i=0}^k$ denotes the *approximation error* as:

$$\epsilon_k \triangleq \varrho_k - \beta_{B_k}\hat{S}\tilde{\varrho}_k, \tag{3.52}$$

so that $Q_{k+1} = T^*Q_k - \epsilon_k$. The result we are going to provide is the boosted counterpart of Theorem 5. Here we want to bound the minimum error that we commit by projecting the residual $T^*Q_k - Q_k$ in the functional space $\mathcal{F}$. This result is very important because it gives us also a way to later bound the error with the empirical norm $\|\cdot\|_{D_{N_k}^{(k)}}$.

Theorem 15. *Let $(Q_i)_{i=0}^{k-1}$ be a sequence of state-action value functions, $(\epsilon_i)_{i=0}^{k-1}$ be the corresponding sequence as defined in* (3.52). *Define $\varrho_k^* \triangleq$*

$(T^*)^{k+1}Q_0 - (T^*)^k Q_0$ and $L = \gamma C_{\mu\to\infty}$. Let $\mathcal{F} \subseteq \mathcal{B}(\mathcal{X}\times\mathcal{A})$ be a subset of measurable functions. Then,

$$\begin{aligned}\inf_{f\in\mathcal{F}} \|f-(T^*Q_k - Q_k)\|_\mu &\le \inf_{f\in\mathcal{F}} \|f-\varrho_k^*\|_\mu \\ &\quad +(1+L)\sum_{i=0}^{k-1} L^{k-1-i}\|\epsilon_i\|_\mu . \qquad (3.53)\end{aligned}$$

Proof. In order to bound $\inf_{f\in\mathcal{F}}\|f-\varrho_k\|_\mu$ we pick any $f\in\mathcal{F}$ and by triangle inequality we have that:

$$\|f-\varrho_k\|_\mu \le \|f-\varrho_k^*\|_\mu + \|\varrho_k^*-\varrho_k\|_\mu . \qquad (3.54)$$

Since by [21], T is $L \triangleq \gamma C_{\mu\to\infty}$-Lipschitz w.r.t. $\|\cdot\|_\mu$, we can bound $\|\varrho_k^*-\varrho_k\|_\mu$ as follows:

$$\begin{aligned}\|\varrho_k^*-\varrho_k\|_\mu &\le \left\|(T^*)^{k+1}Q_0 - T^*Q_k\right\|_\mu + \left\|(T^*)^kQ_0 - Q_k\right\|_\mu \\ &\le L\left\|(T^*)^kQ_0 - Q_k\right\|_\mu + \left\|(T^*)^kQ_0 - Q_k\right\|_\mu \\ &= (1+L)\left\|(T^*)^kQ_0 - (T^*Q_{k-1}+\epsilon_{k-1})\right\|_\mu \\ &\le (1+L)\left(\left\|(T^*)^kQ_0 - T^*Q_{k-1}\right\|_\mu + \|\epsilon_{k-1}\|_\mu\right) \\ &\le (1+L)\left(L\left\|(T^*)^{k-1}Q_0 - Q_{k-1}\right\|_\mu + \|\epsilon_{k-1}\|_\mu\right) \\ &\le (1+L)\left(L\left(L\left\|(T^*)^{k-2}Q_0 - Q_{k-2}\right\|_\mu + \|\epsilon_{k-2}\|_\mu\right)\right. \\ &\quad \left. + \|\epsilon_{k-1}\|_\mu\right) \\ &\le \ldots \le (1+L)\sum_{i=0}^{k-1} L^{k-1-i}\|\epsilon_i\|_\mu . \qquad (3.55)\end{aligned}$$

The statement follows directly from the combination of inequalities (3.54) and (3.55). □

Previous theorem shows how the approximation error of the Bellman residual in the boosted scenario relates to the Bellman residual of Value Iteration (ϱ_k^*) and the errors in earlier iterations. We can see that the error $\|\epsilon_k\|_\mu^2$ is bounded by $\inf_{f\in\mathcal{F}}\|f-\varrho_k^*\|_\mu^2$ which is the minimum error committed by projecting ϱ_k^* onto the functional space $\mathcal{F}$ by minimizing the squared norm $\|\cdot\|_\mu$. The interesting part is that ϱ_k converges to zero for $k\to 0$, which suggests us that our functional space should be able to well approximate functions around zero. $\|\epsilon_k\|_\mu^2$ is also bounded by a linear combinations of $\|\epsilon_j\|_\mu^2$ which suggests us (with no surprise) that previous approximation error should be as small as possible to achieve a good approximation, and also $C_{\mu\to\infty}$ (which is implicitly hidden in L) should be preferable close to zero. This bound will play an important role in the derivation of the finite-sample error bound (Theorem 16).

3.6.3 Finite-Sample Error Analysis

In this section, we derive an upper bound to the difference between the performance of the optimal policy and the performance of the policy learned by B-FQI at the k-th iteration. Such upper bound depends on properties of the MDP, properties of the approximation space and the number of samples.

Since B-FQI is an AVI algorithm, we can bound the performance loss at iteration k ($\|Q^* - Q^{\pi_k}\|_{1,\rho}$) using Theorem 6.

Although the bound in Theorem 6 is shared by all the AVI approaches (e.g., FQI, VPI, B-FQI), for each approach is possible find specific bounds of the approximation errors ϵ_k made at each iteration k. The following theorem provides an upper bound to $\|\epsilon_k\|_\mu^2$ for the case of B-FQI:

Theorem 16. *Let $(Q_i)_{i=0}^k$ be the sequence of state-action value functions generated by B-FQI using at each iteration i a dataset $\mathcal{D}_n^{(i)} = \{X_s^{(i)}, A_s^{(i)}, R_s^{(i)}, X_s'^{(i)}\}_{s=1}^n$ with i.i.d. samples $(X_s^{(i)}, A_s^{(i)}) \sim \mu$, $X_s'^{(i)} \sim P(\cdot|X_s^{(i)}, A_s^{(i)})$ and $R_s^{(i)} \sim \mathcal{R}(\cdot|X_s^{(i)}, A_s^{(i)})$ for $s = 1, 2, \ldots, n$, where each dataset $\mathcal{D}_n^{(i)}$ is independent from the datasets used in other iterations[3]. Let $\epsilon_i \triangleq \varrho_i - \hat{\varrho}_i$ $(0 \le i \le k)$, $\varrho_k^* \triangleq (T^*)^{k+1}Q_0 - (T^*)^k Q_0$, and $\tilde{\varrho}_k \triangleq \hat{T}^* Q_k - Q_k$. Let $\mathcal{F} \subseteq \mathcal{B}(\mathcal{X}, \mathcal{A})$ be a subset of measurable functions. Then,*

$$\|\epsilon_k\|_\mu^2 \le 4 \inf_{f\in\mathcal{F}} \|f - \varrho_k^*\|_\mu^2 + 4(1+L)^2 \sum_{i=0}^{k-1} L^{2i} \sum_{j=0}^{k-1} \|\epsilon_j\|_\mu^2$$
$$+ \frac{24 \cdot 214 B_k^4}{n} \left(\log 42e + 2\log(480eB_k^2 n)V_{\mathcal{F}^+}\right)$$

where $L = \gamma C_{\mu\to\infty}$, $B_k = \max(\|\tilde{\varrho}_k\|_\infty, 1)$, and $V_{\mathcal{F}^+}$ is the VC dimension of $\mathcal{F}^+$ that is the class of all subgraphs of functions $f \in \mathcal{F}$ (see Chapter 9.4 of [35]).

Proof. Since by previous definitions $\|\epsilon_k\|_\mu^2 = \|\varrho_k - \hat{\varrho}_k\|_\mu^2$ and $\hat{\varrho}_k = \beta_{B_k}\hat{S}\tilde{\varrho}_k = \beta_{B_k} \arg\inf_{f\in\mathcal{F}} \|f - \tilde{\varrho}_k\|_{\mathcal{D}_n^{(k)}}^2$, and given that $|\tilde{\varrho}_k| \le B_k = \max(\|\tilde{\varrho}_k\|_\infty, 1)$, we

[3] The indipendence of the datasets at different iterations can be relaxed as done in Section 4.2 of Munos et al. 2008 [55].

can use Theorem 11.5 in [35] to upper bound the above regression error as follows:

$$
\begin{aligned}
\|\epsilon_k\|_\mu^2 &= \|\varrho_k - \hat{\varrho}_k\|_\mu^2 \\
&\leq 2 \inf_{f\in\mathcal{F}} \|f - \varrho_k\|_\mu^2 \\
&\quad + \frac{24 \cdot 214 B^4}{n} \left(\log 42e + 2\log(480eB^2 n) V_{\mathcal{F}^+}\right).
\end{aligned}
$$

Using Theorem 15 and the Cauchy-Schwartz inequality to bound the first term completes the proof. □

The above theorem shows that the error of B-FQI at each iteration k can be bounded by the sum of three main terms, that, respectively, are: the *approximation* error in function space $\mathcal{F}$ of the Bellman error at the k-th iteration of VI, the *propagation* error that depends on the errors at previous iterations, and the *estimation* error induced by having a finite number of samples. The main differences between this result and related results presented in [23, 21, 24] are in the approximation and estimation errors. In B-FQI, ϱ_k^* and $\|\tilde{\varrho}_k\|_\infty$ take the role played, respectively, by $(T^*)^k Q_0$ and $Q_{\max}$ in other FVI approaches, enjoying the advantage of being bounded by smaller constants. For what concerns ϱ_k^*, assuming that Q_0 is initialized at zero for any state-action pair, it is known that $\|\varrho_k^*\|_\infty \leq \gamma^k R_{\max}$. In order to upper bound $\|\tilde{\varrho}_k\|_\infty$ we first introduce the following Lemma, that bounds the supremum norm of the Bellman residuals at iteration k.

