POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMATICA E BIOINGEGNERIA
MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

IMPORTANCE SAMPLING BASED TRANSFER IN REINFORCEMENT LEARNING

Master Thesis of:
Andrea Sessa

Mat. **850082**

Supervisor:
Prof. Marcello Restelli

Co-supervisor:
Dott. Matteo Pirotta

Academic Year 2016 - 2017

Words, words were truly alive on the tongue, in the head
Warm, beating, frantic, winged; music and blood
But then I was young.

Carol Ann Duffy, Little Red Cap

Ringraziamenti

Desidero innanzitutto ringraziare il Prof. Marcello Restelli senza il cui aiuto e sostegno la stesura di questa tesi non sarebbe stata possibile. Ringrazio il mio co-relatore Matteo Pirotta e il dottorando Andrea Tirinzoni, i loro consigli, discussioni e critiche hanno contribuito a rendere unico e interessante il presente lavoro.
Ringrazio, ovviamente, la mia famiglia: mio padre Pietro, mia madre Rosanna e mio fratello Davide senza il cui sostegno morale (ed economico) tutto ciò non sarebbe mai stato possibile.
Ultimi, ma non meno importanti, ringrazio gli amici Filippo, Giorgio e Nicola che, in questi ultimi mesi, mi hanno supportato (e sopportato). La loro compagnia è stata fondamentale per arrivare in fondo a questo lungo percorso. co

Andrea
Jerago, 21 Dicembre 2017

Contents

List of Figures

List of Algorithms

Abstract

In the recent years we have seen a general advance in the field of Artificial Intelligence. In particular Reinforcement learning (RL) has demonstrated itself as one of the most promising branch of machine learning especially in the last 20 years.
RL success is mostly due to its applicability in large and complex control problem (robotics, economics, AI and so on) where the high complexity of the models involved makes impossible, for the human designer, to design a complete and efficient model. In all these different scenarios RL techniques are able to provide a sufficiently accurate estimation of the model dynamics.
A major problem in Reinforcement Learning is represented by the amount of knowledge required to accurately train the agent in a specific environment. A possible approach to this problem is Transfer Learning. The idea is that if the agent has already solved a set of source tasks, then we can use this knowledge to speed-up the learning performance of the agent over the target task. Depending on the specific scenario the transferred knowledge can have different shapes and may require different hypothesis over the set of tasks involved.
We propose a sample-based approach for transfer learning in Batch RL based on the idea of Importance Sampling. For each experience sample collected from the set of the source tasks, we calculate a pair of importance weights w_p and w_r. These weights can be interpreted as correction factors of the sample for reward (w_r) and transition (w_p) models. A low weight means a transition or reward generated in a source task has a very low probability to be observed in the target task and therefore very likely to negatively bias the learning performance (also referred as negative learning). On the other hand, a high weight value means that the sample reward or transition has a high likelihood to be generated in the target and therefore positively speed-up the learning performance.
We apply this idea to a specific batch RL algorithm, namely Fitted Q-Iteration (FQI), using the weights inside the regression algorithm obtaining Weighted Fitted Q-Iteration (WFQI). We provide a procedure to produce an accurate estimation of the weights for each sample. Moreover we also theoretically analyze the properties of our approach and we prove results bounding the amount of bias introduced by the use of source samples. Finally, we empirically validate WFQI over three differ-

ent reinforcement learning classic benchmarks observing a significants improvements in terms of learning speeds and rejection of the negative transfer effect.

Sommario

Negli ultimi anni abbiamo visto un avanzamento generale nel campo dell' Intelligenza Artificiale. In particolare il campo dell'Apprendimento per Rinforzo (RL) ha dimostrato di essere uno dei più promettenti rami in Machine Learning degli ultimi 20 anni.
Il successo di RL è dovuto in larga parte alla sua applicabilità in grandi e complessi problemi di controllo (robotica, economia, AI etc.) dove l'alta complessità dei modelli coinvolti rende impossibile, per un designer umano, una progettazione completa ed efficiente. In tutte queste situazioni le tecniche di apprendimento per rinforzo sono in grado di fornire una stima sufficientemente precisa di tutte le dinamiche del modello.
Uno dei principali problemi in RL è rappresentato dalla grande quantita di esperienza necessaria per un addestramento accurato dell'agente in uno specifico ambiente. Un possibile approccio è rappresentato dall'utilizzo di tecniche di Transfer Learning; l'idea è che se l'agente, in passato, ha già acquisito la conoscenza necessaria a risolvere un insieme di task sorgenti, questa conoscenza può essere riutilizzata per accelerare l'apprendimento su un dato task obiettivo. A seconda dello specifico scenario la conoscenza transferita può assume forme differenti e potrebbe richiedere differenti ipotesi sull'insieme di task coinvolti.
In questa tesi proponiamo un approccio per il transferimento di sample basato sull'idea dell'Importance Sampling. Per ogni campione proveniente dai task sorgenti calcoliamo un a coppia di pesi w_r e w_p. Questi pesi possono essere interpretati come fattori di correzione dello specifico sample per rinforzo (w_r) e dinamica (w_p). Un peso di basso valore indica una transizione o rinforzo, generati in un task sorgente, con una bassa probabiltà di essere osservati nel task obiettivo e perciò estremamente incline a peggiorare la performance di apprendimento nel task medesimo (altrimenti noto come negative transfer). Dall'altro lato un peso di valore elevato (≥ 1) indica che il rinforzo o la dinamica del campione saranno verosimilmente osservabili all'interno del task obiettivo o perciò possa positivamente polarizzarne la performance di apprendimento.
In questo lavoro applichiamo questa ideal ad uno specifico algoritmo di batch reinforcement learning, precisamente Fitted Q-Iteration (FQI), usando i pesi all'interno dell'agoritmo di regressione ottenendo quello che chiamiamo Weighted Fitted Q-Iteration (WFQI). In questa tesi proponiamo un procedura per ottenere una stima accurata dei pesi. In aggiunta

analizziamo il nostro approccio da un punto di vista teorico provando un risultato dove limitiamo il bias introdotto dall'uso di campioni sorgenti. Infine, validiamo empiricamente il nostro algoritmo su una serie di tre differenti benchmark osservando dei miglioramenti significativi in termini di velocità di appredimento e di reiezione dell'effetto del negative transfer.

Chapter 1

Introduction

REinforcement learning has demonstrated itself as one of the most promising branch of machine learning especially in the recent years. RL success is mostly due to its applicability in large and complex control problem (robotics, economics, AI and so on) where the high complexity of the models involved makes impossible, for the human designer, to design a complete and efficient model. In all these different fields RL techniques are able to provide a sufficiently accurate estimation of the model dynamics.
A generic RL algorithm permits an agent to *learn* a specific *task* by rewarding the agent when, during the training, it performs an action that brings it near to the goal or by punishing it when the action chosen bring it away from the goal (as described in Figure 1.1).

Many techniques exist to model such types of situations, one of the most popular and studied are the Markov Decision Processes (abbreviated as MDP). This formalism (we will give a more precise definition in chapter

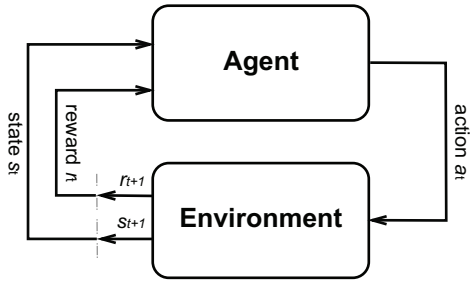

Figure 1.1: *Schematization of the agent/environment interaction in RL*

2) permits to model the problem as a collection of states and actions. Besides a reward function is also defined and used by the agent during the training phase.
When both the model of the problem, that is when the probability distribution given a state-action pair is known to the designer of the problem, and when the reward function is known, well established and efficient techniques are available to optimally solve the problem (eg. Dynamic programming).

Unfortunately in many real-world situations the model and the reward function associated to the MDP are not known and need to be learned by the agent(*model-free*). In this context the term *"solving the problem"* acquires an additional meaning: we want still to determine the optimal policy but at the same time we want also learn the dynamics and reward function of the task. Some techniques do not even need to learn an explicit representation of reward and transition models but just need an intrinsic schematization of them. For example, Q-Learning (or Fitted algorithms) build at each iteration a representation of the Q function of the problem without building an explit representation of rewards and dynamics.

A big problem in the model-free approach is the enormous amount of samples required to learn a good policy, this is true especially in many real world situation where the agent has a limited amount of time and resources to spend on the learning phase. Moreover every time the task at hand changes the entire learning process must be restarted from scratch even if very similar tasks have been already solved by the agent.
This problem motivates the need to have the possibility to transfer and reuse knowledge across different tasks. This idea naturally arises from psychology and cognitive science research (early 90s) where humans are shown to learn a specific task better and faster if they have already solved similar tasks in the past.
In the general machine learning framework the idea of the transfer of knowledge has already been successfully applied to problems in supervised learning (such as recommender systems, text classification, speech classification, etc.).
In the last 10 years the research on transfer learning has started to focus also on reinforcement learning applications.
In the RL field the main idea is that if the agent already knows the solution of a set of *source tasks*, it can reuse that knowledge to speed up the learning of a new tasks (ie. the *target task*).
Every transfer RL algorithm must answer two questions: *what* and *when* to transfer?
The first question can be answered in different ways: one possibility is to transfer instance of knowledge(ie. samples) from the source MDPs to the target MDP ([10]), another possibility is representation transfer where

the transfer algorithm changes the representation of the solution of the source tasks ([19]), finally we have the possibility offered by parameters transfer where what we transfer is an initialization for the parameters of the learning algorithm of the target task (refer to [8]).
This work focuses on the transfer of instances.
The second question highlights a very important point: if we focus on the transfer of samples between tasks, the transfer should only occur when the source task presents some similarities with respect to the target task, if not a phenomena, known in literature as *negative transfer*, may occur. Negative transfer might actually worsen the performance of the learning algorithm over the target task and therefore must be avoided as much as possible(a possible solution is presented in [10], besides the problem will be deeply discussed in this thesis).

When considering the problem of the transfer of knowledge across multiple tasks, a series of natural questions must be answered: Where to fetch the next sample? From which task? Should it be fetched from a source task or directly from the target task?. Methods have been proposed in the past, such has [10], where the authors introduce an algorithm which estimates the similarity between source and target tasks and selectively transfers from the source tasks which are more likely to provide samples similar to those generated by the target task.
Indeed many times the actual environment in which the agent lives is expensive, time-consuming and sometimes dangerous. The prohibitive cost of collecting samples in the real environment(ie. the target task) makes difficult (sometimes impossible) to learn a good policy, again this motivates the need for the reuse of knowledge.

Our approach differs from previous works: we try to perform transfer learning exploiting a very well known idea of statistics: importance sampling. All the samples are transferred to the target task but each sample is enriched with some information that permits the learning algorithm to correct the bias introduced inside the target.
More precisely we will propose an extension of well studied learning algorithm (Fitted Q-Iteration) to permit the reuse of experience samples. The algorithm, called **W**eighted **F**itted **Q-I**teration (from now on abbreviated as WFQI), tries to build an high level representation of reward and transition model of the task at hand and produces an estimation of the relevance (under the form of a weight) of the source samples with respect the target task. The resulting weights are then used, along with the source/target samples, in FQI with a weighted regression algorithm to build an accurate estimation of the Q function.
It is always important to remember that samples from the source tasks are cheap but they may negatively *bias* the agent in the target tasks(ie. negative transfer), on the other hand samples coming from the target

task are expansive(they require the agent to perform trajectories over the real environment) but are unbiased since they come from the same problem the agent is trying to solve.

1.1 Mission and Contributions

The mission of this thesis is to produce a novel method to exploit the reuse of knowledge in the solution of new tasks. This is done by modifying the standard Fitted Q-Iteration algorithm to permit the transfer of samples. We seek for the optimal tradeoff between the number of samples coming from the source tasks and the number of samples coming from the target task by controlling the amount of bias introduced in the target task versus the reduction in variance brought by the transferred samples.
Finally we try to prove a series of theoretical results over the performance of the above mentioned algorithm.

The contributions of this thesis can be summarized as follows:

- **Algorithmical** contribution: We introduce a modification of the standard Fitted Q-Iteration algorithm (WFQI) that makes possible the transfer of knowledge across multiple source tasks and we empirically demonstrate the performance
- **Theoretical** contribution: We develop a theoretical analysis of the Weighted Fitted Q-Iteration (WFQI) to understand performance and limitations of such algorithm.

1.2 Contents Outline

This thesis is structured as follows:

- **Chapter 2** formally introduces the reader to the concept of Markov Decision Process and its application to reinforcement learning problems. In this chapter we introduce part of the notation that will be used in the rest of the thesis.
- **Chapter 3** provides an overview over the transfer learning framework including all the necessary mathematical notation. A short summary over the principal related works is included. This chapter also introduces the Fitted Q-Iteration algorithm.
- **Chapter 4** presents the idea behind importance sampling and formally introduce the framework to permit the transfer of knowledge. Additionally we proposes a modification of the standard FQI algorithm to allow the use of transferred samples.

- **Chapter 5** theoretically validates and motivates the proposed approach.
- **Chapter 6** provides and validates the performances of the algorithm described in the previous chapter.
- **Chapter 7** draws the conclusions and proposes some future line of work.
- **Appendix A** provides some additional details regarding the implementation of WFQI.
- **Appendix B** provides a theoretical background for Gaussian Process introducing the main theoretical and practical results of interest for this thesis.

Chapter 2

Reinforcement Learning

This chapter aims to introduce the reader to the problem of model-free Reinforcement Learning by providing a clear formalization of the main key concepts and the notation that will be used in the rest of the thesis. Moreover in the final part we introduce Fitted Q-Iteration, the state of the art for solving Batch RL problems.

2.1 Markov Decision Processes

In our context a task is identified as a discrete-time Markov Decision Process (abbreviated as MDP).
An MDP can be formalized as a tupe $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ where:

- $\mathcal{S}$ is the set of the states (we assume it to be finite).
- $\mathcal{A}$ is the set of all possible actions in every states (we assume it to be finite)
- $\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the transition model that assign to each state-action pair a probability ditribution(Π) over the state space.
- $\mathcal{R}$: $\mathcal{S} \times \mathcal{A} \to \Pi(\mathbb{R})$ is the reward function that assign to each state-action pair a probability distribution over $\mathbb{R}$.
- $\gamma \in [0, 1]$ is the discount factor, which determine the sensibility of the agent toward future rewards.

At each time step the agent interacts with the external environment according to its current policy, π: $\mathcal{S} \to \Pi(\mathcal{A})$, which given a state return a

probability distribution over the action space.
How the agent can learn a policy? The objective of the agent is to maximize the expected sum of discounted rewards, that means to learn a policy π^* that leads to the maximization of the *utility* of the agent in each state of the MDP.

Definition 1 (Bellman Operator for V)**.** *Given a policy π and a function $V : \mathcal{S} \to \mathbb{R}$, the Bellman operator $\mathcal{T}$ is defined as:*

$$(\mathcal{T}^\pi V^\pi)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s) \right) \tag{2.1}$$

In a MDP the concept of utility can be formalized introducing the definition of *value function*:

Definition 2 (Value Function)**.** *The state–value function $V^\pi(s)$ of an MDP is the expected returns starting from state s, and then following policy π. It is defined as the unique solution of the following Bellman equation:*

$$\mathcal{T}^\pi V^\pi = V^\pi \tag{2.2}$$

which is a fixed point for the Bellman operator $\mathcal{T}$.

Given an MDP the agent is interested in learning a policy that maximizes $V(s)$ in all the states s, that is:

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*(s') \right\} \tag{2.3}$$

where $R(s,a) = \mathbb{E}[\mathcal{R}(s,a)]$
Equation 2.3 is also known as Bellman optimality equation for V.

However for control purposes, rather than the value of each state, it is easier to consider the value of each action in each state.

Definition 3 (Bellman operator for Q)**.** *Given a policy π and a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the Bellman operator $\mathcal{H}$ is defined as:*

$$(\mathcal{H}^\pi Q^\pi)(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a') \tag{2.4}$$

Definition 4 (Q-Function)**.** *The action–value function $Q^\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π. It is defined as the unique solution of the following Bellman equation:*

$$\mathcal{H}^\pi Q^\pi = Q^\pi \tag{2.5}$$

which is a fixed point for the Bellman operator $\mathcal{H}$.

Again the agent will try to learn a policy that maximizes Q^{π} overs the state-action space:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = R(s,a) + \gamma \max_{a' \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) Q^*(s',a') \tag{2.6}$$

where $R(s,a) = \mathbb{E}[\mathcal{R}(s,a)]$
Equation 2.6 is also known as Bellman optimality equation for Q.

In this context a *deterministic* optimal greedy policy can be obtained by choosing the action a that maximizes $Q(s,a)$ in each state s:

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

These equations can be easily generalized to the case where the action and/or the state space are continuous.

2.1.1 Learning in a Markov Decision Process

Despite the simple definitions given in the previous section, the problem of learning an optimal policy in a MDP is very complicated and well known in literature. Many techniques have been introduced in the past (for more references see [21]) adopting different approaches and ideas. We propose a categorization on the base of the following properties:

- **Model-Free** vs **Model-Based**: A model-free technique tries to learn a controller (ie. an optimal policy) without previously learning a model of the environment, on the other hand a model-based technique tries first to learn a model of the environment and then tries to learn the optimal policy over the model. See [1] for a detailed analysis.
- **On-Policy** vs **Off-Policy**: In general when a learning agent uses at least two policies: one (π) is the approximation of the optimal policy a the current iteration and the second ($\hat{\pi}$) is the exploration policy used to get new samples from the environment. In a off-policy algorithm π and $\hat{\pi}$ are distinct so the optimal policy is learnt from samples coming from a different policy ($\hat{\pi}$). In a on-policy algorithm π and $\hat{\pi}$ coincide, the same policy is used both for exploring the environment and at the same time to approximate the optimal policy. See [18] as an example of Off-policy algorithm and [20] for an analysis of On-policy algorithms.
- **Online** vs **Offline**: In an online setting the algorithm processes samples immediately after they have been obtained from the environment (and before the next sample will be sampled), on the other hand in an offline learning algorithm we have two distinct phases:

first the algorithm gather a sufficiently large set of samples (ie. a batch) and only in the second phase this batch is used to learn the optimal policy. These two phases can be repeated multiple times during the learning.

- **Tabular** vs **Function Approximation**: This distinction is related to the internal representation of the models used by the algorithm. Precisely there are algorithm that represents the value of each state of the MDP under the form of a table, this approach has the advantage to be very simple but it is often infeasible when the number of states and actions is too large (for example in infinite MDPs). On the other side the function approximation approach uses a function to associate values to state; in this way the algorithm is able to scale also when the number of states and actions is very large but is much more complicated to implement and mantain with respect to a tabular representation. Refer to [7] for an implementation of Q-Learning with Neural Networks.

- **Value Based** vs **Policy Based**: A value-based algorithm tries to learn the optimal policy by iterating of the space of the value function thanks to the optimality bellman equation (remember that optimal value function is a fixed point of the optimal bellman operator). In an policy based approach the algorithm learns the optimal policy iterating over the space of the policies; this is done by a series of value function evaluation (using the policy of the previous step) and policy improvements (starting from the value function) steps. As an example of policy based methods, consider policy gradient algorithms described in [22].

Regarding the case where both the state and action space are finite and discrete, the simplest algorithms known in literature are Monte Carlo (MC) and Temporal Difference (TD) methods. Both are example of model-free algorithm.
Monte Carlo methods try to learn the optimal policy by sampling *full* random trajectories from the environment, the optimal policy is learned by using samples coming from the trajectories. Temporal Difference, on the other hand, does not need to observe a full trajectory to learn something; leveraging the markov property of the problem TD is able to learn the optimal policy from the single transitions observed along the partial trajectory. Many variations of TD and MC are know (offline/online or on-policy/off-policy) and many recents algorithm are based on the same ideas.
More relatively news techniques includes, for example, SARSA and Q-Learning; the first (SARSA) is based on the idea of applying Temporal Difference learning on the update of the Q function of the MDP obtaining a simple updating rule for Q. Q-Learning is usually performed off-policy, it learns an optimal Q function using a simple update rule that is able to

include, at each step, new information coming from the environment.

The problem with the discrete case is related to its applicability to real-world problems. This is why we also consider the problem of learning in a slightly more general case that is when the action space still remains discrete and finite but the state space becomes continuous and infinite. It may appear that the finite number of actions is still a too big limitation but there exists many scenarios where only a finite and discrete set of actions are available to the agent; for example consider the problem of learning how to play a game in which the agent need to traverse a maze without falling inside traps along the path; in this situation there are only four action available to the agent but the state, ie. the position of the agent inside the environment, is continuos.
The simplest technique to deal with a continuous state space is to discretize it and then resort to the algorithms described for the discrete case. Unfortunately the discretization of the state space may lead to several disadvantages: a coarse discretization might lead to lose important information present inside the state space and produce a bad learning performance on the other hand a too fine discretization may lead to a huge number of finite states and again produce a bad performance.
More sophisticated techniques do not try to transform the MDP into a discrete one but rather try to use some approximators to produce a reliable estimation over the entire state space.
These techniques are divided in:

- **Value Approximation**: In the case of value-approximation algorithms, samples are used to update an estimator of the value function (or also the Q function depending on the specific algorithm) that gives an approximation of the current or the optimal policy. Many reinforcement-learning algorithms fall into this category. Important differences between algorithms within this category is whether they are on-policy or off-policy and whether they update online or offline.

- **Policy Approximation**: The idea is to store the current policy and use the experience sample to update such policy to approximate the optimal one. Algorithms that only store information about the policy are called *direct policy-search* while algorithms that store both the value and the policy are known as *actor-critic* methods.

2.1.2 Batch Mode Reinforcement Learning

One of the main drawbacks of online reinforcement learning is the amount of experience needed to optimally solve a task. In order to overcome this problem batch approaches have been proposed in literature. The main idea behind batch RL is to distinguish between an exploration phase that collects the sample from the environment in the form of tuples (s, a, s', r), called *sampling phase*, and the learning algorithm that learns

the optimal policy on the basis of the samples collected in the sampling phase. The only difference with respect to online RL is that the samples are not immediately used in the learning procedure as soon they are observed but they are first collected, producing what is known as a *batch*, and then used in the learning procedure.
It is also important to remark that no assumption are made on how the samples are collected from the environment. They can be generated by gathering the four-tuples corresponding to one single trajectory (or episode) as well as by considering several independently generated one or multi-step episodes. This concept is of extreme importance for this thesis, indeed Fitted Q-Iteration (presented in the next sections),falls exactly in the category of batch RL algorithms and our main result WFQI is obtained as a variance of FQI.

2.2 Fitted Q-Iteration

In this section we introduce a key element of this work: the Fitted Q-Iteration algorithm (abbreviated as FQI from now on).
The FQI algorithm is a *batch-mode* reinforcement learning algorithm which tries to build an approximation of the Q function. This is done by iteratively extending the optimization horizon up to a certain number of iterations.
The first important property is that it assumes a batch learning model, that is the learning process is divided into two parts: in the first part samples are collected by the agent from the environment, in the second phase the collected samples are used to produce a reliable estimation of the Q function.
The second important property is the use of a regression algorithm in the learning phase of the algorithm: FQI formulates the Q function determination problem as a regression problem. This simple idea permits the algorithm to take full advantage in the context of reinforcement learning of any regression algorithm used inside FQI; this is not possible with standard stochastic approximation algorithms which can only use parametric approximators (eg. neural networks).
In the following we first introduce some of the works related to FQI and then we proceed to formally present and motivate the final version of the algorithm. Moreover, the main theoretical properties and results are presented and discussed.

2.2.1 History and Related Works

The idea of trying to approximate the Q function of a MDP starting from experience collected in batch and exploiting the capability of a supervised learning algorithm has been deeply investigated starting from 2002 with the work by Ormoneit and Sen [14] where the authors propose an algo-

rithm aimed to solve large-scale *dynamic programming* tasks. As already highlighted the main difference, with respect to a reinforcement learning task, is that in dynamic programming the model of the environment is assumed to be known to the agent. Therefore the algorithms first proposed by Ormoneit and Sen and successively by Gordon [6] are known as *Fitted Value Iteration*.
More recently in Ernst [4] a series of algorithm closely related to Fitted Q-Iteration are given. These algorithms, that fall under the category of model-based methods, use the idea of building an auxiliary markov decision process using the data collected and then, exploiting the solution of such MDP, they adjust the parameters of the approximation architecture used to estimate the Q-Function of the original task. They also showed that such procedure is completely equivalent to the application of FQI when the states of the MDP correspond to a partition of the original state space and using as regression method simply the average of the output of the training samples that belong to the same partition.
In Boyan [2] the author proposes the *Least-square temporal difference* (LSTD) algorithm that results to be very similar to FQI when a linear regression method is used.
Finally in 2005 Ernst et al [5] present the current version of fitted q-iteration in combination with tree-based regression techniques. The results presented by the authors are summarized in the rest of the chapter.

2.2.2 The Algorithm

A pseudocode version of the algorithm is presented in Algorithm 1.

Algorithm 1 Fitted Q-Iteration algorithm

procedure FQI($\mathcal{D} = (s, a, s', r)_{k=1}^{N}, myRegressionAlg$)
  $k \leftarrow 0$
  $\hat{Q}_k(s, a) \leftarrow 0 \; \forall (s, a) \in S \times A$
  $\mathcal{D}' = \emptyset$
  while $checkStoppingCriteria()$ **do**
    $k \leftarrow k + 1$
    for $l = 1$ to N **do**
      $i_l = (s_l, a_l)$
      $o_l = r_l + \gamma \max_{a' \in A} \hat{Q}_{k-1}(s'_l, a'_l)$
      $\mathcal{D}' \leftarrow \mathcal{D}' \cup (i_l, o_l)$
    $\hat{Q}_k \leftarrow myRegressionAlg(\mathcal{D}')$

The algorithm assumes the N experience samples to be already collected according to some policy (eg. random). The Q function of the original task is initialized to 0. From $\mathcal{D}$ the algorithm build an auxiliary dataset $\mathcal{D}'$ that will be used by the regression algorithm (myRegressionAlg) to

build a model of the Q-function of the MDP.
Notice also that the first iteration of the algorithm, given that $\hat{Q}_i(s,a)$ is initialized to 0 for every state-action pair, is used to produce an approximation of the expected reward that is: $\hat{Q}_1(s,a) = \mathbb{E}[r(s,a)]$. In successive iterations only the output part of the input/output pairs (i_l, o_l) that constitutes $\mathcal{D}'$ is updated, this is done using the estimation of the Q-function produced at the previous step and combining it, through the Bellmann equation, with the reward and the next state reached in each specific tuple.
It is also important to notice that the different calls to the supervised learning algorithm are totally independent from each other; therefore it is possible to dynamically change the complexity of the regression procedure at each step to reach the best possible bias/variance tradeoff, given the available samples.

2.2.3 Algorithm Motivation

Let first provides an alternative definition for the Bellman operator (for Q). This definition is completely equivalent to 2.4 As usual let $\mathcal{S}$ to be the state space and $\mathcal{A}$ the action space. Let's define H as an operator mapping any function $\mathcal{F} : S \times A \to \mathbb{R}$ and defined as:

$$(H\mathcal{F})(s,a) := \mathbb{E}_w[r(s,a,w) + \gamma \max_{a' \in \mathcal{A}} K(f(s,a,w), a')] \tag{2.8}$$

where the function f represents the dynamics of the environment and w represents a stochastic disturbance that can affect the next state reach by the agent. H is also known as Bellman operator.
The main result is the following:

Theorem 1 (Ernst [5])**.** *The sequence of Q_N-functions defined on $S \times A$ by*

$$\begin{aligned} Q_o(s,a) &= 0 \\ Q_N(s,a) &= (HQ_{N-1})(s,a) \end{aligned} \tag{2.9}$$

converges (in infinity norm) to the Q-function defined as the unique solution of the Bellmann equation

$$Q(s,a) = (HQ)(s,a) \tag{2.10}$$

Let's first consider the deterministic case

$$\begin{aligned} s' &= f(s,a) \\ r &= r(s,a) \end{aligned} \tag{2.11}$$

that is both the transition and reward model are not affected by noise, then equation 2.8 can be rewritten as

$$Q_N(s,a) = r(s,a) + \gamma \max_{a' \in A} Q_{N-1}(f(s,a), a') \tag{2.12}$$

Suppose now that a dataset of experience samples is available $\mathcal{D} = (s_i, a_i, s'_i, r_i)_{i=1}^{\#\mathcal{D}}$. The simplest idea is that we can use Q_{N-1} and $\mathcal{D}$ to obtain Q_N. Given a tuple (s_i, a_i, s'_i, r_i) we have:

$$Q_N(s_i, a_i) = r(s_i, a_i) + \gamma \max_{a' \in A} Q_{N-1}(s'_i, a') \tag{2.13}$$

The problem is that we are dealing with a continuous state space and the samples inside $\mathcal{D}$ do not permit the calculation to generalize over the entire $S \times A$.
To achieve a generalization over the entire state-action space we use the samples inside $\mathcal{D}$ in conjunction with equation 2.13 to build an auxiliary dataset:

$$\mathcal{D}' = \{((s_k, a_k), Q_N(s_k, a_k)), k = 1 \cdots \#\mathcal{D}\} \tag{2.14}$$

Then we can apply a totally generic regression algorithm to $\mathcal{D}'$ and obtain a model that is able to generalize well of the entire state-action space. This model can now be used to obtain a sound estimation Q_N. The process can be repeated up to a certain number of iteration to obtain increasingly good estimation of the Q-function.

The stochastic case can be treated in similar way: the problem is that for some samples (s_i, a_i, s'_i, r_i) the right hand side of equation 2.12 is no more equal to $Q_N(s_i, a_i)$ due to the presence of noise in both the transition and reward model. We can think of $r(s_i, a_i) + \gamma \max_{a' \in A} Q_{N-1}(f(s_i, a_i), a')$ as a realization of a random variable whose expected value is $Q_N(s_i, a_i)$. This is not a problem given that a generic regression algorithm always search for an approximation of the expectation of the output variables conditioned by the input and will return again an estimation of Q_N over $S \times A$.

Once the approximation of the Q-function is available a stationary deterministic control policy for the agent can be simply obtained by:

$$\pi^*(s) = arg \max_{a \in A} Q_N(s, a) \tag{2.15}$$

Some attention must be paid in the $\max$ operator used in the previous equations: when the action space is finite we can simply enumerate for every action the value of Q_N and simply take the maximum. On the other hand when we generalize to the case where the action space is continuous the greedy action selection used for the policy improvement step may become particularly problematic and expensive to be computed at every iteration. Authors in [13] propose an alternative to the $\max$ operator (a soft-greedy action selection) that, in practice, appears to yield an efficient version of the FQI algorithm also in the case of continuous action space.

2.2.4 Stopping Criteria and Convergence

A stopping condition is needed to understand at which iteration the algorithm must be stopped. Suppose that after T iterations our FQI

algorithm yields a Q-function Q_T and let the relevant stationary policy be π_T^*. Our metric is given by the discounted expected reward J_∞^π with:

$$J_\infty^\pi(x) = \lim_{T\to\infty} \mathbb{E}_{w_t}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, \pi(s_t), w_t)|x_0 = x\right] \tag{2.16}$$

where x_0 is the starting state.
Let π^* be the optimal stationary policy for the MDP. A stopping criteria just need to find an upper bound to the difference $||J_\infty^{\pi_N^*} - J_\infty^{\pi^*}||$ (in infinity norm). Authors in [5] give the following inequality:

$$||J_\infty^{\pi_N^*} - J_\infty^{\pi^*}||_\infty \leq 2\frac{\gamma^N B_r}{(1-\gamma)^2} \tag{2.17}$$

where B_r is a bound over the rewards of the MDP. Fixed a desired accuracy (and B_r) equation 2.17 permits to derive an approximate number of iterations needed to reach it.
Another possibility requires to look at the distance between the Q-functions of two consecutive iterations (also known as increment). Unfortunately this criteria has no guarantees to work in practice given that, depending on the specific regression algorithm, we have no assurance that the sequence of Q-function will converge to the optimal one.

Regarding the convergence we first provide a formal definition:

Definition 5 (Ernst [5])**.** *The FQI algorithm is said to converge if there exists a function* $Q : S \times A \to \mathbb{R}$ *such that* $\forall \epsilon > 0$ *there exists an* $n \in \mathbb{N}$ *such that:*

$$||Q_N - Q||_\infty < \epsilon \quad \forall N > n$$

It can be proved that the above conditions are satisfied depending on the specific supervised learning algorithm used within FQI.
In particular given a generic dataset $\mathcal{D} = \{(i_t, o_t)\}_{t=1}^{\#\mathcal{D}}$ we require the regression procedure to satisfy the following conditions:

$$\begin{aligned} f(i) &= \sum_{t=1}^{\#\mathcal{D}} k_{\mathcal{D}}(i_t, i) * o_t \\ \sum_{t=1}^{\#\mathcal{D}} |k_{\mathcal{D}}(i_t, i)| &= 1 \quad \forall i \end{aligned} \tag{2.18}$$

where $k_{\mathcal{D}}$ represents the kernel used in the regression algorithm and that must not change from one call to the other within FQI.
Algorithms that satisfies conditions 2.18 are for example k-nearest neighbors, partition methods, locally weighted averaging and more generally kernel-based methods.

2.2.5 Tree-based Regression

As stated in the previous paragraphs we make no assumption about the particular regression algorithm that could be used inside FQI. In this work, our choice falls on the large family of methods known in literature as Tree-based regression.
Indeed, in practice (eg. [10]), the union between FQI and tree-based methods has often led to very robust and effective algorithms.