Lemma 2. *Let* $(Q_i)_{i=0}^{k-1}$ *be a sequence of state-action value function,* $(\epsilon_i)_{i=0}^{k-1}$ *be the corresponding sequence as defined in* (3.52)*, then*

$$
\|\varrho_k\|_\infty \leq (1+\gamma) \sum_{i=0}^{k-1} \gamma^{k-i-1} \|\epsilon_i\|_\infty + \gamma^k R_{\max}.
$$

Proof.

$$
\begin{aligned}
\|\varrho_k\|_\infty &= \|T^* Q_k - Q_k\|_\infty \\
&= \sup_{x,a} \left| r(x,a) + \gamma \int_{\mathcal{X}} P(dy|x,a) \max_{a'} Q_k(y,a') - Q_k(x,a) \right| \\
&= \sup_{x,a} \left| r(x,a) + \gamma \int_{\mathcal{X}} P(dy|x,a) \max_{a'} \left(T^* Q_{k-1}(y,a') - \epsilon_{k-1}(y,a')\right) \right. \\
&\quad \left. - \left(T^* Q_{k-1}(x,a) - \epsilon_{k-1}(x,a)\right) \right| \\
&\leq (1+\gamma) \|\epsilon_{k-1}\|_\infty + \sup_{x,a} \left| \gamma \int_{\mathcal{X}} P(dy|x,a) \left(\max_{a'} \left(T^* Q_{k-1}(y,a')\right) \right. \right.
\end{aligned}
$$

$$
\begin{aligned}
& \qquad - \max_{a''} \left(Q_{k-1}(y, a'')\right)\Big)\Bigg| \\
\leq\; & (1+\gamma)\left\|\epsilon_{k-1}\right\|_\infty \\
& + \gamma \sup_{x,a} \left| \int_{\mathcal{X}} P(dy|x,a) \left(\max_{a'} \left|T^* Q_{k-1}(y,a') - Q_{k-1}(y,a')\right| \right) \right| \\
\leq\; & (1+\gamma)\left\|\epsilon_{k-1}\right\|_\infty . + \gamma \left\|\varrho_{k-1}\right\|_\infty .
\end{aligned}
$$

By unfolding the recursion and noting that $\|\varrho_0\|_\infty \leq R_{\max}$ when Q_0 is initialized to zero for each state-action pair, the lemma is proved. □

Leveraging on the previous Lemma, we can provide a bound to $\|\tilde{\varrho}_k\|_\infty$.

Lemma 3. *Let $(Q_i)_{i=0}^{k-1}$ be a sequence of state-action value function, $(\epsilon_i)_{i=0}^{k-1}$ be the corresponding sequence as defined in* (3.52)*, then*

$$
\|\tilde{\varrho}_k\|_\infty \leq (1+\gamma) \sum_{i=0}^{k-1} \gamma^{k-i-1} \|\epsilon_i\|_\infty + \gamma^k R_{\max} + 2R_{\max}.
$$

Proof.

$$
\begin{aligned}
\|\tilde{\varrho}_k\|_\infty = \left\|\hat{T} Q_k - Q_k\right\|_\infty \;=\; & \sup_{x,a,x',r} \left| r + \gamma \max_{a'} Q_k(x',a') - Q_k(x,a) \right| \\
=\; & \sup_{x,a,x',r} \Big| r + \gamma \max_{a'} \left(T^* Q_{k-1}(x',a') - \epsilon_{k-1}(x',a')\right) \\
& - \left(T^* Q_{k-1}(x,a) - \epsilon_{k-1}(x,a)\right) \Big| \\
\leq\; & (1+\gamma)\left\|\epsilon_{k-1}\right\|_\infty + 2R_{\max} \\
& + \gamma \sup_{x,a.x'} \Bigg| \Bigg(\max_{a'} \left(T^* Q_{k-1}(x',a')\right) \\
& - \int_{\mathcal{X}} P(dy|x,a) \max_{a''} \left(Q_{k-1}(y,a'')\right) \Bigg) \Bigg| \\
\leq\; & (1+\gamma)\left\|\epsilon_{k-1}\right\|_\infty + 2R_{\max} \\
& + \gamma \sup_{z} \max_{a'} \left|T^* Q_{k-1}(z,a') - Q_{k-1}(z,a')\right| \\
=\; & (1+\gamma)\left\|\epsilon_{k-1}\right\|_\infty + 2R_{\max} + \gamma \left\|\varrho_{k-1}\right\|_\infty .
\end{aligned}
$$

Finally, using Lemma 2 we get the statement. □

From the stated results, it can be noticed that when the errors at previous iterations is small enough, B-FQI can achieve an upper bound to the estimation error at iteration k similar to other FVI methods, but needing fewer samples since the range of the target variable is smaller.

Chapter 4

Empirical Results

> *"They believed that prediction was just a function of keeping track of things. If you knew enough, you could predict anything. That's been cherished scientific belief since Newton.*
> *'And?'*
> *Chaos theory throws it right out the window."*
>
> Ian Malcom, Jurassic Park[1]

In this chapter we provide the experimental settings and the empirical results. In Section 4.1 we introduce the technology details of the framework developed in order to run the experiments, explaining how we implemented such framework and how we optimized computations (further delails can be found in Appendix A). In Sections 4.2, 4.3, 4.4, we define the RL tasks that are used to compare B-FQI and FQI. In this chapter we provide two different analysis: a view on how B-FQI and FQI behave with different model complexity is provided in Section 4.5 while a comparison on how the two algorithms behave with different dataset sizes is provided in Section 4.6. In Section 4.7 we compare B-FQI and FQI in the challenging environment of the bicycle.

[1] The reader may notice that this citation is an unhappy choice for introducing the empirical results (even thought that in this thesis prediction plays a fundamental role). The author would like to reassure the reader that the experiments gave quite satisfying results, and of course none of the environment tested was chaotic. The citation it is both inspired by the fact that the author believes that programming is a chaotic process - but we leave this analysis for further studies, and by the love for the chaos theory, which makes human life more challenging and more beautiful as one could expect.

4.1 Technology Details

In order to develop and test B-FQI we designed a powerful framework called `iFQI`, which provides an implementation of both FQI and B-FQI, allows the user to run a number of different experiments by writing a proper configuration file, and collects all the observed variables on files. `iFQI` is wrote in python (2.7), uses *NumPy* (1.12.0) [78] for matrix calculus, *Keras* (1.2.1) [16] for Neural Networks with *Theano* (0.8.2) [1] back-end for symbolic computations, *SciKit-learn* (0.17.1) [60] for extremely randomized trees (Extra-Trees) [28] and for the dataset reshaping and *OpenAI Gym* (0.7.1) [12] for running RL tasks. `iFQI` is thought to be a flexible framework, allowing the users to define new environments, use different kinds of regressors and write other RL algorithms.

Structure of iFQI `iFQI` is organized in different modules: `envs`, `model`, `experiment`, `evaluation` and `ifqi`. Module `envs` is a collection of environments: some of these environments are wrote by us, while others are just a wrap of OpenAI Gym environments. Module `model` is a collection of the available regressors. In this document we refer to B-FQI as a variation of FQI, but in the code we found more convenient to see FQI as a unique algorithm that receives a normal regressor or a "boosted" regressor (which in the code's nomenclature is called "ensemble"). In this way we could let the regressor perform data reshaping, optimization of the computations, and to sum up the results provided by the regressors when a prediction is required. We must also introduce here the concept of *action-regressor* which corresponds in our code to a collection of regressors, where each one of them approximates the value function of a particular action. `experiment` class is responsible of reading the configuration file and preparing the experiment accordingly. Module `evaluation` collects the datasets and evaluates the policy found at the i-th iteration. Module `ifqi` simply implements the FQI algorithm, abstracting from the kind of regressor used, thus in practice could also be used as B-FQI.

Optimizing B-FQI In Chapter 3 we saw that B-FQI at the iteration k express $Q_{k+1} = Q_k + \hat{S}\hat{\varrho}_k$, where $\hat{\varrho}_k = \hat{T}^* Q_k - Q_k = \sum_{i=0}^{k-1} \hat{\varrho}_i$ and $S\hat{\varrho}_k = \arg\min_{f \in \mathcal{F}} \|\hat{\varrho}_k - f\|_{D^k_{N_k}}$. This implies that in order to fit $S\hat{\varrho}_k$, we requires to sum up k approximation terms. This could become a problem since it implies that B-FQI requires quadratic time considering the overall fitting phase while FQI requires only linear time with respect to the number of iterations. This issue could be circumvented by storing the output of $Q_{k-1}(X_i^{(k-1)})$

and then reusing it to compute $Q_k(X_i^{(k)}) = Q_{k-1}(X_i^{(k)}) + \hat{\varrho}_{k-1}$ since $D_{N_k}^{(k)} = D_{N_{k-1}}^{(k-1)}$ and $X_i^{(k)} = X_i^{(k-1)}$ because the dataset never changes through the iterations. The same reasoning can not be applied in the prediction phase, in fact X_i is assumed to be unseen. This is a problem, because in the prediction phase we must compute and sum the output of a number of regressors, and it could be that the sum of the simpler model of B-FQI requires more time than computing only one output of a more complex model of FQI. Since the complexity of the models used by B-FQI is kept constant iteration by iteration, there will be an iteration k^* such that the sum of k^*+1 simpler models will requires more time than FQI. The problem could be solved by parallelizing the prediction of k models, so that also the prediction phase of B-FQI will potentially cost less with respect to the prediction of FQI. In practice, the parallelization introduces an overhead, and in our case the computations were faster without parallelization, because our models were too simple and the number of models was not high enough to bypass the overhead. We still want to mention this possibility, because in more challenging environments parallelization could play an important role. A pseudo-code of the optimized implementation of B-FQI could be found in Appendix A.

4.2 Swing-Up Pendulum

Swing-Up Pendulum [19] is a classical RL problem. It consists in a pendulum fixed in one of its extremity. The goal is to swing up the pendulum in a vertical position and prevent it to fall. The system is controlled through the continuous torque applied to the pendulum. The pendulum is more formally described by the following dynamics[2]:

$$\begin{aligned} \dot{\theta}_{t+1} &= \dot{\theta}_t - \frac{3g}{2l}\sin\theta_t + \pi + \frac{3}{ml^2 u_t}\mathrm{d}t && (4.1) \\ \theta_{t+1} &= \theta_t + \Delta t \dot{\theta}_t, && (4.2) \end{aligned}$$

where $\Delta t = 0.05$, $g = 10$, $m = 1$, $l = 1$ and $\mu = 0.01$. The scalar value $u \in [-2, 2]$ represents the torque. There is not an absorbing state, and the episode has a fixed duration of 200 steps. The discount factor is $\gamma = 1$. The state is described by $(\cos(\theta), \sin(\theta), \dot{\theta})$, and the initial state is drawn uniformly from $\{(\cos(\theta), \sin(\theta), \dot{\theta}) | \theta \in [-\pi, \pi], \dot{\theta} \in [-1, 1]\}$. The

[2] We use here the dynamics used by the OpendAI Gym's implementation `Pendulum-v0`, which are slightly different by the one used by Doya et al. [19].

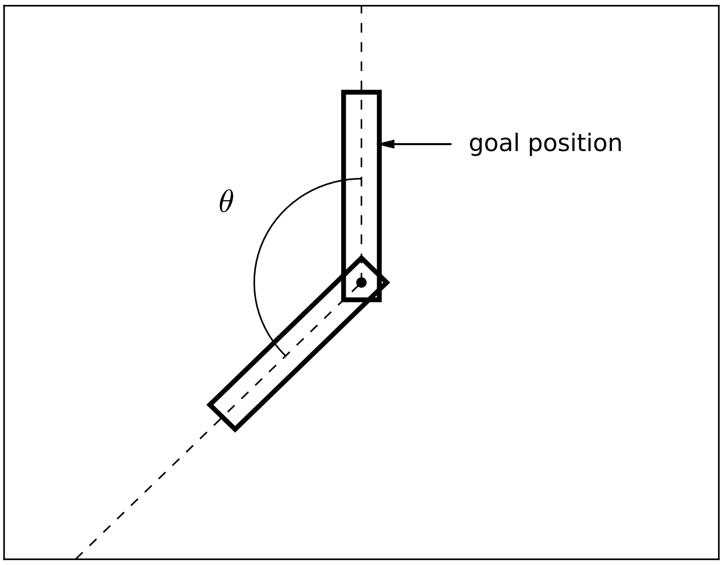

Figure 4.1: Here an example of swing pendulum both with displacement θ from the vertical axis, and in its goal position (with $\theta = 0$).

training set is composed by random episodes, where the action is sampled uniformly from $[-2, 2]$ at each step. Notice that we just use two discrete actions (that correspond to the borders of the interval) for the computation of $\hat{T}$ and the greedy policy. Since the initial state is randomly selected, in the evaluation phase we average the performance over 5 episodes. We average every evaluation among 5 different episodes.

4.3 Cart-Pole

Cart-pole [4][3] problem consists in balancing in upright position a pole attached to a cart moving on a frictionless track. The cart can be controlled by means of a discrete force of $10N$ that is possible to apply both to the left or to the right of the cart. The goal is to keep the pole in vertical position between $|\theta| \leq 12\pi/180$ where $|\theta|$ is the angle between the pole and the upright position. Moreover the cart moves on the track between $|z| \leq 2.4$ where z indicates the horizontal position of the cart. When one of the two conditions fails, the episode terminates. The reward function is expressed

[3] Actually the dynamic described by Barto et al. [4] in his article both the track and the joint between the cart and the pole have a small friction (figure 2.1). In the implementation `Cartpole-v0` of OpenAi Gym, which makes reference to those article, they use simpler dynamics that do not make usage of friction. We report here the dynamics used by OpenAI gym, since we also used those dynamics.

in function of the angle θ, and more precisely $r = \cos(15\theta)^4$. The dynamics are described by:

$$\begin{aligned}
\delta_t &= \frac{F + ml\theta^2 \sin\theta}{m_c + m}, && (4.3)\\
\ddot{\theta}_t &= \frac{g \sin\theta - \cos\theta\delta}{l\left[\frac{4}{3} - \frac{m\cos^2\theta}{m_c+m}\right]}, && (4.4)\\
\ddot{z}_t &= \frac{F + ml[\dot{\theta}_t^{\,2} \sin\theta_t - \ddot{\theta}_t \cos\theta_t]}{m_c + m}, && (4.5)
\end{aligned}$$

where $g = 9.8m/s^2$ corresponds to the gravity acceleration, $m_c = 1kg$ to the mass of the cart, $m = 0.1kg$ to the mass of the pole, $l = 0.5m$ to half of the pole length, and F corresponds to the force applied to the cart. The dynamics are updated every $\Delta t = 0.02$ seconds, thus $\theta_{t+1} = \theta_t + \Delta t\dot{\theta}_t$, $\dot{\theta}_{t+1} = \dot{\theta}_t + \Delta t\ddot{\theta}_t$, $z_{t+1} = z_t + \Delta t\dot{z}_t$ and $\dot{z}_{t+1} = \dot{z}_t + \Delta t\ddot{z}_t$. The state is encoded by $(z, \dot{z}, \theta, \dot{\theta})$. During the evaluation the initial state is sampled uniformly from $\{(z, \dot{z}, \theta, \dot{\theta}) | (z, \dot{z}, \theta, \dot{\theta}) \in [-0.05, 0.05]^4\}$ while during the collection of episodes for the dataset the initial state is sampled uniformly from $\{(z, \dot{z}, \theta, \dot{\theta}) | z \in [-2.4, 2.4], \dot{z} \in [-3.5, 3.5], \theta \in [-0.05, 0.05], \dot{\theta} \in [-3.0, 3.0]\}$. The episodes for the dataset are collected using a random policy.

4.4 Bicycle Balancing

Bicycle Balancing [64] [20][5] is another classical problem, where the goal is to drive a bicycle without falling. The state is described by $x = (\theta, \dot{\theta}, \omega, \dot{\omega}, \psi)$ where θ describes the front wheel angle from the straightforward direction and $\dot{\theta}$ is its angular velocity, ω is the angle that represents the tilt of the bike's frame from the vertical axis and $\dot{\omega}$ is its angular velocity, ψ is the angle between the frame of the bicycle and the x-axis (the bicycle rides on the $x - y$ plane). There are 9 possible actions $\{(d, T) | d \in \{-0.02, 0, 0.02\}, T \in \{-2, 0, 2\}\}$ where d describes the displacement of the rider respect the bicycle frame and T the torque applied to the handlebars.

The episodes terminates when $|\omega| \geq 12\pi/180$. We redefined the reward as $r = -|\omega|\frac{180}{12\pi}$ so that the reward is always between 0 and -1 (since $|\omega| \leq \frac{12\pi}{180}$). When the bicycle falls, the episodes terminates with a reward $r = -1$. The dynamic is described by:

[4] We use $r = \cos(15\theta)$ instead of $r = \cos(\theta)$ because $|\theta| \leq 12\pi/180$ and thus $0.978 \leq cos(\theta) \leq 1$, while since $|15\theta| \leq \pi$, $cos(15\theta)$ could range from -1 and 1.

[5] The implementation of the dynamics are slightly different between Ernst et al. [20] and Randløv et al. [64]. We used the ones described by Ernst et al.

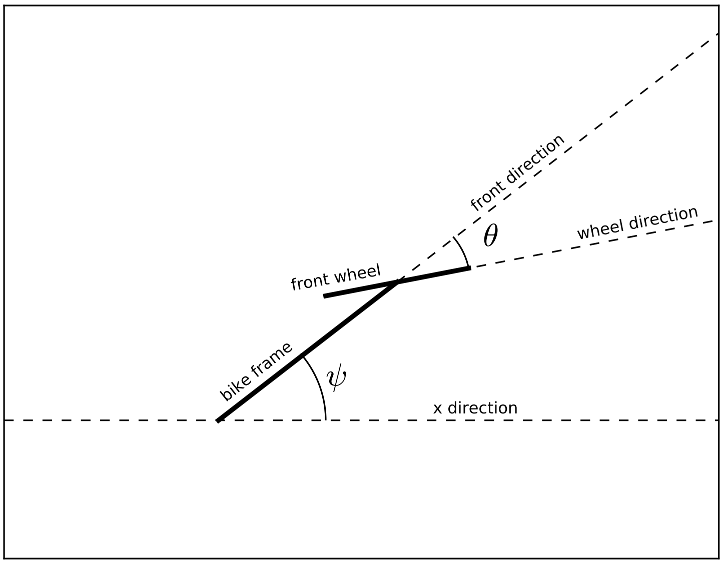

Figure 4.2: Bicycle from the top: Here we can observe the bicycle frame which forms the angle ψ with the x-direction, while θ represents the direction of the front wheel with respect to the direction of the frame.