Characterization

Let $\mathcal{D}$ be the training dataset for the regression. The idea behind tree-based method is to partition the input space in regions then the algorithm will produce a constant prediction in each region, based on the average of the outputs of the variables in $\mathcal{D}$ that belong to that particular region. Let $P(i)$ be a function that assigns to each input i the region to which i belongs to and let I_R be the characteristic function of region R.

$$I_R(i) = \begin{cases} 1 & i \in R \\ 0 & \text{Otherwise} \end{cases} \tag{2.19}$$

Then the model produced by a single regression tree is given by the following equation:

$$f(i) = \sum_{t=1}^{\#\mathcal{D}} k_{\mathcal{D}}(i_t, i) * o_t \tag{2.20}$$

where the kernel $k_{\mathcal{D}}$ is defined by:

$$k_{\mathcal{D}}(i_t, i) = \frac{I_{P(i)}(i_t)}{\sum_{(a,b)\in\mathcal{D}} I_{S(i)}(a)} \tag{2.21}$$

It is common practice not to use a single tree (in particular for the case of Extra Trees) but to build an ensemble of them. The prediction of the ensemble for a given input is now the average of the predictions of the single trees.
Suppose to consider an ensemble of n regression trees with a training dataset $\mathcal{D}$. The dataset is partitioned in n parts, one for each tree. Let $\mathcal{D}_m$ be the training dataset for the m-th tree and also let $P_m(i)$ be the function that assigns to each i the region of the m-th partition it belongs to. Again the produced model is given by equation 2.20 but now the kernel $k_{\mathcal{D}}$ is given by:

$$k_{\mathcal{D}}(i_t, i) = \frac{1}{n}\sum_{m=1}^{n} \frac{I_{P_m(i)}(i_t)}{\sum_{(a,b)\in\mathcal{D}_m} I_{S_m(i)}(a)} \tag{2.22}$$

It is important to notice that both 2.21 and 2.22 satisfies convergence conditions 2.18.

Many different tree-based regression algorithms have been proposed in literature (see [5] for a quite comprehensive compendium). They differ by the number of regression trees they build (one or an ensemble), the way they grow the tree from a generic training set, and, for the methods that build an ensemble of trees, the way in which they partition the initial training dataset.
In the remaining part of this chapter, we focus on a very particular method, Extra Trees, that will be extensively used in the experimental section.

Extra Trees

Extremely **Ra**ndomized Trees is an ensemble method that builds a forest of trees. The main difference with other methods is that each tree is grown separately from the complete initial training dataset.
The algorithm was first proposed in [5].
To determine a test at a node, this algorithm selects K cut-directions at random and for each cut-direction, a cut-point at random. It then computes a score for each of the K tests and chooses among these K tests the one that maximizes the score. Again, the algorithm stops splitting a node when the number of elements in this node is less than a parameter $n_{\min}$. Three parameters are associated to this algorithm: the number M of trees to build, the number K of candidate tests at each node and the minimal leaf size $n_{\min}$. For the complete tree-building procedure refer to Appendix A of [5].

Chapter 3

Transfer Learning

THe key insight behind the idea of the transfer of knowledge comes from psychology: human beings learn a new task much faster if, in the past, they solved other similar tasks. Learning to drive a bike may facilitate learning to drive a car, learning to play organ may help in learning to play piano.
In the last years, data mining and machine learning techniques have achieved significant success in many knowledge-engineering areas such as classification, regression and clustering.
However, a large part of this techniques is based on the assumption that the training data and the test data come from the same distribution (ie. from the same task). The problem is that every time there is a significant difference between the two distribution the entire model must be learned again from scratch. In many real-world application it is impossible, every time the target task changes, to recollect all the needed data to train the new model. It would be nice to reutilize the knowledge (ie. data) collected for tasks similar to the target one.
A typical situation is when the data at hand may be easily outdated. In this case, the data collected at the begin of a period may not follow the same distribution at the end of the same period. Transfer learning may help in avoiding the effort of rebuilding the model every time.
Another field of application of transfer learning that has been deeply investigated in the last years is Reinforcement Learning. In this situation, the need of an efficient transfer of knowledge is even more evident. In many real-world scenario, (such as Robotics or applications involving human interaction) the amount time needed to learn a good policy from samples coming from the target tasks if often prohibitive.

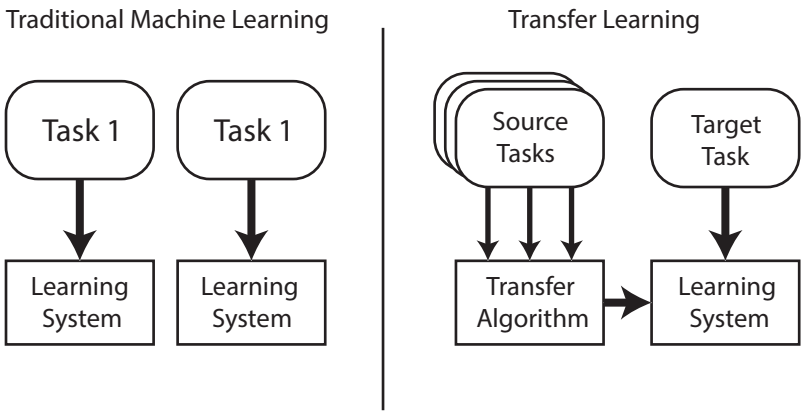

Figure 3.1: *Transfer Learning vs Traditional ML*

Transfer learning tries to reduce the learning time of an optimal policy by *reusing* knowledge coming from previously solved task (Figure 3.1).

3.1 History and Related Works

Scientific research on transfer learning has attracted more and more attention since 1995.
Earlier works focus on supervised learning mainly with application in classification and regression problems. For example [11] analyzes the problem of the transfer of knowledge under classification settings where the test and training distribution are mismatched. The authors propose a solution in which an auxiliary variable μ_i (for each sample (x_i^a, y_i^a) in the training dataset) is introduced to reflect the mismatch between the two distributions; when (x_i^a, y_i^a) is too different with respect the test distribution an appropriate choice of μ_i makes the final classifier less sensitive to the i$-th$ sample.
Another example, but applied to regression problems, can be found in [16]; here the authors try to develop one of the first general framework for transfer learning and analyze its correctness using the *Probably Approximately Correct (PAC)* learning theory. The paper proposes a modification of the classical boosting algorithm to successfully include samples coming from a mismatched distribution; the goal is to learn a high quality model using as much as possible samples coming from the auxiliary data source.
A more theoretical work is proposed by [23]; the authors explore a classification setting in which targets concepts are assumed to be drawn from a known distribution. The main goal was to study the total number of

sample required to learn all targets to an arbitrary specified expected accuracy. The paper shows an interesting connection with the theory of active learning that will be discussed in the next chapter.

Only in the last 10 years research on transfer learning has focused also on reinforcement learning. Works that have been published differ in the type of knowledge transferred (instances, models representation or parameters) or in the number and domain of source and target tasks. In this thesis we mainly focus on the transfer of instances from a set of source tasks to a single target task; considering this setting some interesting results are presented in [10] where the authors successfully show (empirically) that when the samples coming from the source tasks are sufficiently *symilar* to ones already collected for the target task, the transfer of instances is possible and permits to significantly reduce the time needed to learn an optimal policy on the target task.
A more theory-oriented work appears in [9], this paper represents one of the first attempt to provide a theoretical formalization of the sample-transfer problem, in particular the authors provide a series of theoretical bounds for different transfer algorithm showing the potential of knowledge transfer in a single-task learning setting.

3.2 Taxonomy

In this section, inspired by [8], we report a general classification of transfer learning based on: the setting, the transferred knowledge and the objective.

3.2.1 Settings

The settings regards the specification of the state-action domain for the tasks involved in the transfer. We can have three different cases:

- **Transfer from a single source task to a target task with fixed domain**. This setting (single source and target task) is often referred in literature as *inductive transfer learning*. In this particular situation, we assume that both the source and the target task share the same domain, ie. the same state-action space. The transfer algorithm might or might not have access to the target task knowledge during the transfer. In the case, no knowledge from the task is available the algorithm can only perform a very simple transfer (without regarding the problem of the negative transfer) and directly use this knowledge in the target task. On the other hand if some knowledge is available from the target task than a more complex transfer can happen: the algorithm can adapt the knowledge of the source to the knowledge already present in the target, for example by only

selecting, in the source task, those samples that are likely to be generated by the model of the target task.

- **Transfer across multiple tasks with fixed domain**. In this scenario we assume that many source tasks are available. Again, also in this case, all the tasks involved (target and sources) share the same state-action space. We expect that, as the number of source task increases, the performance of the target task to be much better compared with the previous case.

- **Transfer across multiple tasks with different domains**. This case represents the most general situation: multiple tasks are available and each task has a different state-action domain. In this scenario, to manage the complexity of the problem, many of the proposed algorithms reduces to the case with a single source and target task with different domains and focuses on the problem of defining an efficient mapping between the domains of the source and target task.

3.2.2 Knowledge

In every transfer learning algorithm, a central problem is how to define how knowledge is actually used to transfer information from the source task to the target task. A possible categorization proposed by [8] classify the possible knowledge transfer approaches in: instance transfer, representation transfer and parameter transfer:

- **Instance transfer**. In this scenario, the transferred knowledge assumes the shape of samples of trajectories collected in the source tasks. Each sample can be viewed as a tuple of four elements: the staring state s, the action taken in a, the next state s' reached after applying a to s and the relevant reward r. The transferred samples can now be used in the target task, in a model-free approach, for example, to build an accurate approximation of the value function of each state.

- **Representation transfer**. The idea is that each RL algorithm uses a specific representation of the task and of its solution. After the source tasks have learned an accurate solution, the transfer algorithm tries to abstract the different task-solution representations to permit the target task to take advantage of them. Examples range from the use of reward shaping functions to MDP augmentation through options.

- **Parameter transfer**. Every RL algorithm is characterized by a series of parameters that define its own initialization. The key idea is that a good initialization can provide a big advantage, in terms of time to convergence, to the RL algorithm running over the

target task. Examples can be the initial Q values for a Q-Learning algorithm or the learning rate α for a more generic algorithm. In some other situation it may be useful to adapt and change the parameters according to the target task.

3.2.3 Objectives

It happens often in machine learning (especially in unsupervised settings) to have difficulty in measuring the quality of the results. This is true also in transfer learning where several metrics have been defined:

- **Learning speed improvement**. Here the objective is to measure the gain of the transfer algorithm in terms of amount of experience needed to learn the target task. The complexity is then measured in terms of samples needed to learn the optimal policy, often referred in literature (especially in supervised learning settings) as sample complexity. This type of objective is common when the algorithm is based on the transfer of instances. In practice three different metrics are commonly used: *time to threshold*, *area ratio* and *finite-sample analysis*. Time to threshold measures how much experience (ie. samples) are needed to the algorithm to reach a fixed threshold, the main disadvantage is that the threshold may be arbitrarily chosen and bad representing the real situation. The area ratio is formally defined as the ratio between the learning curve with and without transfer, precisely:

  $$r = \frac{\text{area with transfer} - \text{area without transfer}}{\text{area without transfer}}$$

  while this metrics does not suffer from the disadvantage previously described, it is scale dependent; that is it depends on the unit of measure of the learning curve.
  The last metric represents a more theoretical approach: the idea is to derive bounds that strictly depend on the algorithm used, the number of samples available from the source task and the initialization parameters of the algorithm itself.

- **Asymptotic improvement**. In the major part of the applications of transfer algorithms, obtaining an optimal solution over the target task is often infeasible. In this situation, usually, we are interested in getting the solution that asymptotically achieves the best performances. This type of objective is usually targeted by representation transfer algorithm where the structure of the hypothesis space is transformed so to accurately approximate the solution of the target task. Up to date no transfer learning algorithm is guaranteed to improve the average approximation error with respect to a non-transfer algorithm.

- **Jumpstart improvement**. This type of objective is usually targeted by parameters transfer algorithms. As already mentioned above the performance of any reinforcement learning algorithm strongly depends on the initialization of the parameters. A bad initialization often negatively bias the target task and leads to longer times of convergence. Common ways to measure this type of improvements are often related to the comparison of the performance of the algorithm, in one case initialized with a suitable hypothesis and in another randomly initialized.

3.3 Bias-Variance tradeoff

In this section, we revise the theory behind of the bias-variance tradeoff. This concept represents a key idea in a general machine learning framework and plays a central role in this thesis. For this discussion we do not focus only on the reinforcement learning framework given that this concept is much more general.

The idea behind the bias-variance dilemma is that for any machine learning algorithm the prediction error can always be decomposed in the sum of three components:

$$TotalError = IrreducibleError + Bias + Variance$$

The irreducible error, as the term suggest, can not be reduced regardless the algorithm used. It is the error introduced from the particular realization of the training dataset $\mathcal{D}$, it represents the intrinsic error present in the data.
Assume $\mathcal{H}$ to be the space of all the possible concepts (id. functions, also referred as hypothesis) learnable by any machine learning algorithm. Suppose to fix a precise algorithm $\mathcal{A}$ and indicated with $\mathcal{C} \subset \mathcal{H}$ the set of concepts algorithm $\mathcal{A}$ can produce as output. The true model (ie. the concepts that we are trying to learn) $\hat{f}$ might be inside or outside $\mathcal{C}$. On the basis of the samples inside the dataset we can estimate an error function L that, given an hypothesis, it returns the generalization error. We indicate as $\bar{f}$ the concept learnt by $\mathcal{A}$ by minimizing L.
The situation is summarized by figure 3.2

The **bias** can be seen as the distance between the hypothesis learnt by $\mathcal{A}$ and the true concept $\hat{f}$. On the other side the **variance** can be seen as the difference between what the algorithm has learnt from a particular dataset and what the algorithm expect to learn.
Consider the case where we have a perfect estimation of L (ie. when the dataset $\mathcal{D}$ contains an infinite number of samples); in this case we have an extremely low variance while the bias depends wether $\hat{f}$ is inside or outside $\mathcal{C}$.

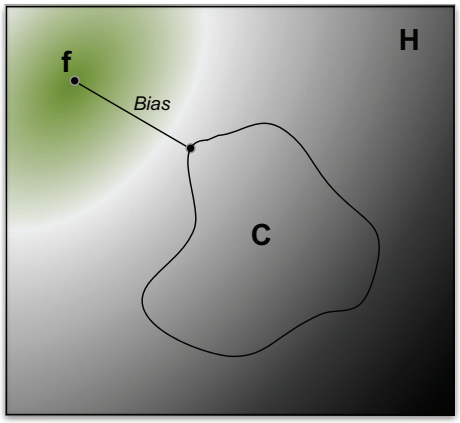

Figure 3.2: *The hypothesis space $\mathcal{H}$, the true function f and the error function (color gradient)*

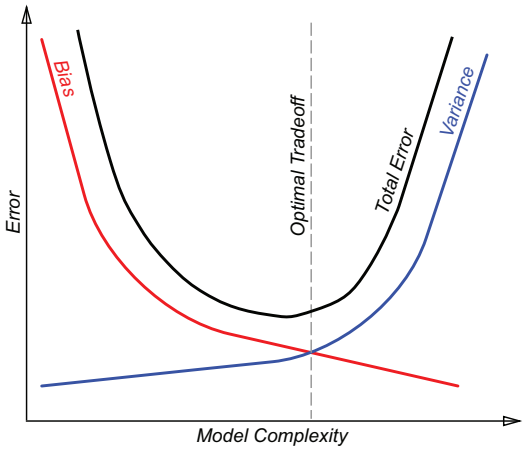

Figure 3.3: *The graph shows the relationship between bias,variance and total error with respect to the model complexity*

Suppose now that $\mathcal{D}$ contains a limited number of samples, then the estimation of L will not be perfect but it will depend on the number of samples available and on the complexity of $\mathcal{A}$; the higher the complexity of the model and the lower the number of samples the higher the variance (*overfitting*). The smaller the complexity of the model the higher the bias (*underfitting*). The idea is summarized in figure 3.3.
The *dilemma* consists in managing the complexity of the model, the power of the model and the number of samples in the dataset to reach a situation where both variance and bias are close to zero, that is minimize as much as possible the total error of the model.
As already highlighted the bias-variance dilemma is fundamental importance also in the context of transfer learning. When samples are transferred from a source task to a target task we are introducing a bias in the learning performance of the target, but at the same time, given that more samples are available for the learning algorithm, we are reducing the variance of the estimation. The objective of the next chapters will be to seek the optimal tradeoff between the bias increase and variance reduction.

3.4 Formal specification and Assumptions

In the following we provide the notation used in the rest of the work.

Definition 6. *A task T is an MDP defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_T, \mathcal{R}_T, \gamma)$.*

In our situation we have a set of source MDPs plus a target MDP: $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$ where $\mathcal{M}_1, \ldots, \mathcal{M}_{n-1}$ are the source MDPs and $\mathcal{M}_n$ is the target MDP.

In our framework we focus on the transfer of samples between the source MDPs and the target MDP; a sample in our context is a tuple $(s, a, s^{'}, r)$ where s and $s^{'}$ are states, a is an action and r is the reward associated with the transition s, a, s'.
We also assume that all the tasks (target included) share the same state-action space, so that no mappings are needed to transfer the samples.

Precisely we suppose that a number N_t of samples are available from the target task and N_s total samples are available from all the target task, with $N_t << N_s$.
Moreover, as already stated, a transfer algorithm cannot blindly transfer all the samples to the target task due to the possibility of a negative transfer; indeed samples coming from the target task bring a certain amount of variance reduction and are unbiased while samples coming from the source tasks still bring to a reduction over the variance but they are biased given that they come from a task that a priori might be very different from the target task. Therefore when transferring samples from the source tasks an estimation of the bias must be available and more precisely we say that the transfer is positive when the increment of the bias is lower than the decrement of the variance. In our case we provide a way to minimize the bias introduced by the transferred samples.