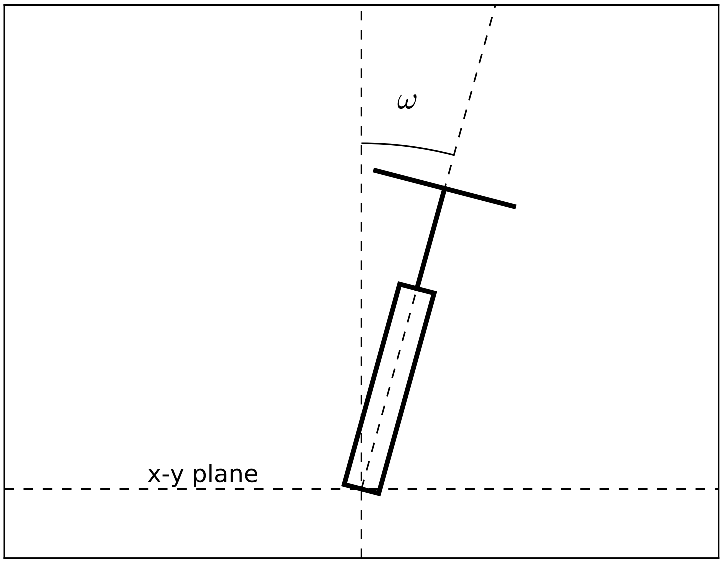

Figure 4.3: Bicycle from front view: Here we can observe the bicycle forming an angle of ω with respect to the vertical axis.

$$
\begin{aligned}
\omega_{t+1} &= \omega_t \Delta t \dot{\omega}_t, && (4.6)\\
\omega_{t+1} &= \dot{\omega}_t + \Delta t\Big(\frac{1}{I_{bc}}(Mhg\sin(\varphi_t) - \cos(\varphi_t) \\
&\quad (I_{dc}\dot{\sigma}\dot{\theta}_t + \mathrm{sign}(\theta_t)v^2(\mathrm{invr}_{f_t} + \mathrm{invr}_{b_t}) + Mh\mathrm{invr}_{CM_t})))\Big), && (4.7)\\
\theta_{t+1} &= \begin{cases} \theta_t + \Delta t\dot{\theta}_t & \text{if} \quad |\theta_t + \Delta t\dot{\theta}_t| \leq \frac{80}{180}\pi, \\ \mathrm{sign}(\theta_t + \Delta t\dot{\theta}_t)\frac{80}{180}\pi & \text{if} \quad |\theta_t + \Delta t\dot{\theta}_t| > \frac{80}{180}\pi, \end{cases} && (4.8)\\
\dot{\theta}_{t+1} &= \begin{cases} \dot{\theta}_t + \Delta t\frac{T - I_{dv}\dot{\sigma}\dot{\omega}_t}{I_{dl}} & \text{if} \quad |\theta_t + \Delta t\dot{\theta}_t| \leq \frac{80}{180}\pi, \\ 0 & \text{if} \quad |\theta_t + \Delta t\dot{\theta}_t| > \frac{80}{180}\pi, \end{cases} && (4.9)\\
x_{b_{t+1}} &= x_{b_{t+1}}\Delta tv\cos(\psi_t), && (4.10)\\
y_{b_{t+1}} &= x_{b_{t+1}}\Delta tv\sin(\psi_t), && (4.11)\\
\psi_{t+1} &= \psi_{t+1} + \Delta tv\mathrm{sign}(\theta_t)v\mathrm{invr}_{b_t}, && (4.12)
\end{aligned}
$$

with

$$
\begin{aligned}
\varphi_t &= \omega_t + \frac{\arctan(d_t + w_t)}{h}, && (4.13)\\
\mathrm{invr_{f_t}} &= \frac{|\sin(\theta_t)|}{l}, && (4.14)\\
\mathrm{invr_{b_t}} &= \frac{|\tan(\theta_t)|}{l}, && (4.15)\\
\mathrm{invr_{CM_t}} &= \begin{cases} \frac{1}{\sqrt{(l-c)^2 + \frac{1}{\mathrm{invr_{b_t}}^2}}} & \text{if} \quad \theta_t \neq 0, \\ 0 & \text{otherwise}, \end{cases} && (4.16)
\end{aligned}
$$

where w_t is a uniform noise drawn from the interval $[-0.02, 0.02]$, $\Delta t = 0.01$, $v = 10/3.6$,$g = 9.82$, $d_{CM} = 0.3$, $c = 0.66$, $h = 0.94$, $M_c = 15$, $M_d = 1.7$, $M_p = 60.0$, $M = M_c + M_p$, 0.34, $\dot{\sigma} = v/r$, $I_{bc} = (13/3M_c h^2 + M_p(h+d_{CM})^2))$, $I_{dc} = M_d^2 r^2$, $I_{dv} = 9/4M_d^2 r^2$ and $l = 1.11$. The initial state during the collection of the random episodes for the training set is sampled uniformed from $\{(\theta, \dot{\theta}, \omega, \dot{\omega}, \psi)|\theta = \dot{\theta} = \omega = \dot{\omega} = 0, \psi \in [-\pi, \pi]\}$ while for evaluation we evaluated with one initial state $x \in \{(\theta, \dot{\theta}, \omega, \dot{\omega}, \psi)|\theta = \dot{\theta} = \omega = \dot{\omega} = \psi = 0\}$. In order to make the computation faster, we evaluated the bicycle with some difference with respect to Ernst et al. [20]

1. Ernst et al. consider the episode finished either when the bicycle falls or when it rides for 50000 steps. We instead let the episode ends after "only" 5000 steps (which correspond to $50s$),

2. Ernst et al. evaluate the policy from 9 different initial states which differs only for a different ψ. We instead evaluate once with $\psi = 0$:

this should not change the expected value of J^π because ψ does not effect the future rewards.

4.5 Complexity Analysis

The goal of the complexity analysis is to study how the algorithms FQI and B-FQI perform with different model complexity. In order to do this we use models with different complexity that share the same hyper-parameters for both FQI and B-FQI and we measure the performance by varying the number of neurons in the case of Neural Network [31] and the maximum depth of the Extra-Trees [28].

Swing-Up Pendulum In the case of Swing-Up pendulum we collected 10 datasets of 2000 episodes to average the results. We perform one test with a comparison between Neural Networks and one with Extra Trees. The hyper parameters of the neural networks were found by means of a genetic algorithm [50][6]. The neural networks has only one layer with *sigmoid* as activation function. The number of neurons is not fixed, as we use it as a measure of the complexity of the functional space. The weights are initialized with glorot uniform, and trained with MRSProp [31]. We used early stopping [30][7] to decide when to stop the algorithm with $patience = 5$ and $\delta_{min} = 0.06$, with a batch size of 2000 and a validation set of 10%. The extra-trees ensemble is composed of 30 regressors with a minimum number of samples per split of 4 and a minimum number of samples per leaf of 2. The complexity analysis shows that B-FQI achieves an optimal performance with less complex model with respect to FQI, in fact, as we can see in Figures 4.4, 4.5, B-FQI requires more or less 5 neurons for achieve an high performance in contrast to FQI which requires 11 neurons to behave similarly. The same happends with Extra-Trees: B-FQI requires a max depth of 9 while FQI of 13.

Cart-Pole In the case of Cart-Pole we collected 10 datasets of 2000 episodes to average the results. We performed one test with a comparison between Neural Networks and one with Extra Trees, both with the usage of the action regressor (already introduced in Section 4.1). The hyper-parameters of the Neural Networks were found by means of a genetic algorithm. The Neural Networks has only one layer with *sigmoid* activation. The number of

[6] The implementation details are provided in Appendix.

[7] We provide an explanation of early stopping in Appendix.

neurons is not fixed, as we use it as a measure of the complexity of the functional space. The weights are initialized with Glorot Uniform, and trained with RMSProp. We used Early Stopping to decide when to stop the algorithm with $patience = 600$ and $\delta_{min} = 0.58$, with a batch size of 2000 and a validation set of 10%. The extra-trees ensemble is composed of 30 regressors with a minimum number of samples per split of 4 and a minimum number of samples per leaf of 2. The complexity analysis (Figures 4.6, 4.7) shows clearly that in the case of extra-trees B-FQI achieves the optimal policy with a simper model (we could say with max depth 5), while FQI to reach the same performance needs a model with max depth 13. The analysis with the neural networks shows instead that both the algorithms perform similarly with respect to complexity.

Results The complexity analysis shows in the case of swing-up pendulum that B-FQI achieves an optimal performance with less complex model with respect to FQI, in fact, as we can see in Figures 4.4, 4.5, B-FQI requires more or less 5 neurons for achieve an high performance in contrast to FQI which requires 11 neurons to behave similarly. The same happends with Extra-Trees: B-FQI requires a max depth of 9 while FQI of 13. The experiment with cart-pole (Figures 4.6, 4.7) shows clearly that in the case of extra-trees B-FQI achieves the optimal policy with a simper model (we could say with max depth 5), while FQI in order to reach the same performance needs a model with max depth 13. The analysis with the neural network is less clear, because the results are very similar. This former result could be explained by considering that both the algorithms seem to perform very well with few neurons (and this could be explained because we are using the action regressor, which probably helps).

4.6 Samples Analysis

The goal of samples analysis is to compare the difference of performance between FQI and B-FQI as a function of the dataset dimension. In order to make a fair comparison, we select from the previous analysis, two different models for FQI and B-FQI that perform similarly with a dataset of 2000 episodes, and then we measure the performance at the last iteration with datasets of different number of episodes. Each value plotted is averaged by 30 datasets of the same size.

Swing-Up Pendulum Both for FQI and B-FQI we selected the simplest (in terms of complexity) model with a performance higher than -1.5. The

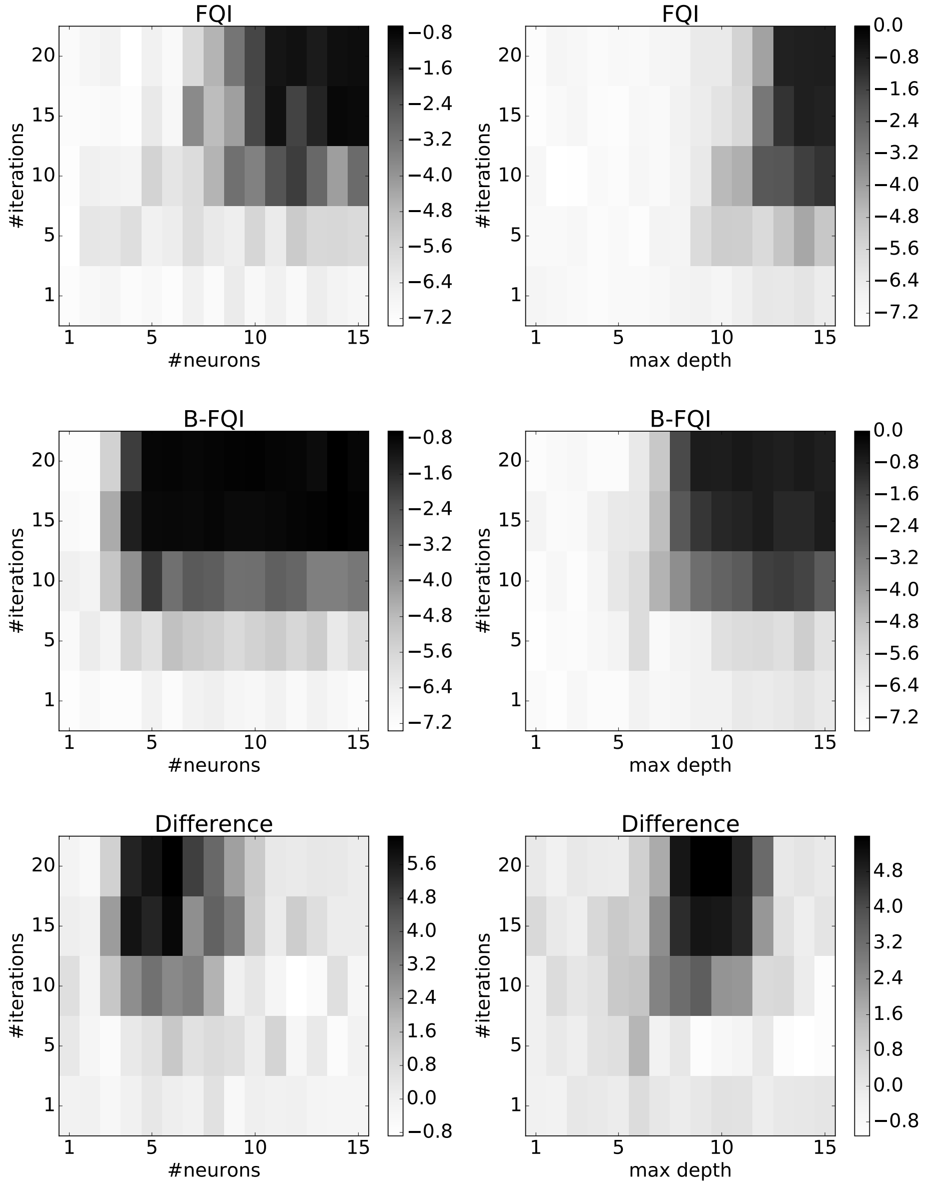

Figure 4.4: **Swing-up model complexity***: score of the greedy policy at last iteration* (K) *w.r.t. model complexity and iterations for neural networks (left column) and extra-trees (right column). The heatmap Difference show the score* $J^{\pi_K}_{\text{DIFF}} = J^{\pi_K}_{\text{B-FQI}} - J^{\pi_K}_{\text{FQI}}$.

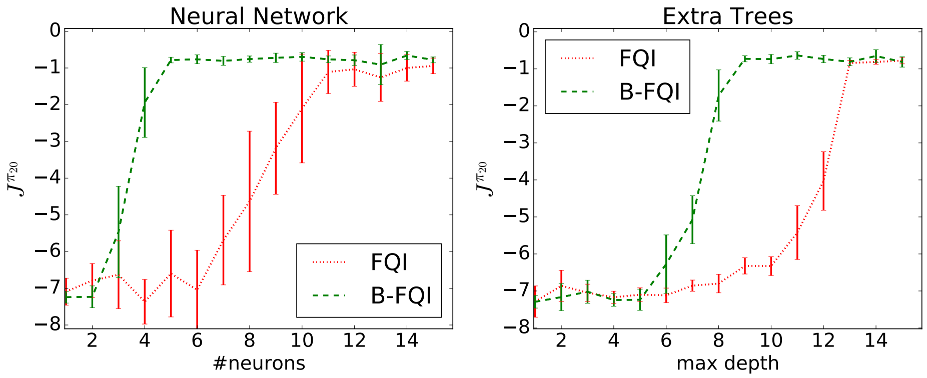

Figure 4.5: **Swing-up model complexity***: score of the greedy policy at iteration* 20 *(*$J^{\pi_{20}}$ *w.r.t. the model complexity (left: neural networks, right: extra-trees). Confidence intervals at* 95% *are shown.*

selected configurations are the following ones. The number of neurons in Neural Networks is 5 for B-FQI and 11 for FQI, while the max depth of the Extra Trees is 6 and 13 for B-FQI and FQI, respectively. All the other parameters are as described before.

Cart-Pole Differently from Swing-Up Pendulum, we used two different thresholds for extra-trees and neural networks, in order to pick up models that in the worst case disadvantage B-FQI. We selected the simplest model complexity that achieved 190.0 at the 10-th iteration in the case of neural networks, that results to be 2 neurons for B-FQI and 4 neurons for B-FQI, while we used 198.0 as threshold for extra-trees which turned out to correspodt 6 as max depth for B-FQI and 14 for FQI.

Results The samples analysis shows in this case of swing-up pendulum that B-FQI seems to use more efficently the number of samples: in fact even using a lower-complexity model, we can see from Figure 4.8 that B-FQI almost always beat FQI. In the case of cart-pole shown in Figure 4.9, the result is less clear since both FQI and B-FQI gives similar results. The difference between the results of the two experiments could be explained by the fact that it was impossible to choose two models for FQI and B-FQI that performed perfectly equally with a datasets of 2000 episodes, thus in some cases (like in cart-pole) we have choosen a model that penalizes too much B-FQI. Another reason could be that effectively there is not a difference between FQI and B-FQI with respect the efficency of data usage: while boosting methods are well known for efficency in terms of model complexity,

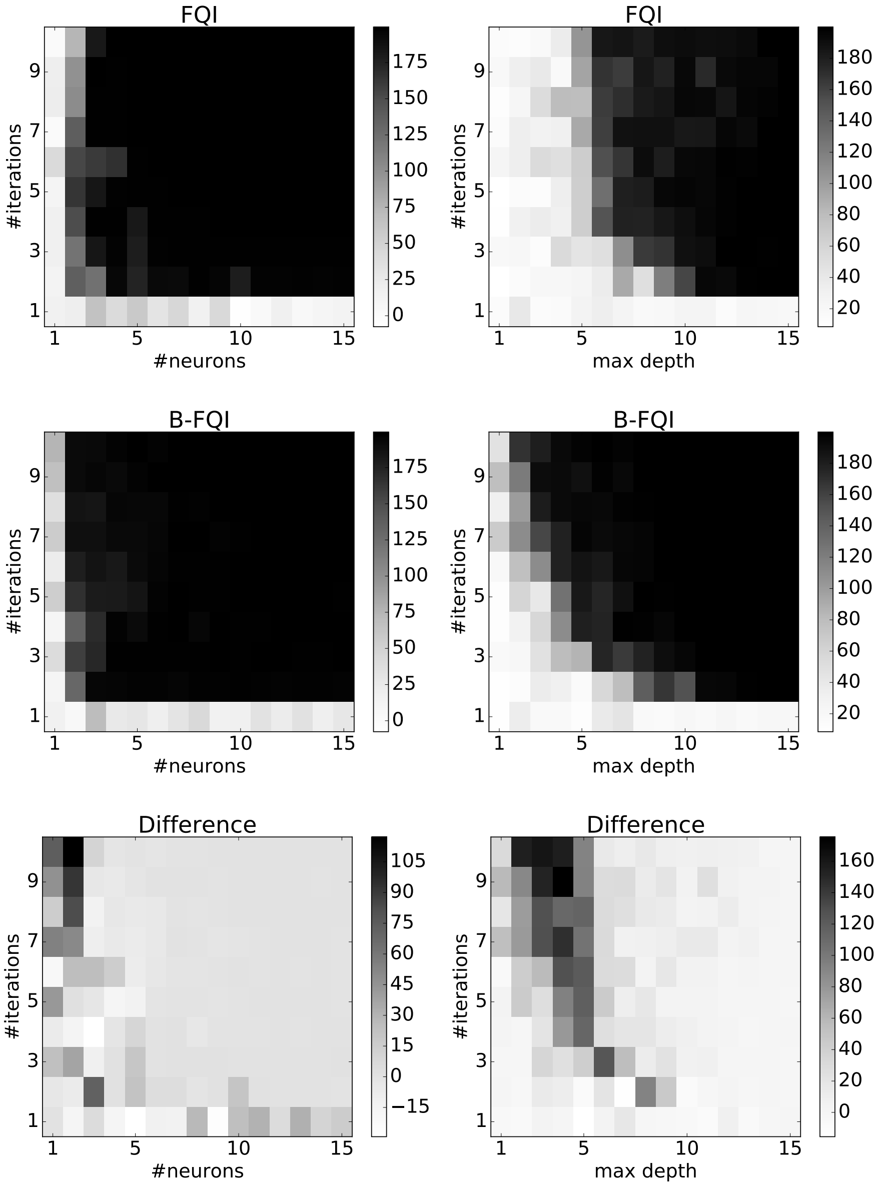

Figure 4.6: **Cart-Pole model complexity***: score J^{π_K} obtained by the algorithms applying the greedy policy π_K in the cart-pole balancing experiments with neural networks (left column) and extra trees ensembles (right column) measured w.r.t. the number of iterations K and model complexity. The heatmap called Difference show the score $J^{\pi_K}_{diff} = J^{\pi_K}_{BFQI} - J^{\pi_K}_{FQI}$.*