Chapter 4

Weighted Fitted Q-Iteration

In this chapter we introduce the central algorithm of this thesis: Weighted Fitted Q-Iteration (from now on abbreviated as WFQI from now on). The WFQI algorithm is a *batch-mode* reinforcement learning algorithm which, as the FQI algorithm described in Chapter 2, try to build an approximation of the Q-function of the MDP by iteratively extending the optimization horizon.
WFQI is built on top of FQI, it maintains all the properties described in the previous chapters but it also uses the additional information in the dataset to permit the transfer of experience sample.
Again it is a batch-mode learning algorithm so the learning procedure is divided into two distinct parts: in the first part samples are collected, according to some policy, only from the target task. In the second part the samples in the dataset are actually used to approximate the Q-function.
One important difference with respect to the standard FQI algorithm consists in the information required: each sample must contain additional information to permit its transfer. The overall transfer procedure, in conjunction with importance sampling, is depicted in figure 4.1.
Another important property, inherited from FQI, is the use of a regression algorithm in the learning phase of the algorithm Again this simple idea permits WFQI to take advantage of the specific regression algorithm. Moreover our *weighed* approach to the problem of transfer, being it totally separate from weight calculation procedure, permits a very high flexibility in the transfer process. Ideally, we could select the weights

that perfectly minimizes the effect of the negative transfer. In general, depending on the specific problem, many others choices are possible.

A key point in our approach is the use of weights along with the samples in the datasets. These weights are defined according to idea behind *importance sampling*.
In many applications, we face the problem of estimating the mean of a random variable $\mu = \mathrm{E}[f(\boldsymbol{X})]$ where the function $f(x)$ is always 0 except for a small region R and the probability for $\boldsymbol{X}$ to assumes values in R, $\mathbb{P}(\boldsymbol{X} \in R)$, is very small.
In such situation the application of stochastic methods, like Monte Carlo, fails because it is very difficult to have even one point inside the interesting region R. Problems of this type arise very often in practice and in very different domains such as physics, bayesian inference, rare events simulations etc.
The central point is that to effectively apply Monte Carlo-like techniques some points from the interesting (important) region must somehow be taken. The idea of importance sampling is to take samples not from the original distribution of $\boldsymbol{X}$ but from an auxiliary distribution, known as the *importance distribution*, that overweights the important region, hence the name *importance* sampling.
The use of importance sampling can bring huge gains and make tractable problems where Monte Carlo is usually ineffective. On the other hand there are also known situation where the use of an importance distribution in place of the nominal one can be a disadvantage yielding estimates with extremely high variance (in some case infinite).

The first part of this chapter is dedicated to the description and motivation of WFQI, while the remaining is devoted to the problem of the weights estimation.

4.1 The WFQI Algorithm

In this section we describe WFQI and we analyze its properties and relationship with respect to FQI. Notice that, being the weight estimation problem totally separated from WFQI, in this section we describe WFQI assuming that the weights for each sample are known.
Algorithm 2 describes a pseudocode version of Weighted Fitted Q-Iteration.

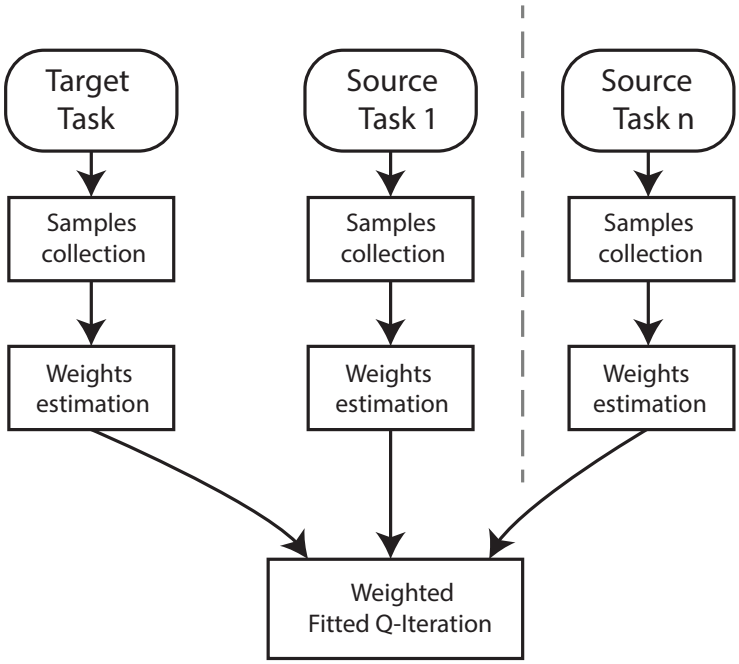

Figure 4.1: *The overall process of transfer: from the collection to the use with WFQI*

Algorithm 2 Weighted Fitted Q-Iteration algorithm

procedure WFQI($\mathcal{D} = (s, a, s', r, w_s, w_r)_{k=1}^{N}, myWeightedRegressionAlg$)
  $k \leftarrow 0$
  $\hat{Q}^0(s, a) \leftarrow 0 \; \forall (s, a) \in S \times A$
  $D' \leftarrow (s_k, a_k, r_k)_{k=1}^{N}$
  $\hat{Q}^1 \leftarrow myWeightedRegressionAlg(\mathcal{D}', w_r)$
  $\mathcal{D}' = \emptyset$
  while $checkStoppingCriteria()$ **do**
    $k \leftarrow k + 1$
    for $l = 1$ to N **do**
      $i_l = (s_l, a_l)$
      $o_l = Q^1 + \gamma \max_{a' \in A} \hat{Q}_{k-1}(s', a')$
      $\mathcal{D}'$**+=** (i_l, o_l)
    $\hat{Q}^k \leftarrow myWeightedRegressionAlg(\mathcal{D}', w_s)$

More informations about the specific implementation of WFQI are given in appendix A.

The main differences with respect to the standard FQI algorithm are:

- **Dataset**: The experience sample contained in the dataset are enhanced with additional information namely w_r and w_s. These informations are needed by WFQI in order to permit the transfer of samples coming from different source tasks. These weights, for reward (w_r) and transition model (w_p), need to be separately estimated and then injected inside the existing dataset.
- **Regression Algorithm**: The only real modification needed in algorithm 2, with respect to the standard FQI, consists in a new requirement over the used regression algorithm: in order to effective transfer samples, the regression algorithm must permit to specify, for each sample, a weight and accordingly use it in the fitting procedure. Notice that different weights are used in different part of the algorithm; at the iteration 0 the Q-function is initialized to zero this means that the initial dataset is exclusively built using the rewards of the sample thus making useless the use of w_s. Moreover, in the successive iterations, instead of building the dataset $\mathcal{D}'$ using the usual Bellman equation $o_l = r + \gamma \max_{a' \in A} \hat{Q}^{k-1}(s', a')$ we substitute r with $\hat{Q}^1$ which represents the weighted reward function (according to w_r), thus, in the regression we only need to weight the samples with w_s.

4.2 Algorithm Motivation

The motivation of using Importance Sampling in the framework of Transfer Learning comes directly from the need of a strong theoretical basis

behind the transfer procedure. Many sample-based transfer approaches, see for example [10], focus on defining measures to capture the concept of similarity between tasks and samples. In our case, the possibility to support the calculation and use of each weight with a strong theoretical background gives the possibility, not only to perform an effective transfer of samples, but also to quantify and control the amount of bias introduced in the estimation.

WFQI exploits the same idea of the standard FQI algorithm. It uses a dataset of experience sample, previously collected, in conjunction with a generic weighted regression algorithm to produce an estimation of the Q-function for the target task. The main difference, as already outlined above, is that it permits the use of samples coming from tasks different from the one it tries to solve.
If we assume that w_p, w_r are the *ideal* weights (ie. given by an oracle), then, according to the results of importance sampling presented in the previous chapter, the use of the corresponding samples coming from the set of source tasks is unbiased and may lead to a potential reduction in the variance of the estimation.
On the other side, the ideal weights can not be calculated in practice given that their knowledge would require a perfect knowledge of the reward and transition model of the target task meaning that nothing has to be learned. When using the estimated weights the use of the corresponding samples from the set of source tasks in the learning process of the target is not unbiased. This means that in addition to the potential reduction in variance an additional increase in bias is possible.
The necessity of calculating the pair of weights for each source sample motivates the need of explicitly modelling the reward and transition model for all the tasks involved in the transfer (this problem will be addressed in the final part of this chapter).

Another important characteristic of WFQI is the peculiar use of the weights associated with each samples, that permits to transfer samples even in situations where either the reward or the dynamics represented by the sample is significantly different from the target task. A formal and detailed theoretical analysis of the errors coming from the use of estimated weights is provided in the last sections of this Chapter.

In conclusion WFQI represents a very simple and intuitive modification of the original FQI algorithm. Besides, these adjustments do not negatively affect the computational complexity of the algorithm (only two additional multiplication per iteration are needed) and has the remarkable advantage of permitting the reuse of past knowledge (which in general tends to speed-up the learning process).
It also important to note some of the possible disadvantages: WFQI uses extensively the underlying idea of importance sampling and therefore

may not be effective in every situation.

4.3 Introduction to Importance Sampling

In this section, and in the remaining part of this Chapter, we introduce the general theory of Importance Sampling (sometimes abbreviated as IS), we start from a problem apparently unrelated with the problem of transfer in RL, later we describe how the transfer of samples across tasks can be reframed into the general idea of IS.

Let $\boldsymbol{X}$ be a continuous random variable distributed according to a density function $p(x)$. Our problem is to find:

$$\mu = \mathrm{E}[f(\boldsymbol{X})] = \int_{\mathcal{R}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \tag{4.1}$$

where p is a probability density function over $\mathcal{R} \subseteq \mathbb{R}^d$ and f is a generic function over $\boldsymbol{X}$. p is known as the **nominal distribution**.
As already stated we want to estimate the mean of $f(\boldsymbol{X})$ not from samples drawn from p but from a set of samples drawn from an auxiliary distribution q from now on known as the **importance distribution**. Recalling equation 4.1 we have with simple algebraic manipulation:

$$\mu = \int_{\mathcal{R}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \int_{\mathcal{R}} f(\boldsymbol{x})\frac{p(\boldsymbol{x})q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} = \mathbb{E}_q\left[f(\boldsymbol{X})\frac{p(\boldsymbol{X})}{q(\boldsymbol{X})}\right]$$

$w(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}$ is the **importance weight** for the sample $\boldsymbol{x}$. The result is very intuitive: we can estimate the mean of $f(\boldsymbol{X})$ by drawing samples from q and correcting them with a multiplicative factor equal to $\frac{p(\boldsymbol{X})}{q(\boldsymbol{X})}$.

Notice that the probability density function $q(x)$ does not to be strictly positive everywhere but it sufficient to have $q(x) > 0$ whenever $f(x)p(x) \neq 0$. It is also important to notice that $q(x)$ can never be equal to 0 (if a sample has been drawn from q then $q(x) > 0$ in any case).

We now proceed to give some additional definition and we state some results to understand when the use of importance sampling is useful and when it is not.
We define the importance sampling *estimate*, $\hat{\mu}_q$, of μ as

$$\hat{\mu}_q = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)p(X_i)}{q(X_i)} \tag{4.2}$$

The fundamental point is that we are able to compute the estimate in 4.2 only when both p and q are known. This is not always the case, in particular when we will try to apply these ideas to the problem of the

transfer of knowledge we will need to resort to some method to estimate the ration between the two densities. For this section we will assume that the ration p/q is computable.

Theorem 2 ([15])**.** *Let $\hat{\mu}_q$ the estimate provided by equation 4.2 where $\mu = \int_{\mathcal{D}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$ and $q(\boldsymbol{x}) > 0$ whenever $f(\boldsymbol{x})p(\boldsymbol{x}) \neq 0$. Then $\mathbb{E}_q[\hat{\mu}_q] = \mu$ and $Var_q[\hat{\mu}_q] = \frac{\sigma_q^2}{N}$ where*

$$\sigma_q^2 = \int_{\mathcal{D}} \frac{(f(\boldsymbol{x})p(\boldsymbol{x}))^2}{q(\boldsymbol{x})} d\boldsymbol{x} - \mu^2 = \int_{\mathcal{D}} \frac{(f(\boldsymbol{x})p(\boldsymbol{x}) - \mu q(\boldsymbol{x}))^2}{q(\boldsymbol{x})} d\boldsymbol{x} \tag{4.3}$$

Proof. Let $\mathcal{Q} = \{\boldsymbol{x} | q(\boldsymbol{x}) > 0\}$ then

$$\mathbb{E}_q\left[\frac{f(\boldsymbol{X})p(\boldsymbol{X})}{q(\boldsymbol{X})}\right] = \int_{\mathcal{Q}} \frac{f(\boldsymbol{x})p(\boldsymbol{x})}{q(\boldsymbol{x})} q(\boldsymbol{x}) d\boldsymbol{x} = \int_{\mathcal{Q}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} =$$
$$\int_{\mathcal{D}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} + \int_{\mathcal{Q}\cap\mathcal{D}^c} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} - \int_{\mathcal{D}\cap\mathcal{Q}^c} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} =$$
$$\int_{\mathcal{D}} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \mu = \mathbb{E}_q[\hat{\mu}_q]$$

The second part of the proof can be obtained in a similar way:

$$\text{Var}[\hat{\mu}_q] = \frac{1}{n}\left[\int_{\mathcal{Q}} \left(\frac{f(\boldsymbol{x})p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^2 q(\boldsymbol{x})d\boldsymbol{x} - \mu^2\right] = \frac{1}{n}\left[\int_{\mathcal{D}} \left(\frac{f(\boldsymbol{x})p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^2 q(\boldsymbol{x})d\boldsymbol{x} - \mu^2\right]$$

Simple manipulations gives equation 4.3. □

Expressions in equation 4.3 give a simple way to understand when importance sampling can succeed or fail. The best importance distribution is the one that minimizes $\int_{\mathcal{D}} \frac{(f(\boldsymbol{x})p(\boldsymbol{x}))^2}{q(\boldsymbol{x})} d\boldsymbol{x}$. Moreover if we analyze the right integral expression (always in 4.3) it is easy to notice that the numerator is small when $f(\boldsymbol{x})p(\boldsymbol{x})$ is similar to $\mu q(\boldsymbol{x})$. This happens when $f(\boldsymbol{x})p(\boldsymbol{x})$ is proportional to $q(\boldsymbol{x})$. On the other hand also the denominator can cause some problems: a region with a small q can amplify whatever lack of proportionality in the numerator.
In theory choice of q such that $\sigma_q^2 = 0$ are possible but of course of no use in practice. The choice of a good importance distribution is a complex problem that requires educated guesses, numerical search and domain-specific knowledge.

4.4 Transfer with Importance Sampling

In this section we face the problem of the transfer of knowledge across multiple tasks. We assume the context of batch-mode reinforcement learning and, for the sake of simplicity, we suppose that a single source task is available. Later on we will extend our transfer model for the case

of multiple source tasks.
We suppose some samples have been previously collected from the target task producing a dataset $\mathcal{T} = \{(s_i, a_i, s'_i, r_i)\}_{i=1}^{N_t}$ and similarly some samples (the samples that have to be transferred) have been obtained from the source task producing a dataset $\mathcal{S} = \{(s_i, a_i, s'_i, r_i)\}_{i=1}^{N_s}$. Typically we will assume $N_t << N_s$.

4.4.1 Single Source Task

Suppose to transfer a sample (s_i, a_i, s'_i, r_i) from the source task to the target task. The transfer affects the target task on two sides: the reward model and the probability transition model.
Suppose, for simplicity, to consider only the effect on the reward. This last element is modeled as a function:

$$r : S \times A \to \mathbb{R} \tag{4.4}$$

Given a state-action pair the function will return the expected reward for the agent. Every time a transfer is performed we are introducing a bias inside the estimation of the reward function for the target task.
The idea is to apply importance sampling to the samples of the source task used in the estimation of the reward function of the target task. Let be w_r the likelihood weight associated to the reward model. Select a sample pair (s, a, s', r) from the source task (we are assuming that the source and target share the same state-action space). Then we have

$$w_r = \frac{\text{likelihood of } r \text{ to be generated in } \mathcal{S}}{\text{likelihood of } r \text{ to be generate in } \mathcal{T}} \tag{4.5}$$

There are some important facts to be observed:

- When the stochastic model of the reward is perfectly known in both task the w_r can be perfectly calculated, in this setting the use of r in the target is completely unbiased. Of course if the two tasks are quite different then the reduction of variance carried by r will be much lower with respect to the case where r is directly sampled from the target.

- In practice the reward model of the source and target tasks are not known (otherwise the agent has nothing to learn, at least for the reward, and transfer will be useless). This means that we need a procedure to accurately estimates the likelihood for both tasks. A possible procedure will be discussed in the next section.

- As already observed in the general discussion about importance sampling, the denominator of w_r can never be zero (if the sample has been drawn from the source then its likelihood is by definition

different from 0). Some numerical problems may arise when the denominator is close to zero, in such cases w_r may be very large and this tends to increase the variance of the estimation.

Identical considerations hold for the probability transition model, also in this case we have a stochastic model that given a state-action pair returns a probability distribution over the next state.
Again the idea is to apply importance sampling also to the transition model, this means that a second likelihood weight w_p must be defined. Fix a sample (s, a, s', r) to be transferred then:

$$w_p = \frac{\text{likelihood of } s \to s' \text{ to be generated in } \mathcal{S}}{\text{likelihood of } s \to s' \text{ to be generate in } \mathcal{T}} \tag{4.6}$$

The overall transfer procedure for the single source case requires to collect the two datasets $\mathcal{T}$ and $\mathcal{S}$ from the two tasks. A weight estimation procedure is applied to the datasets. The output is a new pair of datasets $\mathcal{S}'$ and $\mathcal{T}'$ where

$$\begin{cases} \mathcal{S}' = \{(s_i, a_i, s'_i, r_i, w_r^i, w_p^i)\}_{i=1}^{N_s} \\ \mathcal{T}' = \{(s_i, a_i, s'_i, r_i, 1, 1)\}_{i=1}^{N_t} \end{cases}$$

Notice that the weights for the samples coming from the target tasks (that do not need to be transferred) are obviously fixed to 1.

4.4.2 Multiple Source Tasks

The extension to the case where multiple source tasks are present is straightforward. Let $\mathcal{T}$ the dataset of the samples collected from the target task and let $\mathcal{S}_i$ the dataset of samples collected from the i-th source task.
Then the transfer procedure works exactly as in the single source case: the source datasets are considered one at a time, a weight estimation procedure is applied to the samples contained in the current dataset so that we obtain

$$\mathcal{S}_i = \{(s_i^j, a_i^j, s_i^{'j}, r_i^j, w_r^{i,j}, w_p^{i,j})\}_{j=1}^{N_s}$$

This is repeated for every source tasks. As in the previous case for the target task the weights for reward and transition model are fixed to 1 (ie. nothing to transfer).

Two remaining open problems are:

1. How to estimates the likelihood weights for transition and reward model when the relevant densities are not known? This problem will be addressed in the next two sections.
2. How to effectively use the likelihood weights inside a learning algorithm? The conclusive sections of this chapter will propose a variation of the well-known FQI algorithm to include the information coming from the likelihood weights.

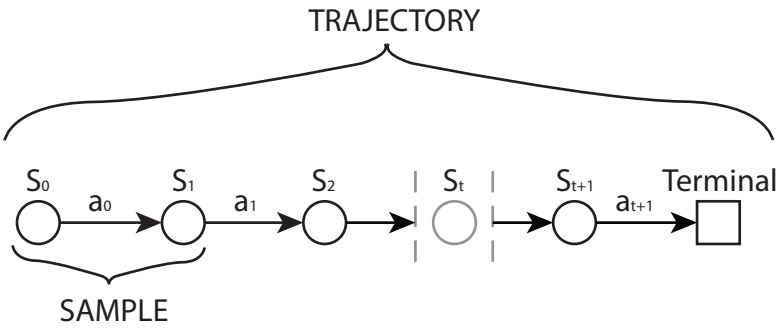

Figure 4.2: *Trajectory vs Sample*

4.4.3 Transfer of Samples vs Transfer of Trajectories

One implicit assumption done so far is the type of knowledge to be transferred to the target task. In the last years two main approaches have been proposed: the transfer of *trajectories* or the transfer of *samples*. Figure 4.2 depicts the difference between the two entities.
A trajectory is characterized by an initial state, a terminal state (where the trajectory ends because either one of the goal state or a maximum horizon has been reached by the agent) and by a series of intermediate states that link the initial with the terminal state. Each state is linked with each other with an action that performs the transition between the two states. On the other hand, a sample is characterized by four-tuples (s, a, s', r) where s is the state in which the agent is performing action a reaching the successive state s' and obtaining reward r.

The transfer of a sample is, in our specific transfer learning framework, more advantageous than the transfer of an entire trajectory. Indeed the transfer of an entire trajectory does not permit to decide which part of the trajectory is actually beneficial for the transfer and which are not. There may be part of the trajectory that could represents dynamics that may negatively bias the agent inside the target task.
On the other hand, the transfer of samples permits a finer control over the transfer process. Once the samples are collected from the source tasks we loose the information about the specific trajectory each sample belongs to. Our approach permits to estimate a pair of weights for each sample, these weights are used by the learning algorithm to correct the bias introduced by such samples.

As an example consider the problem of the transfer of a trajectory where 99 % of the transition/rewards are identical as in the target task but 1 % of the trajectory is completely different with respect the target the target. In this case, the overall likelihood of the trajectory will be approximately zero leading to an ineffective transfer. In the case we had considered each single sample of the trajectory a much more effective transfer could

have been accomplished.

4.4.4 Weights Selection

One key point in our transfer procedure is the calculation of the likelihood weights that are used by the learning to correct the negative bias. To each sample we associate a pair of weights w_r and w_p. Given a sample (s, a, s', r) the weight w_r is defined as the ratio between the likelihood of reward r to be generated in the target task and the likelihood of r to be generated in the corresponding source task (from which the sample has been collected). On the other side w_p is the weight associated with the transition model and it is defined as the ratio between the likelihood for the transition $s \to s'$ to be generated in the target task and the likelihood, for the same transition, to generated in the correspondent source task.

Ideal case

In the case where the probabilistic models for reward and transition are perfectly known it is possible to calculate the true values for the importance weights of each sample in the dataset. We assume that both reward and transition are normally distributed:

$$x_S \sim \mathcal{N}(\mu_S, \sigma^2), \quad x_T \sim \mathcal{N}(\mu_T, \sigma^2) \tag{4.7}$$

where x is either the reward r or the next state s' for a sample (s, a, x). In this case the corresponding weight can be obtained applying the definition of importance weight:

$$w(x) = exp\left(-\frac{(x-\mu_T)^2}{2\sigma^2}\right) exp\left(\frac{(x-\mu_S)^2}{2\sigma^2}\right) \tag{4.8}$$

Notice that we assume the variance of the reward/transition model to be identical in both the source and target task. This assumption is without loss of generality, indeed in the case of different variances we just need to consider an additional constant term, $\frac{\sigma_S^2}{\sigma_T^2}$, in Equation 4.8. This assumption will be maintained in the next chapters and sections.

Non-Ideal case

As already observed in the previous chapters it is not possible, in practice, to perfectly calculate the pair of weights for each sample in the dataset. This is due to the fact that the model for the reward and transitions are not known in both the source and target task. In the case they were known then we are no more in the settings of reinforcement learning but we are solving a dynamic programming problem where the model of the environment is perfectly known to the agent and the objective is to approximate the optimal policy. In the following, we propose a procedure

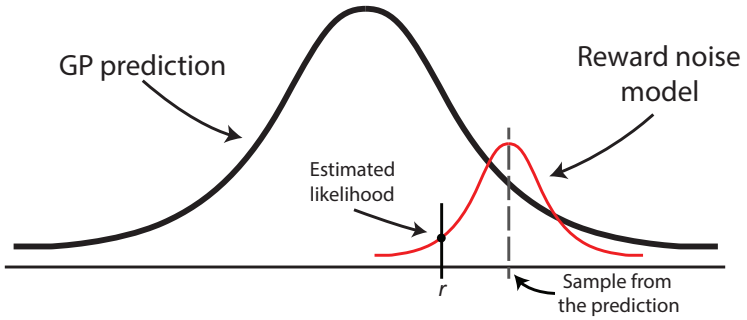

Figure 4.3: *Estimating the reward/transition weight from a GP prediction.*

to estimate the weights based on Gaussian Process (GP) regression.
The main idea behind the estimation procedure is to make a hypothesis over the reward and transition model. Suppose for now to consider only the estimation of the weight associated with the reward, w_r. We hypothesize a gaussian model for the reward model for both the source and target task. The idea is to place two distinct GP regressors, one for each task, over the entire state-action space. For an extensive discussion about GPs refer to appendix B.
Once the gaussian process has been fitted using the samples in the dataset for the source and target task, we fix a sample (s, a, s', r) and we ask a prediction, for the pair (s, a), to the GP associated with the source and to the GP associated with the target.
Now, we proceed to give an extensive description of the estimating procedures.
A *first* idea to obtain an estimation $\tilde{w}_x$ is to directly sample the weights distribution, which is unknown and not available in practice.
Suppose to have a source sample (s, a, x), where x is either r or s', and we want to compute its importance weight. First, we predict the distribution if the mean of x under the target model and the source model using our previously fitted Gaussian Processes:

$$\tilde{\mu}_T \sim \mathcal{N}(\mu_{GP,T}, \sigma^2_{GP,T}), \quad \tilde{\mu}_S \sim \mathcal{N}(\mu_{GP,S}, \sigma^2_{GP,S}) \tag{4.9}$$

Then we consider the estimated importance weights:

$$\tilde{w}(x) = \tilde{w}_x(\tilde{\mu}_T, \tilde{\mu}_S) = exp\left(-\frac{(x-\tilde{\mu}_T)^2}{2\sigma^2}\right) exp\left(\frac{(x-\tilde{\mu}_S)^2}{2\sigma^2}\right) \tag{4.10}$$

where the notation $\tilde{w}_x(\tilde{\mu}_T)$ is used to emphasize the dependence on the two means (which now are random variables) and on the fixed value of x.

We sample N_{smp} from both $\tilde{\mu}_T$ and $\tilde{\mu}_S$ then for the i-th sample μ_x^i we consider the two distributions:

$$n_x^i \sim \mathcal{N}(\mu_x^i, \sigma^2_{GP,T}), \quad d_x^i \sim \mathcal{N}(\mu_x^i, \sigma^2_{GP,S}) \tag{4.11}$$

Then the estimation according to the i-th sample is given by:

$$w^i(x) = \frac{n^i(x)}{d^i(x)} \tag{4.12}$$

Notice that when the prediction from the GP of source and target task are perfect, ie. $\sigma^2_{GP,T} = \sigma^2_{GP,S} = 0$, $\tilde{w}(x)$ converges to $w(x)$. The procedure is graphically summarized in Figure 4.3.
Despite being a very simple approach some problems may arise when choosing the final weight among the N_{smp} estimations. The selection of different quantiles of the weight distribution may lead, in practice, to very different performance (good in some cases, poor in others). Furthermore we do not have any theoretical backup to support the choice of a particular sample among the N_{smp}.
We are able to prove the following result on $\tilde{w}_x$:

Lemma 1. *The expectation of $\tilde{w}_x(\tilde{\mu}_T, \tilde{\mu}_S)$ under the distributions defined in Equation 4.9 is:*

$$\mathbb{E}[\tilde{w}_x(\tilde{\mu}_T, \tilde{\mu}_S)] = \begin{cases} \frac{\sigma^2}{\sigma^2 - \sigma^2_{GP,S}} \frac{\mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma^2_{GP,T})}{\mathcal{N}(x; \mu_{GP,T}, \sigma^2 - \sigma^2_{GP,S})} & \sigma^2_{GP,S} < \sigma^2 \\ \infty \quad Otherwise \end{cases} \tag{4.13}$$

Proof. Suppose that $\sigma^2_{GP,S} < \sigma^2$. Then:

$$\begin{aligned} \mathbb{E}[\tilde{w}_x(\tilde{\mu}_T, \tilde{\mu}_S)] &= \iint \frac{exp\left(-\frac{(x-\tilde{\mu}_T)^2}{2\sigma^2}\right)}{\sqrt{2\pi\sigma^2}} \frac{\sqrt{2\pi\sigma^2}}{exp\left(-\frac{(x-\tilde{\mu}_S)^2}{2\sigma^2}\right)} \\ &\frac{exp\left(-\frac{(\tilde{\mu}_T-\mu_{GP,T})^2}{2\sigma^2_{GP,T}}\right)}{\sqrt{2\pi\sigma^2_{GP,T}}} \frac{exp\left(-\frac{(\tilde{\mu}_S-\mu_{GP,S})^2}{2\sigma^2_{GP,S}}\right)}{\sqrt{2\pi\sigma^2_{GP,S}}} d\tilde{\mu}_T\tilde{\mu}_S \\ &= \int \frac{exp\left(-\frac{(x-\tilde{\mu}_T)^2}{2\sigma^2}\right)}{\sqrt{2\pi\sigma^2}} \frac{exp\left(-\frac{(\tilde{\mu}_T-\mu_{GP,T})^2}{2\sigma^2_{GP,T}}\right)}{\sqrt{2\pi\sigma^2_{GP,T}}} d\tilde{\mu}_T \\ &\int \frac{\sqrt{2\pi\sigma^2}}{exp\left(-\frac{(x-\tilde{\mu}_S)^2}{2\sigma^2}\right)} \frac{exp\left(-\frac{(\tilde{\mu}_S-\mu_{GP,S})^2}{2\sigma^2_{GP,S}}\right)}{\sqrt{2\pi\sigma^2_{GP,S}}} d\tilde{\mu}_S \end{aligned} \tag{4.14}$$

The first integrand is over the product of two Gaussian densities, which is known to be (see [3]):

$$\mathcal{N}(\tilde{\mu}_T; \bar{\mu}_T, \bar{\sigma}^2_T)\mathcal{N}(x; \mu_{GP,T}, \sigma^2 + sigma^2_{GP,T}) \tag{4.15}$$

where the values of the mean $\bar{\mu_T}$ and variance $\bar{\sigma}_T^2$ of the first density are not important to complete the proof since such density integrates out in the previous equation. By adopting a procedure as the one described in [3], we can write the ratio of the two Gaussians densities in the second integral as:

$$\frac{\sigma^2}{\sigma^2 - \sigma_{GP,S}^2} \frac{\mathcal{N}(\tilde{\mu}_S; \bar{\mu}_S, \bar{\sigma}_S^2)}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 - \sigma_{GP,S}^2)} \tag{4.16}$$

Proof. It can be easily verified by taking:

$$\bar{\mu}_S = \frac{\tilde{\mu}_S - \tilde{\mu}_{GP,S}}{\sigma^2 - \sigma_{GP,S}^2}, \quad \bar{\sigma}^2 = \frac{\sigma^2 \sigma_{GP,S}^2}{\sigma^2 - \sigma_{GP,S}^2} \tag{4.17}$$

Then by substituting and writing explicitly the Gaussian densities we have:

$$\frac{\sigma^2}{\sigma^2 - \sigma^2} \frac{\sqrt{2\pi}}{\sqrt{2\pi}} \frac{\sqrt{\sigma^2 - \sigma_{GP,S}^2}}{\sqrt{\frac{\sigma^2 \sigma_{GP,S}^2}{\sigma^2 - \sigma_{GP,S}^2}}} \exp\left(\frac{(x - \mu_{GP,S})^2}{2(\sigma^2 - \sigma_{GP,S}^2)}\right) \exp\left(-\frac{(\tilde{\mu}_S - \bar{\mu}_S)^2}{2\frac{\sigma^2 \sigma_{GP,S}^2}{\sigma^2 - \sigma_{GP,S}^2}}\right) =$$

$$\frac{\sigma}{\sigma_{GP,S}} \exp\left(\frac{\sigma_{GP,S}^2 x^2 + \sigma_{GP,S}^2 \tilde{\mu}_S^2 - 2\sigma_{GP,S}^2 x \tilde{\mu}_S^2 - \sigma^2 \tilde{\mu}_S^2 - \sigma^2 \mu_{GP,S}^2 + 2\sigma^2 \tilde{\mu}_S \mu_{GP,S}}{2\sigma^2 \sigma_{GP,S}^2}\right) =$$

$$\frac{\sqrt{2\pi\sigma^2}}{exp\left(-\frac{(x - \tilde{\mu}_S)^2}{2\sigma^2}\right)} \frac{exp\left(-\frac{(\tilde{\mu}_S - \mu_{GP,S})^2}{2\sigma_{GP,S}^2}\right)}{\sqrt{2\pi\sigma_{GP,S}^2}}$$

□

where, again, the values of $\bar{\mu}_S$ and $\bar{\sigma}_S^2$ are not relevant to our proof. Thus:

$$\begin{aligned}
\mathbb{E}[\tilde{w}_x(\tilde{\mu}_T, \tilde{\mu}_S)] &= \int \mathcal{N}(\tilde{\mu}_T; \bar{\mu}_T, \bar{\sigma}_T^2) \mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma_{GP,T}^2) d\tilde{\mu}_T \\
&\int \frac{\sigma^2}{\sigma^2 - \sigma_{GP,S}^2} \frac{\mathcal{N}(\tilde{\mu}_S; \bar{\mu}_S, \bar{\sigma}_S^2)}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 - \sigma_{GP,S}^2)} d\tilde{\mu}_S \\
&= \mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma_{GP,T}^2) \int \mathcal{N}(\tilde{\mu}; \bar{\mu}_T, \bar{\sigma}_T^2) d\tilde{\mu}_T \\
&\frac{\sigma^2}{\sigma^2 - \sigma_{GP,S}^2} \frac{1}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 - \sigma_{GP,S}^2)} \int \mathcal{N}(\tilde{\mu}_S; \bar{\mu}_S, \bar{\sigma}_S^2) d\tilde{\mu}_S \\
&= \frac{\sigma^2}{\sigma^2 - \sigma_{GP,S}^2} \frac{\mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma_{GP,T}^2)}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 - \sigma_{GP,S}^2)}
\end{aligned} \tag{4.18}$$

To prove that the expectation diverges when $\sigma_{GP,S}^2 \geq \sigma^2$, simply notice that:

$$\bar{\sigma}_S^2 = \frac{\sigma^2 \sigma_{GP,S}^2}{\sigma^2 - \sigma_{GP,S}^2} \tag{4.19}$$

which clearly makes $\int \mathcal{N}(\tilde{\mu}; \bar{\mu}_S, \bar{\sigma}_S^2) d\tilde{\mu}_S$ diverge when $\sigma_{GP,S}^2 \geq \sigma^2$. □

An interesting result is that the expected weights coincide with the true ones when the Gaussian Processes prediction is perfect (ie. $\mu_{GP,T} = \mu_T, \mu_{GP,S} = \mu_S$, and $\sigma^2_{GP,T} = \sigma^2_{GP,S} = 0$). However, the fact that such expectation could very easily diverge makes weight selection rather complicated. In practice, we would like to have estimated weights similar to the true ones when the GPs are accurate, but we would also like to limit the effect of inaccurate prediction from the GPs. An estimator combining these two characteristics is :

$$\tilde{w}(x) = \frac{\mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma^2_{GP,T})}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 + \sigma^2_{GP,S})} \tag{4.20}$$

which still converges to the true weights when the Gaussian Process is perfect, but does not have any problem when $\sigma^2_{GP,S} \geq \sigma^2$ and has a smaller variance. Intuitively, $\mu_{GP,T}$ and $\mu_{GP,S}$ are our best guesses for the true means, but we want also to take into account the uncertainty in the Gaussian Process prediction. This is done by summing the variance of our models and those of the GPs. The resulting weights are very precise when the GP are precise, while the effect of bad predictions is limited by the fact that the two densities have larger standard deviation.