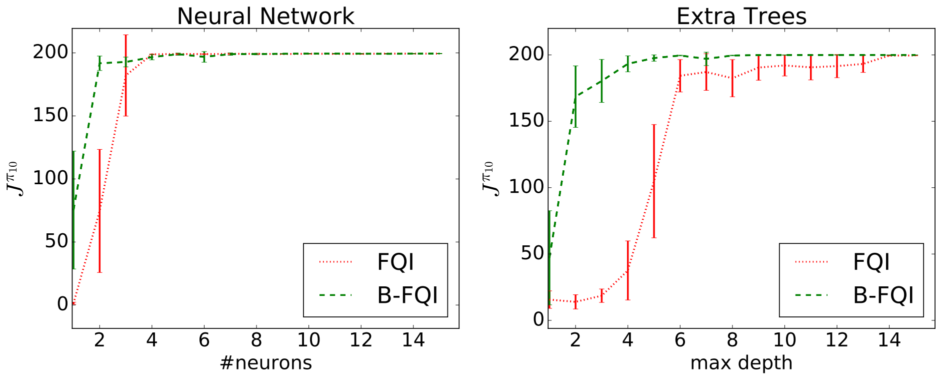

Figure 4.7: **Cart-Pole model complexity**: *score $J^{\pi_{10}}$ obtained by the algorithms applying the greedy policy π_{10} in the cart-pole balancing experiments with neural networks (left) and extra trees ensembles (right). Level 95% confidence intervals are shown.*

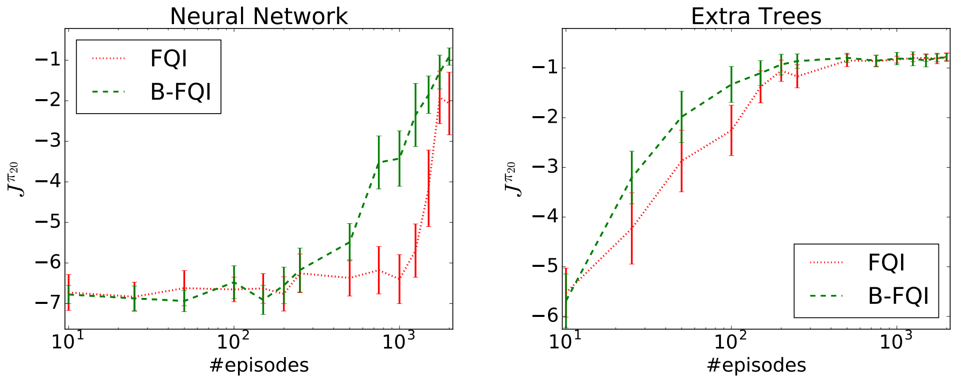

Figure 4.8: **Swing-up sample complexity**: *score of the greedy policy at iteration 20 ($J^{\pi_{20}}$ w.r.t. the dataset size (left: neural networks, right: extra-trees). Confidence intervals at 95% are shown.*

it is not clear wether boosting provides data efficiency.

4.7 The Bicycle experiment

We finally performed an experiment with the bicycle balancing problem to show the difference between B-FQI and FQI in a more challenging environment. The experiment is performed both for extra-trees and for neural networks. In the case of extra-trees we used the same configuration of Ernst et al. [20], except for the max depth of 17 which is not defined in their paper. For Neural Network, we used a model with 2 layers of 10 sigmoidal neurons. The genetic algorithms selected $\delta_{min} = 0.011$ and $patience = 75$ as parameters for the Early Stopping. In figure Figure 4.10 we can see that

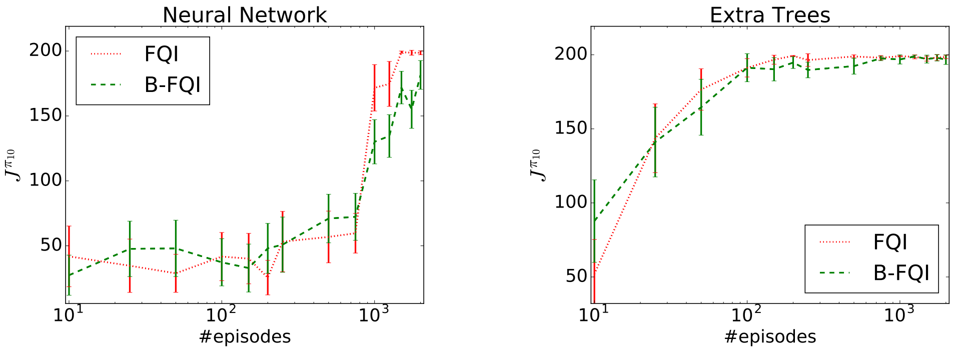

Figure 4.9: **Cart-pole sample complexity***: score $J^{\pi_{20}}$ obtained by the algorithms in the cart-pole balancing experiments applying the greedy policy π_{20} learned using datasets of increasing size with neural networks (left) and extra trees ensembles (right). Level 95% confidence intervals are shown.*

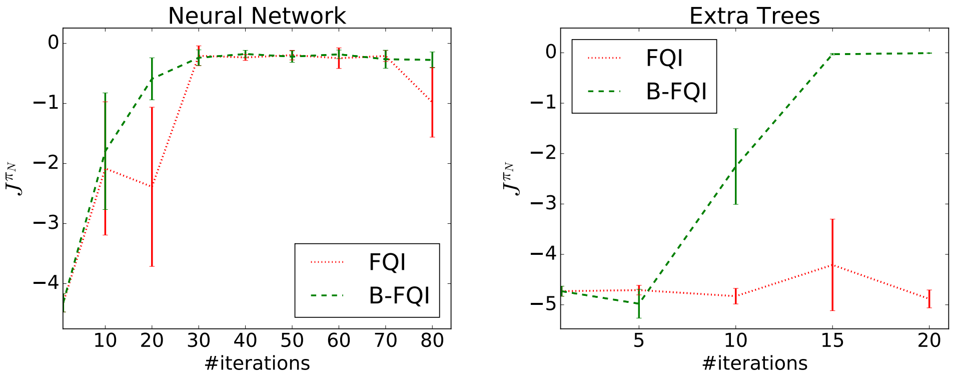

Figure 4.10: **Bicycle performance***: score of the greedy policy π_K as a function of the iterations with neural networks (left) and extra-trees (right). Confidence intervals at 95% are shown.*

B-FQI performs very well and approaches the optimal policy on the 15th iteration with Extra Trees, while with the same configuration FQI perform very poorly. The fact that FQI performs such bad is due to the fact that we have introduced max depth of 17 while Ernst et al. [20] does not define a max depth allowing the trees to grow as much is required. With Neural Networks both B-FQI and FQI perform well without a significant difference. In Figure 4.11 we can see the average number of steps for the same experiments.

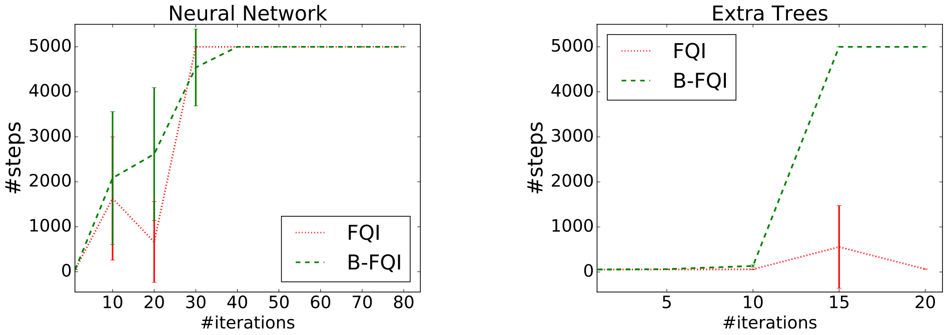

Figure 4.11: **Bicycle Steps***: Number of steps in which the bicycle moves without falling during an episode obtained by the algorithms in the bicycle balancing experiments applying the greedy policy π_K learned using neural networks (left) and extra trees ensembles (right). Level 95% confidence intervals are shown.*

Chapter 5

Conclusion and Future Works

"E infine
candida e lieve
si scioglie, la neve.
(Candid and mild,
eventually melts,
the snow.)"

Silvia Bradanini

The goal of our work was to provide an efficient way to solve RL tasks possibly finding an ad-hoc learner that exploits the properties that are shared among all those tasks. We focused on the already existing class of value iteration algorithms, and in particular of Fitted Q-Iteration, which is simple to analyze since avoids the exploration-exploitation trade-off and it consists in iterating the empirical optimality Bellman operator and an approximation phase. The main question was on how, knowing the properties of the bellman operator, we could derive a smart function approximator. Actually, before approaching to the current solution, we studied different solutions, like for example to reuse features of a neural network between on iteration and the following one: is in fact clear that one of the most interesting property of AVI methods is that the action-state value function found at one iteration could be used to explain the value function in the following one. We found out that a very effective way to solve our problem was to introduce an incremental learner that at each iteration was approximating the bellman residual. We know from RL theory that the Bellman residual should converge to zero, moreover we know from SL that there is a class of boosting algorithms that uses models composed by weak regressors where each regressor learns the residual: this class is the L_2-Boost class. We provided the theoretical propagation error bounds showing what conditions are required for B-FQI to converge and what error is provided with the case of non-greedy bellman operator. The theoretical finite-sample analysis shows that the error bound could be decomposed in three components that are related to the error committed in approximating the residual, the approximation error committed the previous iteration and a term that involves the functional complexity and the number of samples. This former result shows clearly the advantage of approximating the residual, because when the algorithm converges, the Bellman residual ϱ_k^* converges to zero, and the common sense suggests that the approximation complexity should be very small. The same analysis suggests that fewer samples are required for approximating the residual for the same reason. We also provided by means of experiments two different studies: the complexity analysis and the sample analysis. The empirical results shown that B-FQI always approaches to the optimal value function with a model that is simpler (or in the worst-case equivalent) with respect to the model used by FQI, showing that our supposition was correct, while the results from the sample analysis are not so clear yet. B-FQI results to be a very interesting algorithm, and it deserves surely further study. While will be necessary maybe to enrich both our theoretical analysis and our empirical results (testing B-FQI in with more challenging RL tasks and providing a better samples analysis), it will be necessary also to expand our studies in order to provide a more general and efficient algorithm. With "more general" we mean that actually B-FQI belongs to the class of FVI algorithms, but would be nice to provide the correspective algorithm also for online methods. We also said that we would like to provide a more efficient version of the algorithm: it is in fact evident that the weakness of B-FQI is that it uses regressors of same complexity through all the iterations. It is clear that the Bellman residual tends to be more complex in earlier iterations, while it becames simpler iteration by iteration. We should find a way to dynamically restrict the functional space both to provide a more robust algorithm and to save more computational resources.

Increasing time efficiency. There are a lot of euristic ways to dynamically change the complexity of the functional space in order to save time. Very unfortunately it is difficult to derive an analytic formula for which it is possible to find a way to increase the time efficiency and at the same time restrict the functional space. It really depends by the regressor that we are using: in the case of an Extra-Tree we might want to reduce the max-depth of the trees while in the case of neural networks the number of neurons. This requires that B-FQI must be able to detect without a theoretical support how to restrict the functional space. We propose here two different solutions: the first one is a version of B-FQI that uses at each iteration two learners, one slighlty more expressive that the other. We compare the accuracy of both the learners: if the accuracy is very similar, in the next iteration we keep the learner with lower complexity and we introduce also a new regressor with less expressivity, discarding the regressor with higher complexity. The second technique that we propose is to use boosting also for approximating the Bellman residual, using very weak regressor as base procedure in such the way that when the Bellman residual is close to zero the boosting procedure will fit it in very few iterations, using also a very limited functional space.

More theoretical approaches. Another way could be to provide regularization, like is it done in R-FQI, in such the way that the regularization coefficient adapts dinamically in order to restrict the functional space avoiding the overfitting. This method is very interesting, because it is applicable to all parametric regressors, providing a more sophisticated way to restrict the functional space, and maybe it would allow to find an optimal way to dynamically change the penalization coefficient and thus to dynamically restrict the functional space. Very unfortunately, while this seems a good way to provide a robust algorithm from the theoretical point of view, it will not provide time efficiency.

Bibliography

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[2] Leemon Baird et al. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37, 1995.

[3] Andrew R Barron, Albert Cohen, Wolfgang Dahmen, and Ronald A DeVore. Approximation and learning by greedy algorithms. *The annals of statistics*, pages 64–94, 2008.

[4] 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.

[5] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[6] Dimitri P Bertsekas. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific*, 2011.

[7] DP Bertsekas and J Tsitsiklis. N.(1996) neuro-dynamic programming. *Athena Scientific.*

[8] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[9] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[10] Leo Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[11] Leo Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

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

[13] Peter Bühlmann and Torsten Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, pages 477–505, 2007.

[14] Peter Bühlmann and Torsten Hothorn. Boosting Algorithms: Regularization, Prediction and Model Fitting. *Statistical Science*, 22(4):477–505, apr 2008.

[15] Peter Bühlmann and Bin Yu. Boosting with the l 2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

[16] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[17] Robert H Crites and Andrew G Barto. Elevator control using multiple reinforcement learning agents. *Kulwer Academeic Pulishers, Boston*, 1997.

[18] Geoff Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive approximation*, 13(1):57–98, 1997.

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

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

[21] Amir-Massoud Farahmand. *Regularization in Reinforcement Learning*. PhD thesis, Edmonton, Alta., Canada, 2011. AAINR89437.

[22] Amir Massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *2009 American Control Conference*, pages 725–730, June 2009.

[23] Amir Massoud Farahmand, Rémi Munos, and Csaba Szepesvári. Error propagation for approximate policy and value iteration. In *NIPS*, pages 568–576. Curran Associates, Inc., 2010.

[24] Amir Massoud Farahmand and Doina Precup. Value pursuit iteration. In *NIPS*, pages 1349–1357, 2012.

[25] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[26] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[27] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

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

[29] Mohammad Ghavamzadeh, Alessandro Lazaric, Rémi Munos, and Matt Hoffman. Finite-sample analysis of lasso-td. In *International Conference on Machine Learning*, 2011.

[30] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning Series. MIT Press, 2016.

[32] Geoffrey J Gordon. Approximate solutions to markov decision processes. *Robotics Institute*, page 228, 1999.

[33] Abhijit Gosavi. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, 55(1):5–29, 2004.

[34] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[35] László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A Distribution-Free Theory of Nonparametric Regression.* Springer series in statistics. Springer, 2002.

[36] Trevor J. Hastie and Robert John Tibshirani. *Generalized additive models.* Monographs on statistics and applied probability. Chapman & Hall, London, 1990.

[37] Ronald A Howard. Dynamic programming and markov processes.. 1960.

[38] Vasile I Istrătescu. *Fixed point theory: an introduction*, volume 7. Springer, 1981.

[39] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 6. Springer, 2013.

[40] Lars Juhl Jensen and Alex Bateman. The rise and fall of supervised machine learning techniques, 2011.

[41] Jeffrey Johns, Christopher Painter-Wakefield, and Ronald Parr. Linear complementarity for regularized policy evaluation and improvement. In *Advances in neural information processing systems*, pages 1009–1017, 2010.

[42] Michael Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript*, 45:105, 1988.

[43] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[44] 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.

[45] J Zico Kolter and Andrew Y Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528. ACM, 2009.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[47] Michael Krödel and Klaus-Dieter Kuhnert. Reinforcement learning to drive a car by pattern matching. In *Joint Pattern Recognition Symposium*, pages 322–329. Springer, 2002.

[48] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[49] Sridhar Mahadevan, Nicholas Marchalleck, Tapas K Das, and Abhijit Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 202–210. MORGAN KAUFMANN PUBLISHERS, INC., 1997.

[50] Melanie Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[52] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning.* MIT press, 2012.

[53] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[54] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *NIPS*, pages 1046–1054, 2016.

[55] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

[56] Gerhard Neumann and Jan R Peters. Fitted q-iteration by advantage weighted regression. In *Advances in neural information processing systems*, pages 1177–1184, 2009.

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

[58] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2-3):161–178, 2002.

[59] Yagyensh Chandra Pati, Ramin Rezaiifar, and PS Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

[60] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[61] Matteo Pirotta. *Reinforcement learning: from theory to algorithms.* PhD thesis, Italy, 2016.

[62] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

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

[64] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, volume 98, pages 463–471. Citeseer, 1998.

[65] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *Journal of Machine Learning Research*, 15(1):335–366, 2014.

[66] Martin Riedmiller. *Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method*, pages 317–328. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[67] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[68] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.

[69] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[70] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[71] Satinder Singh and Dimitri Bertsekas. Reinforcement learning for dynamic channel allocation in cellular tlephone. *Proc. NIPS96*, 1996.

[72] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media, 2005.

[73] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[74] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[75] John N Tsitsiklis, Benjamin Van Roy, et al. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[76] Aad van der Vaart and Jon A. Wellner. *Preservation Theorems for Glivenko-Cantelli and Uniform Glivenko-Cantelli Classes*, pages 115–133. Birkhäuser Boston, Boston, MA, 2000.

[77] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[78] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[79] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[80] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[81] Tong Zhang, Bin Yu, et al. Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 33(4):1538–1579, 2005.

[82] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[83] Ding-Xuan Zhou. Capacity of reproducing kernel spaces in learning theory. *IEEE Transactions on Information Theory*, 49(7):1743–1752, 2003.

Appendix A

About the Implementation

We describe here more in detail the practical implementation that we used. We will use pseudo-code in order to represent in a more compact and abstract way the main ideas, allowing us to focus only on such passages that we think to be more interesting. However, the implementation is available at `https://github.com/teopir/ifqi/tree/tosatto`. We present three classes that will be useful for the reader to understand the key point of our implementation: `FQI`, `ActionRegressor` and `EnsembleRegressor`.

The class `FQI` provides a general FQI algorithm abstracting from the kind of regressor used: in this way it is possible to both use it for run B-FQI or classic FQI.

In our code we use many kinds of regressor: all of them must implement the method `fit` which accepts two arguments X and Y and finds a function f such that it minimizes $f(X) - Y$ and a method `predict` which accepts only one argument X and it returns $f(X)$. In our implementation we added an optional argument to `predict` that allows to optimize computations.

We provide a sort of optimization relying on the fact that during a single iteration of FQI we pass always the same X to the regressor, thus is possible to save at every call of the `fit` procedure the partial summation S in such way to avoid to computing every time the previous bellman residuals. The `ActionRegressor` class uses different regressor for each action: this kind of regressor is particularly suited when the action space is discrete. The main idea behind this class is again to relieve `FQI` from this work.