To furtherly justify Equation 4.20 we also provide, in Figure 4.4, a comparative plot of equations 4.20 and 4.13. It is clear that when $\sigma^2_{GP,S}$ is sufficiently small (ie. many source samples are available) than the two equations lead to the same weight estimation. On the other hand, when the prediction of the GP has a high variance Equation 4.13 provides a weight value that increases as $\sigma^2_{GP,S}$ approximate the model variance σ^2. This, in practice, leads to estimate very high weights and to a poor performance (as shown in the experiments). Moreover in the unfortunate situation in which $\sigma^2_{GP,S} > \sigma^2$ the formula does not permit the estimation of the weight (in practice the sample is discarded).
Despite these disadvantages we can still make good use of Equation 4.13. The asymptote in Figure 4.4 occurs when $\sigma^2_{GP,S} = \sigma^2$, as a consequence a possible idea to obtain a more accurate estimation is to overestimate σ^2 shifting the asymptote to the right side of the graph, this modification produces a much lower number of discarded samples and, at the same time, a curve much closer to the ideal one.
On the other hand, Equation 4.20, referring to Figure 4.4, is always able to provide an estimation for $\hat{w}$, and when the prediction of the GP has a high variance the estimated weight is much lower with respect the previous case. In general, we observe that Equation 4.20 leads to estimations much closer to the ideal one and better results on the benchmarks.

In the experiments chapter we compare the performance of the use of Formula 4.13 without σ^2 overestimation, Formula 4.13 with σ^2 overestimation and finally Formula 4.20. Refer to Chapter 6 for more detail

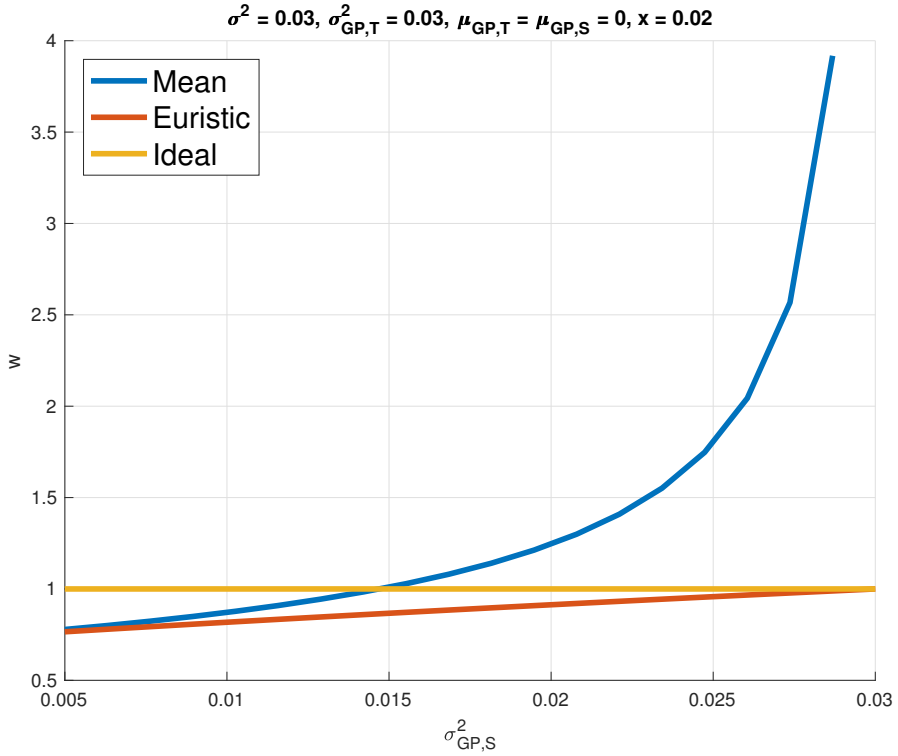

Figure 4.4: *Weight estimations according to Equation 4.13 (red), 4.20 (blue) and the ideal weight (orange)*

about the experimental results.

A pseudocode implementation of the procedures to estimate weights for rewards and dynamics, in conjunction with use 4.20, are given by Algorithm 3 and 4.

Algorithm 3 w_r estimation algorithm

procedure ESTIMATERWWEIGHT((s, a, s', r), GP_r, x)
  $W \leftarrow \emptyset$
  $\mathcal{N}_S = (\mu_{GP,S}, \sigma^2_{GP,S}) \leftarrow GP^S_r.predict(s, a)$
  $\mathcal{N}_T = (\mu_{GP,T}, \sigma^2_{GP,T}) \leftarrow GP^T_r.predict(s, a)$
  for $i = 0$ to N_{samp} **do**
    $d_s \leftarrow \mathcal{N}(r; \mu_{GPs}, \sigma^2_r + \sigma^2_{GPs})$
    $d_t \leftarrow \mathcal{N}(r; \mu_{GPt}, \sigma^2_r + \sigma^2_{GPt})$
    $w_r \leftarrow \frac{d_t}{d_s}$
    $W += w_r$

It is also important to notice that the procedure described above is just a possible way to deal with the problem of weight estimation. As we will see in the next chapter the learning algorithm (Weighted Fitted Q-Iteration) is totally independent of the method used to calculate the weights as-

Algorithm 4 w_p estimation algorithm

procedure ESTIMATETRWEIGHT((s, a, s', r), GP_t)
  $W \leftarrow \emptyset$
  $\mathcal{N}_S = (\mu_{GP,S}, \sigma^2_{GP,S}) \leftarrow GP^S_p.predict(s, a)$
  $\mathcal{N}_T = (\mu_{GP,T}, \sigma^2_{GP,T}) \leftarrow GP^T_p.predict(s, a)$
  for $i = 0$ to N_{samp} **do**
    $d_s \leftarrow \mathcal{N}(s'; \mu_{GPs}, \sigma^2_r + \sigma^2_{GPs})$
    $d_t \leftarrow \mathcal{N}(s'; \mu_{GPt}, \sigma^2_r + \sigma^2_{GPt})$
    $w_p \leftarrow \frac{d_t}{d_s}$
    $W+ = w_p$

sociated with the sample. This, in our opinion, represents a further advantage with respect other transfer learning related algorithms.

Chapter 5

Theoretical Analysis

In this chapter we try to provide some theoretical basis regarding our approach to the problem of transfer. In particular we first try to understand if it is possible to provide some theoretical bounds to the error introduced by both the use of source samples and the estimation of the weights of rewards and dynamics. We are able to show that, under very simple hypothesis, over the tasks it possible to bound the bias introduced by the use of source samples in conjunction with estimated weights.

5.1 Errors Bounding

5.1.1 Assumptions

Suppose $P^S(s, a, s', r) = P^S(r|s, a)P^S(s'|s, a)P^S(s, a)$ is the joint density of the samples under the source model, and P^T the corresponding density under the target model. Let indicate with x either the reward r or s' for a given sample (s, a, s', r). Suppose, without loss of generality, that only one source MDP exists, indeed, in the case multiple source tasks are available no substantial modifications are needed given that no relationship are assumed to exists between tasks. Any regressor we could adopt in our Weighted Fitted Q-Iteration algorithm tries to minimize the expectation of some loss function L under the target distribution:

$$arg\min_{\theta} \mathbb{E}_{P^T}\left[L(\hat{Q}^1_\theta(s, a) + \max_{a'} \hat{Q}_{t-1}(s', a'))\right] \tag{5.1}$$

This can be computed under the source distribution by employing importance sampling:

$$arg\min_{\theta} \mathbb{E}_{PS}\left[w_p(s,a,s')L(\hat{Q}^1_\theta(s,a) + \max_{a'}\hat{Q}_{t-1}(s',a'))\right] \tag{5.2}$$

where:

$$w_p(s,a,s') = \frac{P^T(s'|s,a)P^T(s,a)}{P^S(s'|s,a)P^S(s,a)} \tag{5.3}$$

According to our approach, we neglect the terms $P(s,a)$ and, thus, consider the importance weights defined by:

$$w_p(s,a,s') = \frac{P^T(s'|s,a)}{P^S(s'|s,a)} \tag{5.4}$$

Notice that this approximation does not introduce any error in the estimation of the Q-function since, for every pair (s,a), the target $\hat{Q}^1 + \max_{a'}\hat{Q}_{t-1}(s',a')$ is always correctly weighted.
We consider Gaussian models for both the reward and the transitions dynamics, thus:

$$P^T(r|s,a) = \mathcal{N}(\mu_{r,T},\sigma_r^2), \quad P^S(r|s,a) = \mathcal{N}(\mu_{r,S},\sigma_r^2) \tag{5.5}$$

and:

$$P^T(s'|s,a) = \mathcal{N}(\mu_{p,T},\sigma_p^2), \quad P^S(s'|s,a) = \mathcal{N}(\mu_{p,S},\sigma_p^2) \tag{5.6}$$

We suppose, without loss of generality, the variances are known and do not change between source and target. Let us denote the densities we estimate from the available samples as:

$$\tilde{P}^T(r|s,a) = \mathcal{N}(\tilde{\mu}_{r,T},\sigma_r^2), \quad \tilde{P}^S(r|s,a) = \mathcal{N}(\tilde{\mu}_{r,S},\sigma_r^2) \tag{5.7}$$

and:

$$\tilde{P}^T(s'|s,a) = \mathcal{N}(\tilde{\mu}_{p,T},\sigma_p^2), \quad \tilde{P}^S(s'|s,a) = \mathcal{N}(\tilde{\mu}_{p,S},\sigma_p^2) \tag{5.8}$$

Suppose to have N target samples and M source samples (typically $N << M$). Then, the empirical expected loss considered by our regressor is:

$$\hat{L} = \frac{1}{N+M}\sum_{i=1}^{N} L(r_i,s'_i) + \frac{1}{N+M}\sum_{j=1}^{M}\tilde{w}_p(s'_j)L(r_j,s'_j) \tag{5.9}$$

With $\tilde{w}$ we indicated the estimated weights. Notice that we keep the dependency on (s,a) implicit for ease of notation.

5.1.2 Analysis

In this section we assume that the state space and the reward function are bounded, anyway the analysis is possible also in the unbounded case but less powerful results.

$$\forall s \in \mathcal{S} : s \in [-S,S], \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A} : \mathcal{R}(s,a) \in [-R,R]$$

Under these hypothesis we can prove the following result:

Lemma 2. *The true importance weights are bounded:*

$$w(x) \leq W < \infty \tag{5.10}$$

Proof. The importance weights can be written as:

$$w(x) = \frac{P^T(x)}{P^S(x)} \tag{5.11}$$

Let us bound $w(x)$ using the Gaussian hypothesis:

$$\begin{aligned}
w(x) &= exp\left(\frac{2x(\mu_{x,T} - \mu_{x,S}) - (\mu_{x,T}^2 - \mu_{x,S}^2)}{2\sigma_x^2}\right) \\
&= \frac{exp\left(\frac{2x(\mu_{x,T}-\mu_{x,S})}{2\sigma_x^2}\right)}{exp\left(\frac{(\mu_{x,T}^2-\mu_{x,S}^2)}{2\sigma_x^2}\right)} \\
&\leq \frac{1}{exp\left(\frac{(\mu_{x,T}^2-\mu_{x,S}^2)}{2\sigma_x^2}\right)} \max\left\{exp\left(\frac{2X(\mu_{x,T} - \mu_{x,S})}{2\sigma_x^2}\right), exp\left(-\frac{2X(\mu_{x,T} - \mu_{x,S})}{2\sigma_x^2}\right)\right\} \\
&= \frac{exp\left(\frac{X|\mu_{x,T}-\mu_{x,S}|}{\sigma_x^2}\right)}{exp\left(\frac{(\mu_{x,T}^2-\mu_{x,S}^2)}{2\sigma_x^2}\right)} \\
&= W \\
&< \infty
\end{aligned} \tag{5.12}$$

Where x can indicate either the reward r or the next state s' of a given sample. □

Notice that, in practice, any estimated weight larger than W can be safely omitted or clipped to zero since it would correspond to a wrong density estimation, thus justifying our assumption.

In the following we proceed to give a bound over $|\hat{L} - \mu_L|$. We initially assume that the reward models of target and source are identical and no generalization error is introduced by the regressor over $\hat{Q}^1$. In the second part we also add the error introduced by the regression, and use of estimated weights, over the reward.
Again we consider the empirical loss function:

$$\hat{L} = \hat{L}_T + \hat{L}_S = \frac{1}{N+M}\sum_{i=1}^{N} L(r_i, s_i') + \frac{1}{N+M}\sum_{j=1}^{M} \tilde{w}_p(s_j')L(r_j, s_j') \tag{5.13}$$

where $L \in [0, 1]$. The first term refers to the unbiased samples coming the target task, that, by definition have unitary weights. The second term refers to the biased samples coming from the source task and must be

corrected using the corresponding importance weights.

Our goal is to find a bound on $|\hat{L} - \mu_L|$. This can be easily obtained by simple algebraic manipulations of the previous equation:

$$\begin{aligned}
&\left|\hat{L} - \mu_L\right| = \left|\hat{L}_T - \frac{N}{M+M}\mu_L + \hat{L}_S - \frac{M}{N+M}\mu_L\right| \\
&\leq \left|\hat{L}_T - \frac{N}{N+M}\mu_L\right| + \left|\hat{L}_S - \frac{M}{N+M}\mu_L\right| \\
&\leq \left|\mathbb{E}_{P^T}[\hat{L}_T] - \frac{N}{N+M}\mu_L\right| + \left|\hat{L}_T - \mathbb{E}_{P^T}[\hat{L}_T]\right| \\
&+ \left|\mathbb{E}_{P^S}[\hat{L}_S] - \frac{M}{N+M}\mu_L\right| + \left|\hat{L}_S - \mathbb{E}_{P^S}[\hat{L}_S]\right| \\
&= \left|\hat{L}_T - \mathbb{E}P^T[\hat{L}_T]\right| + \left|\mathbb{E}_{P^S}[\hat{L}_S] - \frac{M}{M+N}\mu_L\right| + \left|\hat{L}_S - \mathbb{E}_{P^S}[\hat{L}_S]\right|
\end{aligned} \tag{5.14}$$

where the last equality holds since the first term is zero (the estimator $\hat{L}_T$) is unbiased). We now proceed to bound the three terms separately:

Lemma 3. *Given $\delta > 0$, with probability at least $1-\delta$, the term $\left|\hat{L}_T - \mathbb{E}_{P^T}[\hat{L}_T]\right|$ is bounded by:*

$$\left|\hat{L}_T - \mathbb{E}_{P^T}[\hat{L}_T]\right| \leq \frac{2\log(2/\delta)}{3(N+M)} + \sqrt{\frac{N\log(2/\delta)}{2(N+M)^2}} \tag{5.15}$$

Proof. From Bernstein's inequality, we know that:

$$P\left(\left|\sum_{i=1}^{N} L_i - N\mathbb{E}_{P^T}[L]\right| > t\right) < 2exp\left(-\frac{t^2/2}{NVar[L] + Ct/3}\right) \tag{5.16}$$

By setting $t = (N+M)\epsilon$, we obtain:

$$P\left(\left|\hat{L} - \mathbb{E}[\hat{L}]\right| > \epsilon\right) < 2exp\left(-\frac{(N+M)^2\epsilon^2/2}{NVar[L] + (N+M)C\epsilon/3}\right) \tag{5.17}$$

Notice that C, ie. the range of L, is 1, and, from Popoviciu's inequality, $Var[L] \leq 1/4$. By setting the right-hand side equal to δ and solving for ϵ, we obtain the desired result. □