Algorithm 8: FQI

```
Data: $x$ is the column vector of current states, $a$ is the column
      vector of actions, $r$ is the column vector of reward signal, $y$ is
      the column vector of next states, $t$ is a column vector
      indicating with 1 when $y$ is absorbing, 0 otherwise and
      regressor is our approximator
Procedure FQIIteration()
    $q \leftarrow r + \gamma(1 - t) \max_{a \in \mathcal{A}} \text{regressor.predict}([y, a], \text{optimize=true})$;
    $X[0] \leftarrow x$;
    $X[1] \leftarrow a$;
    regressor.fit$(X, q)$;
end
```

Algorithm 9: EnsembleRegressor

```
regressorList ← [];
Procedure Fit (X, Y)
    regressor ← newRegressor();
    regressor.fit$(X,\ S - Y)$;
    $S \leftarrow S + \text{regressor.predict}(X)$;
    regressorList.append(regressor) ;
end
Function Predict (X, optimize=false)
    if optimize then
        partialSum ← 0;
        for regressor ∈ regressorList do
            partialSum ← partialSum + regressor.predict(X);
        end
        return partialSum;
    else
        return S;
    end
end
```

Algorithm 10: ActionRegressor

```
regressor $\leftarrow$ $|\mathcal{A}|$ new regressors;
Procedure Fit (X, Y)
    for $a \in \mathcal{A}$ do
        index $\leftarrow$ where $X[:, 1] = a$;
        regressors[$a$].fit($X$[index,0], $Y$[index]);
    end
end
Function Predict (X, optimize=false)
    for $a \in \mathcal{A}$ do
        indexes $\leftarrow$ where $X[:, 1] = a$;
        regressors[$a$].fit($X$[indexes], $Y$[indexes]) predict[$a$] $\leftarrow$
          regressors[$a$].predict($X$[indexes,0], optimize);
    end
end
```

Appendix B

Early Stopping

Early Stopping [65] is a form of regularization used to avoid overfitting. It is used with those learners that approximate the target function iteratively, providing at each iteration a better approximation. Early stopping prevents such learners approaching to a function that fits too well the dataset by preventing in advance that the training error decreases too much. This technique is often associated with gradient methods [80], but it has also been used with boosting methods [81]. What is interesting about early stopping, is that it keeps in account the *validation error*: the error committed on predicting samples that were not used for training the learner. This is made because, even if the training error should in theory always decrease approaching to an asymptotic error, the validation error after a decreasing phase starts to increase again: this behavior is devoted to the bias-variance trade-off (Figure B.1). What we would like to do in order to prevent overfitting, is to stop the training algorithm when we believe to have reached the minimum validation error (thus a good generalization). Since early stopping is a meta-algorithm it does not define the so called *stopping criterion.* Lutz Prechelt shows in his article “Early Stopping - but when?” [62] that the choice of a right stopping criterion is crucial.

The implementation of early stopping that we are using in our code is provided by Keras [16] and it uses two main parameters to answer the question “when to stop?”. We say that the training is improving during the iteration k if the validation error at iteration k is lower than the validation error at the previous iteration decreased by δ_{min}. We will stop the training if it is not improving for p consecutive iterations where p is also called *patience.*

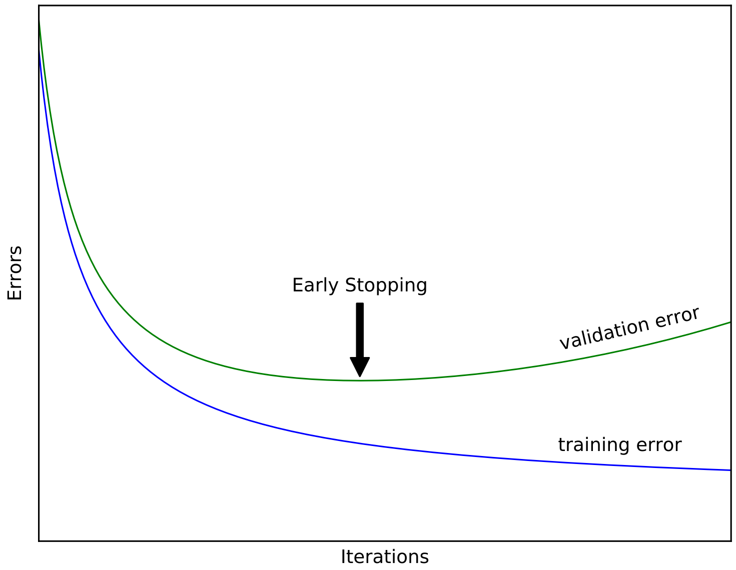

Figure B.1: Early Stopping: it is possible to see here that while the training error tends to decrease, the validation error at a certain point starts to increase. From this point we say that the learning process starts to overfit, this is the right point where to stop the training algorithm.

Algorithm 11: Early Stopping

```
Data: A patience $p$, a minimum improvement $\delta_{\min}$, and a regressor.
$noImprovement \leftarrow 0$;
$previousLoss \leftarrow +\infty$;
while $noImprovement < p$ do
    $validationLoss \leftarrow$ regressor.train();
    if $validationLoss < previousLoss - \delta_{\min}$ then
        $noImprovement \leftarrow 0$;
    else
        $noImprovement \leftarrow noImprovement + 1$;
    end
end
```

Appendix C

Genetic Algorithm

With Genetic Algorithms [79] we refer to a class of optimization algorithms that are inspired by the natural process of selection. Genetic algorithms proceed by means of an intelligent random search that generates a bunch of solutions. The search starts by generating some random solutions. These former are evaluated and some of them are selected by means of a stochastic process that gives higher pobability to the best ones. The new solutions are generated with a procedure inspired by the sexual reproduction from a combination of two previous selected solutions (the parents). A further randomization is provided by perturbing the new solutions with random *mutations*. Then the algorithm iterates this procedure in order to generate each time new solutions. Genetic algorithms are very powerful because they generally are very robust to noise, they can deal with large state space and they do not require particular objective function, unlike it happens for Gradient Descent algorithm, which requires differentiable loss function. Genetic algorithms proceed usually with a *population* of candidate solutions that are often called *individuals*. Each individual is described by its *chromosome* that determine its behaviour. The "goodness" of each individual is evaluated by means of a *fitness* function, the goal of the genetic algorithm is to find individuals with higher fitness. The fitness is usually the objective function in our optimization problem. The genetic algorithm proceeds by means of iterations that are called *epochs*. In each epoch, every individual belonging to the population is evaluated, then we select with high probability individuals with high fitness (*selection* phase). We then combine the chromosomes of the selected individuals in order to generate new individuals: the combining procedure is also called *crossover*. In order to overcome the possibility to stuck in a local optimum, we introduce a stochastic *mutation* of the chromosome of the individuals - which consists in modifying

Algorithm 12: Genetic Algorithm

```
Start: Generate random population with $n$ chromosomes;
repeat
    Fitness: Evaluate the the fitness of each chromosome in the
      population;
    newPopulation ← {};
    for $n - k$ times do
        Selection: Select two parents from the population according
          to their fitness;
        Crossover: With a crossover probability cross over the
          parent to form a new offspring;
        Mutation: With a mutation probability, mutate some allele
          in the individual;
        newPopulation ← newPopulation ∪ {newIndividual};
    end
    Elitism: Add to newPopulation the $k$ best individuals from
      population;
    population ← newPopulation;
until some criterion is accomplished;
```

randomly with low probability some chromosomes. Sometimes, in order to preserve the optimal solution found we select the k best individuals at each epoch: this practice is called *elitism.*

In our experiments we used a genetic algorithm to infer the parameters $\delta_{\min}$ and *patience* for the early stopping and also the activation function of a neural network composed by two layers of 10 neurons each, with the goal to find a configuration that was able to well represent the value function with the classic FQI. The chromosomes were encoded with five genes: $(f, a_{\delta_{\min}}, b_{\delta_{min}}, a_{patience}, b_{patience})$. The activation function was encoded by f which can assume three different values: 0 for tanh, 1 for sigmoid and 3 for relu. $\delta_{\min}$ is encoded as $\delta_{\min} = a_{\delta_{\min}} 10^{b_{\delta_{\min}}}$ where $a_{\delta_{\min}} \in \{10, 11, \dots, 99\}$ and $b_{\delta_{\min}} \in \{-3, -4, \dots, -15\}$. *patience* is encoded $\delta_{\min} = a_{patience} 10^{b_{patience}} + 1$[1] where $a_{patience} \in \{10, 11, \dots, 99\}$ and $b_{patience} \in \{-1, 0, 1\}$. This way to encode $\delta_{\min}$ and *patience* gives us the chance to explore a big range of numbers using two digit precision. The selection is exploited in a very naïve way: the parents of the new individuals of the future generation are selected with equal probability from the 50% best in-

[1] We add $+1$ in the formula to avoid the case that of a *patience* $= 0$.

dividuals. This selection process turned out to be effective in our problem. There are several ways to exploit crossover when we use integer numbers to encode chromosomes. We sampled the genes of the new individual randomly from the interval between the values for that gene of the parents: this seem a good way to mix exploration and exploitation. The probability of introducing a mutation in a new individual is quite high ($p = 0.8$), and when a mutation happens, every gene has a probability of 0.2 to mutate: this allows also multiple mutations in a chromosome. A mutation with probability 0.8 introduces an high rate of exploration, but this effect is balanced with the presence of elitism, which preserves the 20% of the best individuals in the next population and from the selection, which discards the 50% worst individuals. The size of the population is 20 and this value is chosen to best exploit parallelization on our server. The fitness function is computed as the mean discounted reward at the last iteration in 10 different runs of FQI on the chosen environment, using different datasets from the ones used in our experiments (but generated in the same way). The choice of averaging the discounted reward was both to provide stability in the genetic algorithm (the fitness procedure is less variable), and also in order to prevent the genetic algorithm to find parameters that works well only for a particular dataset. We want to underline that the genetic algorithm finds the parameters (activation function, $\delta_{\min}$ and *patience*) that optimize the performance of FQI: in this way, since our objective is to show that B-FQI works better (under same conditions) with respect to FQI, we provide pessimistic results, and when B-FQI performs better than FQI it can not be imputed to the introduction of this optimization procedure.