Lemma 4. *Given $\delta > 0$, with probability at least $1-\delta$, the term $\left|\hat{L}_S - \mathbb{E}_{P^S}[\hat{L}_S]\right|$ is bounded by:*

$$\left|\hat{L}_S - \mathbb{E}_{P^S}[\hat{L}_S]\right| \leq \frac{2W\log(2/\delta)}{3(N+M)} + \sqrt{\frac{2MW^2\log(2/\delta)}{(N+M)^2}} \tag{5.18}$$

Proof. The proof follows straightforwardly from the one of Lemma 3. Simply notice that now the range of $\tilde{w}_p(s')L(r,s')$ is $C = W$ and a trivial bound on its variance is $Var[\tilde{w}_p(s')L(r,s')] \leq W^2$. □

Lemma 5. *The term* $\left|\mathbb{E}_{P^S}[\hat{L}_S] - \frac{M}{N+M}\mu_L\right|$ *is bounded by:*

$$\left|\mathbb{E}_{P^S}[\hat{L}_S] - \frac{M}{N+M}\mu_L\right| \leq \frac{M}{N+M} \tag{5.19}$$

Proof. Notice that we can write:

$$\begin{aligned}\left|\mathbb{E}[\hat{L}_S] - \frac{M}{N+M}\mu_L\right| &= \left|\frac{M}{N+M}\mathbb{E}[\tilde{w}_p(s')L(s')] - \frac{M}{M+N}\mu_L\right| \\ &= \frac{M}{N+M}|\mathbb{E}_{P^S}[\tilde{w}_p(s')L(s') - w_p(s')L(s')]|\end{aligned} \tag{5.20}$$

The latter term can be bound with:

$$\begin{aligned}&|\mathbb{E}_{P^S}[\tilde{w}_p(s')L(s') - w_p(s')L(s')]| \\ &\quad= \left|\mathbb{E}_{P^S}\left[\frac{\tilde{P}^T(s')}{\tilde{P}^S(s')}L(s')\right] - \mathbb{E}_{P^S}\left[\frac{P^T(s')}{P^S(s')}L(s')\right]\right| \\ &\quad\leq \left|\mathbb{E}_{P^S}\left[\frac{\tilde{P}^T(s')}{\tilde{P}^S(s')}L(s')\right] - \mathbb{E}_{P^S}\left[\frac{\tilde{P}^T(s')}{P^S(s')}L(s')\right]\right| \\ &\quad+ \left|\mathbb{E}_{P^S}\left[\frac{\tilde{P}^T(s')}{P^S(s')}L(s')\right] - \mathbb{E}_{P^S}\left[\frac{P^T(s')}{P^S(s')}L(s')\right]\right| \\ &\quad= \left|\mathbb{E}_{P^S}\left[\frac{\tilde{P}^T(s')}{\tilde{P}^S(s')P^S(s')}\left(P^S(s')L(s') - \tilde{P}^S(s')L(s')\right)\right]\right| \\ &\quad+ \left|\mathbb{E}_{P^S}\left[\frac{1}{P^S(s')}\left(\tilde{P}^T(s')L(s') - P^T(s')L(s')\right)\right]\right| \\ &\quad= \left|\int \tilde{w}_p(s')\left(P^S(s')L(s') - \tilde{P}^S(s')L(s')\right)ds'\right| \\ &\quad+ \left|\int \left(P^T(s')L(s') - \tilde{P}^T(s')L(s')\right)ds'\right| \\ &\quad\leq 2W\mathcal{TV}(P^S||\tilde{P}^S) + 2\mathcal{TV}(P^T||\tilde{P}^T)\end{aligned} \tag{5.21}$$

where $\mathcal{TV}$ denotes the *total variation distance* between two probability measures. Remember that the total variation can be written as:

$$\mathcal{TV}(p||q) = \frac{1}{2}\sup_{f:|f|<1}\left|\int (f(x)p(x) - f(x)q(x))dx\right| \tag{5.22}$$

This last equation can be further expanded by considering Pinsker's inequality:

$$\begin{aligned} & 2W\mathcal{TV}(P^S||\tilde{P}^S) + 2\mathcal{TV}(P^T||\tilde{P}^T) \\ & \leq 2W\sqrt{\frac{1}{2}\mathcal{KL}(P^S||\tilde{P}^S)} + 2\sqrt{\frac{1}{2}\mathcal{KL}(P^T||\tilde{P}^T)} \\ & \leq 2W\sqrt{\frac{|\mu_{p,S} - \tilde{\mu}_{p,S}|^2}{4\sigma_p^2}} + 2\sqrt{\frac{|\mu_{p,T} - \tilde{\mu}_{p,T}|^2}{4\sigma_p^2}} \\ & \leq W\frac{|\mu_{p,S} - \tilde{\mu}_{p,S}|}{\sigma_p} + \frac{|\mu_{p,S} - \tilde{\mu}_{p,S}|}{\sigma_p} \end{aligned} \tag{5.23}$$

where $\mathcal{KL}$ is the Kullback-Leibler divergence. By combining the latter bound with Equation 5.20 we obtain the desired result. □

Finally, by combining the previous result we obtain the final result:

Theorem 3. *Given $\delta > 0$, with probability at least $1 - 2\delta$:*

$$\begin{aligned} \left|\hat{L} - \mu_L\right| \leq & \frac{2\log(2/\delta)}{3(N+M)} + \sqrt{\frac{N\log(2/\delta)}{2(N+M)^2}} + \frac{2W\log(2/\delta)}{3(N+M)} + \sqrt{\frac{2MW^2\log(2/\delta)}{(N+M)^2}} \\ & + \frac{M}{N+M}\left(W\frac{|\mu_{p,S} - \tilde{\mu}_{p,S}|}{\sigma_p} + \frac{|\mu_{p,S} - \tilde{\mu}_{p,S}|}{\sigma_p}\right) \end{aligned} \tag{5.24}$$

Proof. The proof is an immediate conseguence of Lemma 3, 4 and 5. □

The last bound is very loose but provides some useful insights. When the predictions of the GPs are perfect and $M, N \to \infty$ then $|\hat{L} - \mu_L| \to 0$. In practice we have that when, the number of source samples is sufficiently high, the predictions of the GPs precise and a good generalization from the regressor used in WFQI, the bias introduced by the use of biased samples with estimated importance weights is reasonably low.

We could also consider the bias introduced by the use of $\hat{Q}^1(s,a)$ in place of the reward coming from the experience sample. This error is composed of two parts: the first one is due to the generalization error introduced by the specific regressor used in WFQI, the second part, again, is given by the use of estimated weights for the reward model and can be studied with an analysis analogous to the one of the transition model. Notice this bound does not provide any insight on how the errors propagate in different iterations of WFQI.

The next chapter is devoted to the empirical validation of the theoretical results presented so far. We observe how different number of source and target sample affects the performance of WFQI and appears to be always respectful of the results obtained in this chapter.

Chapter 6

Experimental Results

This chapter is devoted to the experimental evaluation of WFQI on classic RL problems. We test the proposed algorithm on two versions of the puddle world and on the activity swing-up. Both environments have a continuous state-space and discrete action-space. We evaluate the performance of WFQI in terms of expected return under different numbers of source/target samples varying also the number of source tasks involved.
We start presenting the metrics we adopt to evaluate our approach and finally, we report the results over the different environments.

6.1 Evaluation Metrics

Defining a metric for the notion the notion of "good transfer" is, in general, a hard task and in literature different metrics have been proposed. As already highlighted in the previous section we plot the results in terms of expected discounted return J_π of the best policy learnt by WFQI after performing the transfer:

$$J_\pi = \sum_{t=0}^{T} \gamma^t R(s_t, \pi(s_t)) \tag{6.1}$$

The resulting graphs are quite useful because they permit to have a graphical overview of the jumpstart improvement and of the time to learn the optimal policy from the source and target samples.

6.2 The Puddle World Environment

In this set of experiments we deal with MDPs with continuous state space (and discrete action space).
Puddle-world is an environment similar to grid-world where the agent must move toward a safe area while avoiding to walk inside the puddles. Only four actions (N,S,W,E) are allowed in each state. When the agent bump in the border of the environment it always return to the state where the action was initially performed.
The settings for the experiment is the following: two task (target and source), $\gamma = 0.99$, the size is 10×10, and the maximum horizon is set to 30 iterations.

Puddles are modeled as (inverse) bivariate gaussian:

$$
\begin{aligned}
P(x, y) &= -\frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho}} e^{-\frac{z}{2(1-\rho)^2}} \\
z &= \frac{(x-\mu_x)^2}{\sigma_x^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} + \frac{(y-\mu_y)^2}{\sigma_y^2} \\
\rho &= \frac{\sigma_{xy}}{\sigma_x\sigma_y}
\end{aligned}
\tag{6.2}
$$

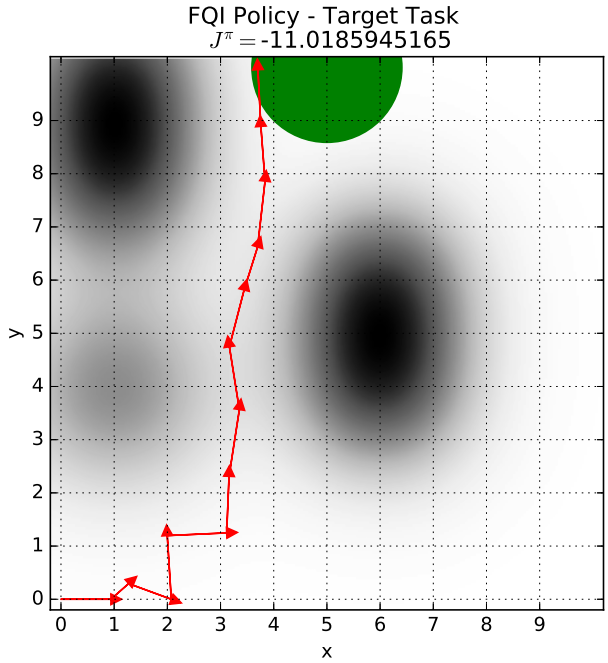

Figure 6.1: *An example puddle world environment, in red the FQI policy.*

We propose two variations of the puddle world environment: in the first

one when the agent comes close to a puddle he receives a penalty proportional to the distance from the centre of the puddle; in the second variation when the agent come close to a puddle the length of the step reduces proportionally with respect to the distance from the center of the puddle.

In the first variation the reward for (s, a) is given by the following set of equations. Let be $\mathcal{U}$ the set of puddles then:

$$r(s,a) = \begin{cases} \mathcal{N}(0, 0.01) + \sum_{u \in \mathcal{U}} 100 P_u(s') & s' \textit{ is at goal} \\ -1 + \mathcal{N}(0, 0.01) + \sum_{u \in \mathcal{U}} 100 P_u(s') & s' \textit{ otherwise} \end{cases} \tag{6.3}$$

The dynamics is simply given by modifying deterministically the coordinates of s depending on the action a chosen by the agent, the step-lenght α is unitary. Finally we add a Gaussian noise $\mathcal{N}(0, 0.04)$ to both the coordinates to obtain s'.

In the second variation the reward is still given by Equation 6.3, but the dynamics is now dependent also on the position of the agent. In particular α is no more unitary but given by the following equation:

$$\alpha = \frac{1}{1 + 30 \sum_{u \in \mathcal{U}} P_u(s')} \tag{6.4}$$

When the agent comes closer to the center of a puddle $\alpha \to 0$.

6.3 Target and Sources Tasks

In this section, we present the tasks involved in the experiments. The source tasks have different degrees of similarity with respect to the target task. The source task 3 (figure 6.5) has the highest similarity but the puddles are slightly misplaced and the goal region is smaller and shifted on the right of the space. The source task 1 (figure 6.3) maintains some similarities with the target but a large puddle on the bottom-left corner may produce very different policies. The last source task (figure 6.4) is the one that has the highest differences with respect to the target, indeed the puddle configuration is very different.

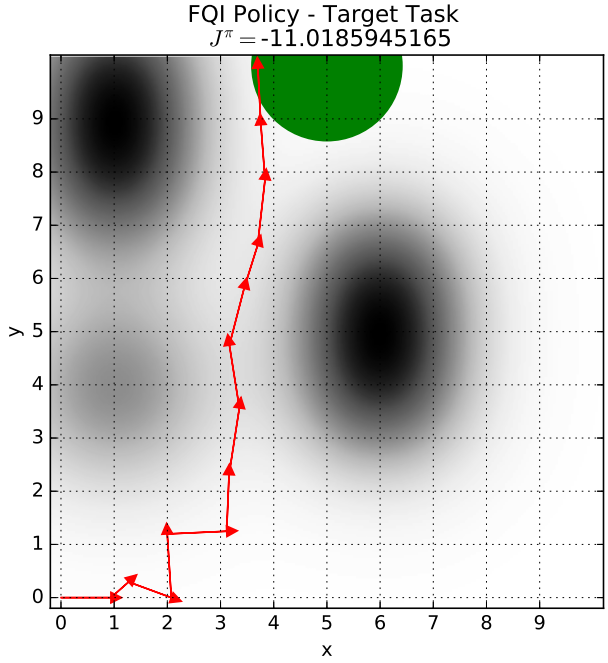

Figure 6.2: *The target task used in the experiments, in red the FQI policy.*

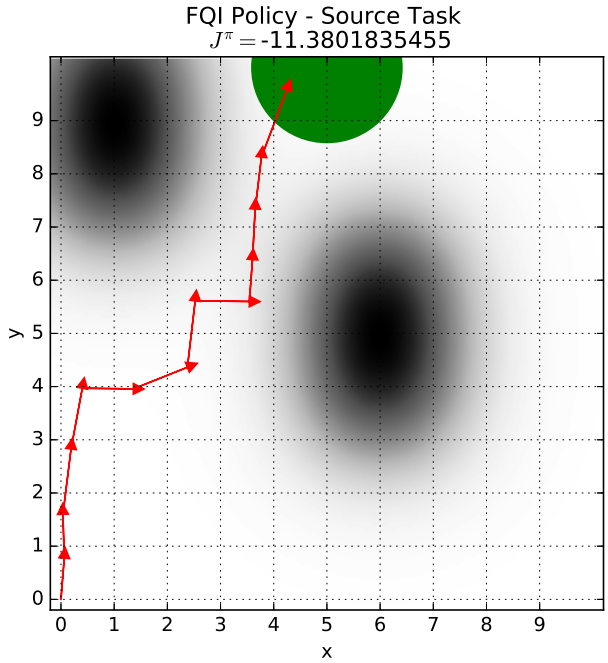

Figure 6.3: *Source task 1 used in the experiments, in red the FQI policy.*

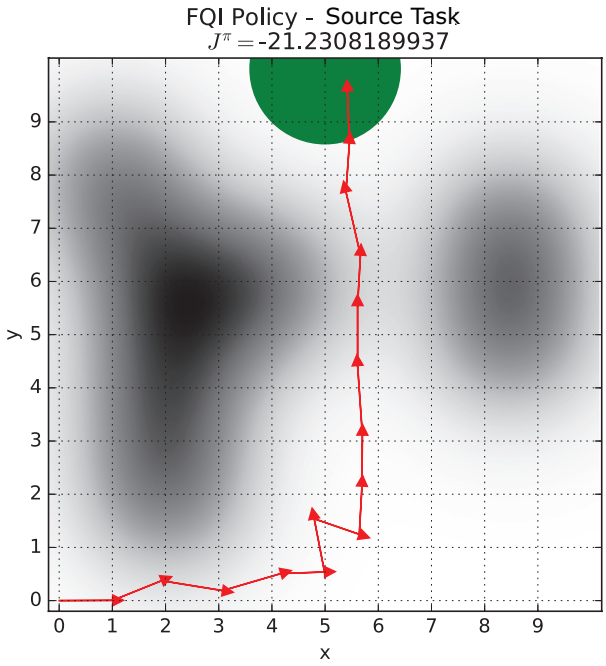

Figure 6.4: *Source task 2 used in the experiments, in red the FQI policy.*

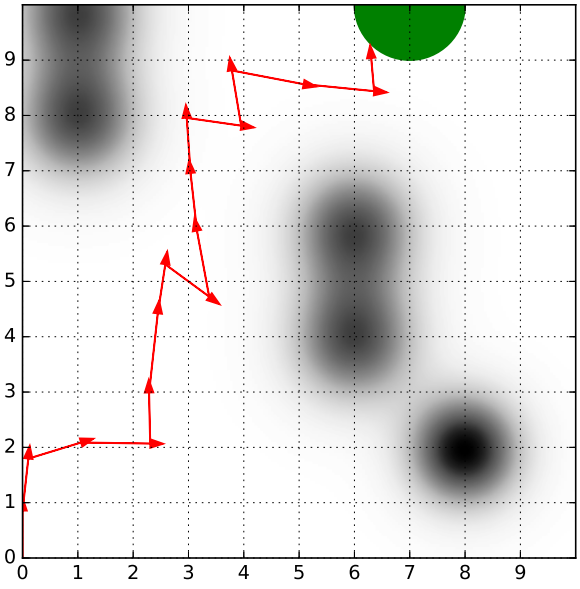

Figure 6.5: *Source task 3 used in the experiments, in red the FQI policy.*

6.4 Results - Puddle World Version 1

This version of puddle world is meant to test the reward transfer capability of the algorithm. In this scenario when the agent comes closer to a puddle it receives a penalty on the reward proportional to the distance with respect to the centre of the puddle. The dynamics of the agent are unchanged (ie. the step remains unitary).
From the perspective of the weight calculation this means that only the weight associated with the reward needs to be estimated, the weights associated with the dynamics are always unitary.
In the following experiments, the results are averaged over 50 runs, the initial position of the agent is randomized at each run in the area $[0, 2][0, 2]$.

The parameters for the tree-based WFQI algorithm are: 50 estimators, 5 random splits and 2 minimum samples per leaf. Source samples are collected using the red policies reported in the previous figures.

6.4.1 Ideal Weights

For this set of experiments we assume to be able to calculate the ideal weights, that is we assume to perfectly know the model of the reward (the dynamics are identical in both the source and target task). Given a perfect knowledge of the model of the environment, it is possible to obtain a perfect estimation of the weights leading to an unbiased use of the source samples in the target task.

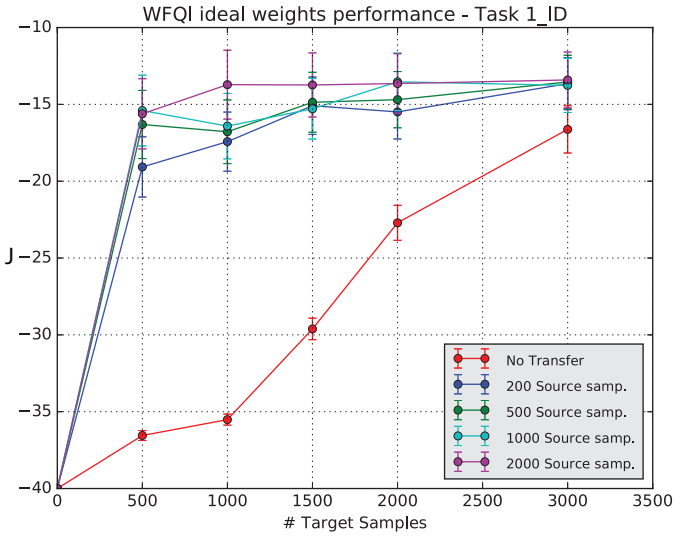

Figure 6.6: *WFQI performance Source 1 / Target task, ideal weights*

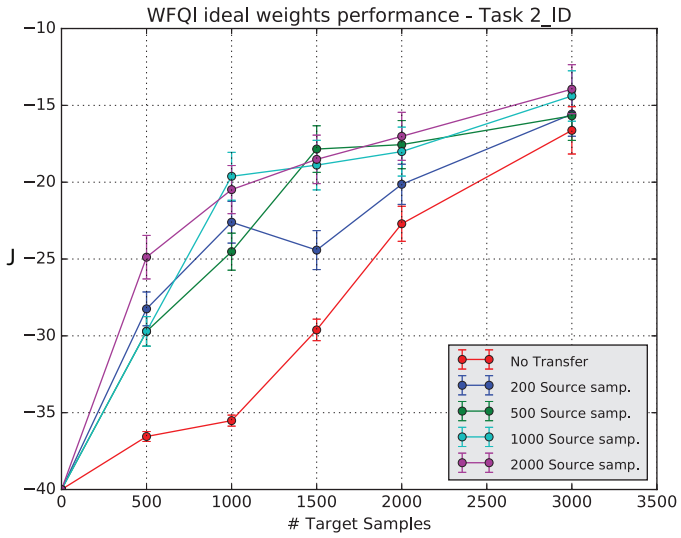

Figure 6.7: *WFQI performance Source 2 / Target task, ideal weights*

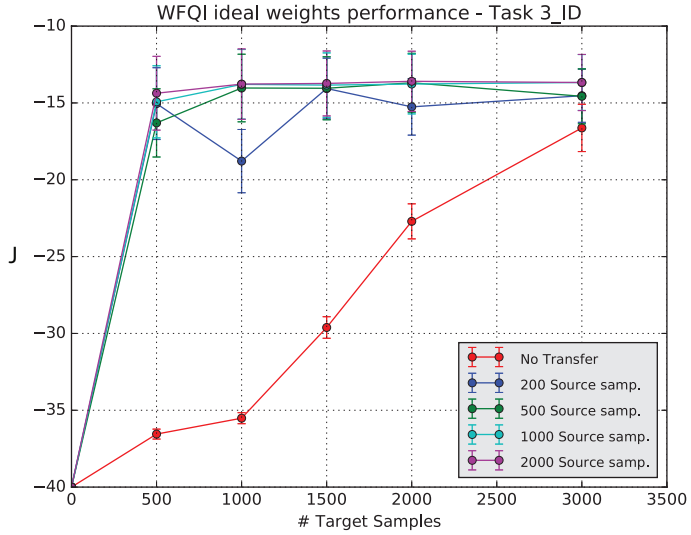

Figure 6.8: *WFQI performance Source 3 / Target task, ideal weights*

In this last set of experiments the samples are transferred from all the source tasks toward the target in equal quantities.

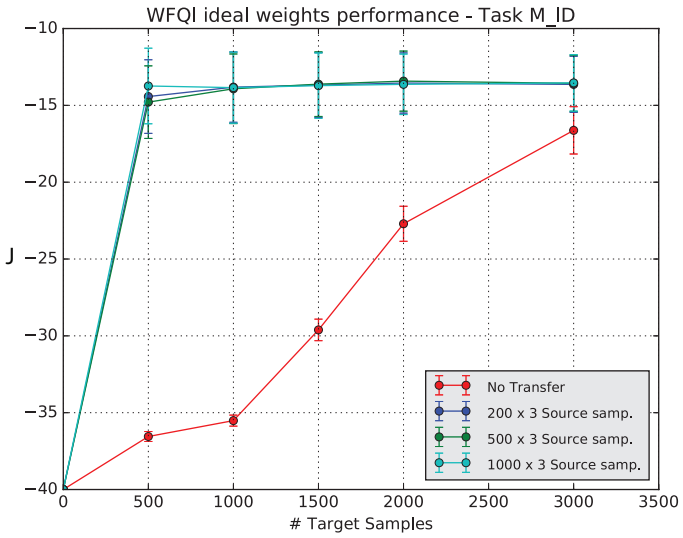

Figure 6.9: *WFQI performance Source 1/2/3/Target task, ideal weights*

The results reflect the theoretical expectation: the best performances are achieved with the source tasks 1 and 3, on the other side, the source task 2 appear to be of little help.
Another important observation is that our algorithm is successfully able to limit the phenomena of the negative transfer; indeed the performance in figure 6.9 seems to not be influenced by the bad sample present in the source task 2.

6.4.2 Estimated Weights

For this set of experiments we assume to be not able to calculate the ideal weights, they are calculated using algorithm 3. These algorithms provide only an estimation of the real weights, therefore the use of the corresponding source samples in the target task is not perfectly unbiased (as in the previous scenario). A theoretical bounds over the introduced error has been given in the next chapter. Weights are estimated according to Equation 4.20.

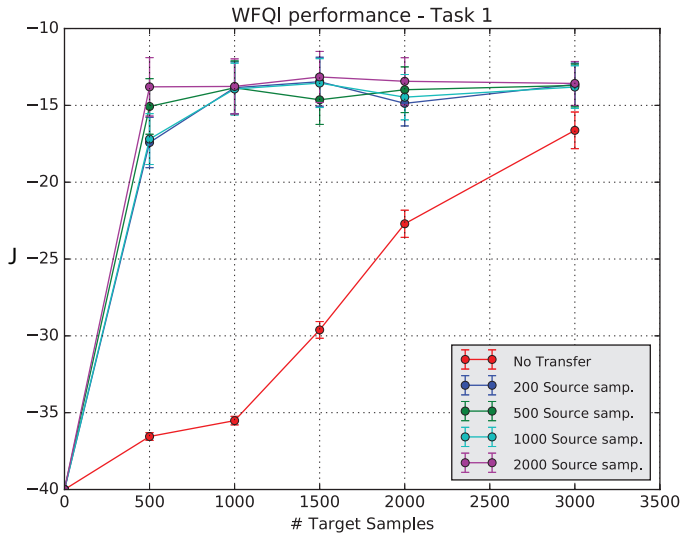

Figure 6.10: *WFQI performance Source 1 / Target task, est. weights*

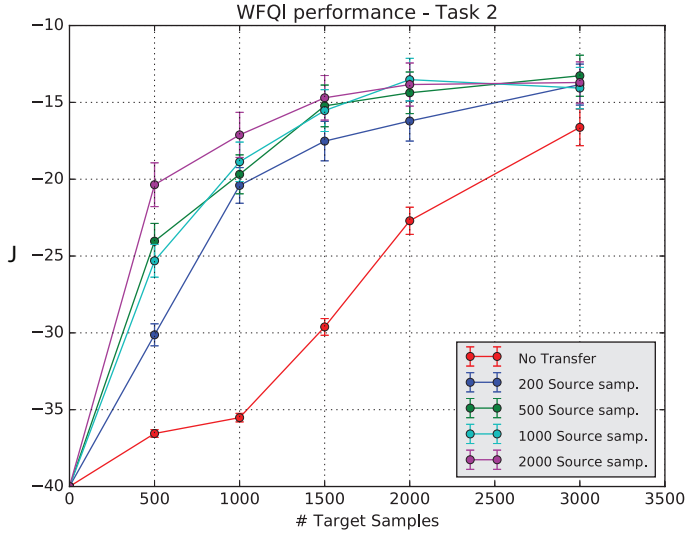

Figure 6.11: *WFQI performance Source 2 / Target task, est. weights*

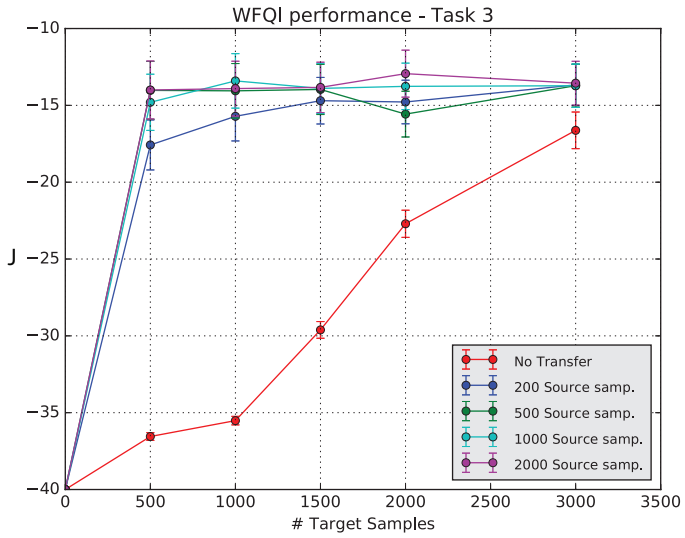

Figure 6.12: *WFQI performance Source 3 / Target task, est. weights*

In this last set of experiments the samples are transferred from all the source tasks toward the target in equal quantities.

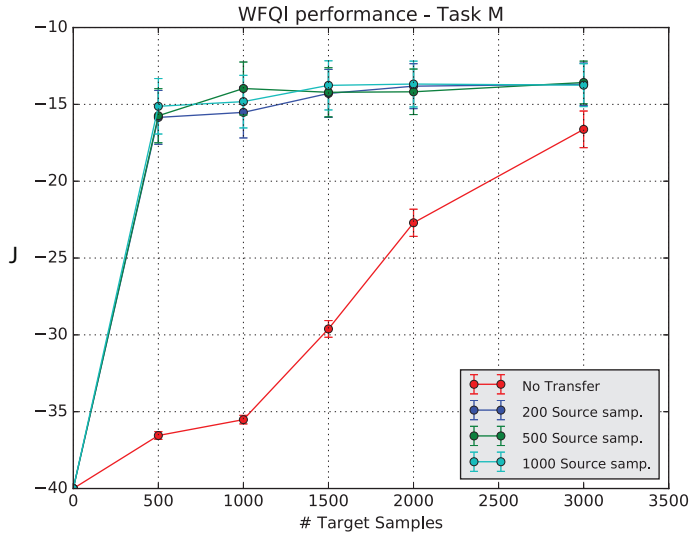

Figure 6.13: *WFQI performance Source 1 / 2 / 3 / Target task, est. weights*

Again he results reflects the theoretical expectation: the best performances are achieved with the source tasks 1 and 3, on the other side the source task 2 appear to be of little help.
Also in this situation our algorithm is able to maintain the ability to successfully limit the phenomena of the negative transfer; indeed the performance in figure 6.13 seems to not be influenced by the bad sample present in Source task 2.

In addition we provide the results obtained using Equation 4.13 using the correct and an overestimated value for σ^2. The task at hand is very simple but we can notice, in the case we do not fake the value of σ^2, a poor performance when the number of source samples is low. This effect is due to the fact that when $\sigma^2_{GP,S}$ is high formula 4.13 produces a very imprecise estimation of the weights.

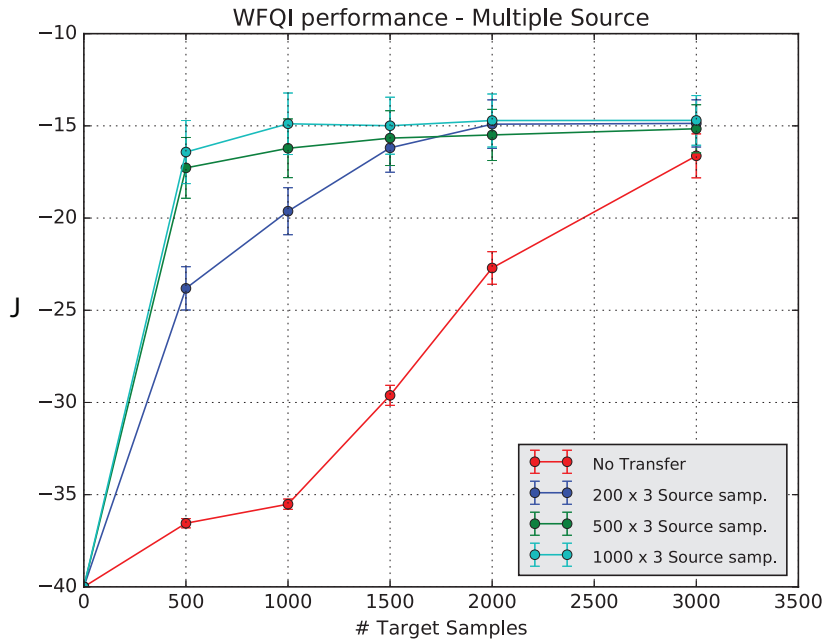

Figure 6.14: *Performance over Puddle-World V1, no σ^2 overestimation, $\sigma^2 = 0.01$, multiple source tasks*

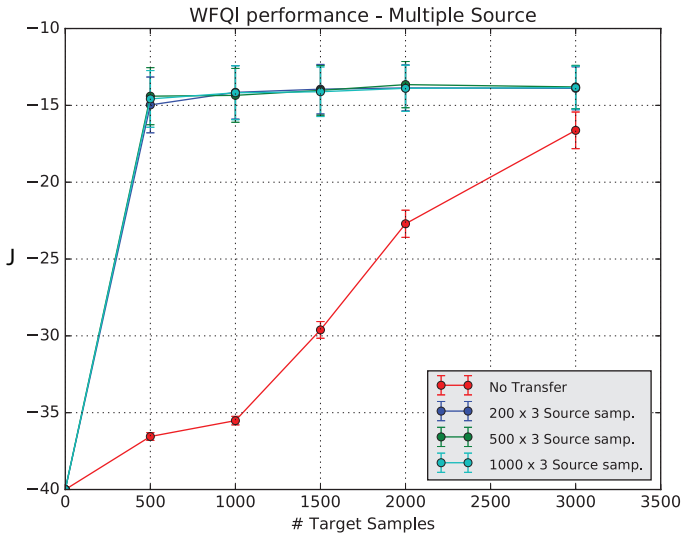

Figure 6.15: *Performance over Puddle-World V1, with σ^2 overestimation, $\sigma^2 = 0.09$, multiple source tasks*

As a final remark notice that using a weight $w = w_r * w_p$ in this environment would be of great disadvantage. Indeed here we assume that the transition model does not change between source and target task while the reward model can change arbitrary. This means we can assume $w_p = 1$ while the value of w_r can change depending on the specific sample. If the source task is significantly different from the target it means that w will assume a low value in the entire state-action space thus limiting the transfer. On the other hand with our approach, where reward weights are only used in the first iteration and transition weights are used in the remaining, the transfer procedure can fully exploit the similarities of the dynamics and thus maximizes the number of transferred samples.

6.5 Results - Puddle World Version 2

This environment is a variation of the previous puddle world, in this scenario when the agent is close to a puddle in addition to the reward penalty it also modifies its step length that decreases proportionally to the distance with respect to the center of the puddle. This situation is much more complicated from the transfer perspective: in addition to the weight necessary for the reward model, in this case we need to compute a weight also for the transfer of the dynamics of the agent (that now are changed).

6.5.1 Ideal Weights

In the first set of experiments we test the performance of WFQI using the ideal weights, that is we assume to perfectly know the stochastic model

for both the reward and transition model. As expected we obtain over the single task transfer an almost perfect performance, meaning that also with a very small amount of transferred source samples we reach the score achieved by the optimal policy.

The parameters for the tree-based WFQI algorithm are: 50 estimators, 2 random splits and 1 minimum samples per leaf.

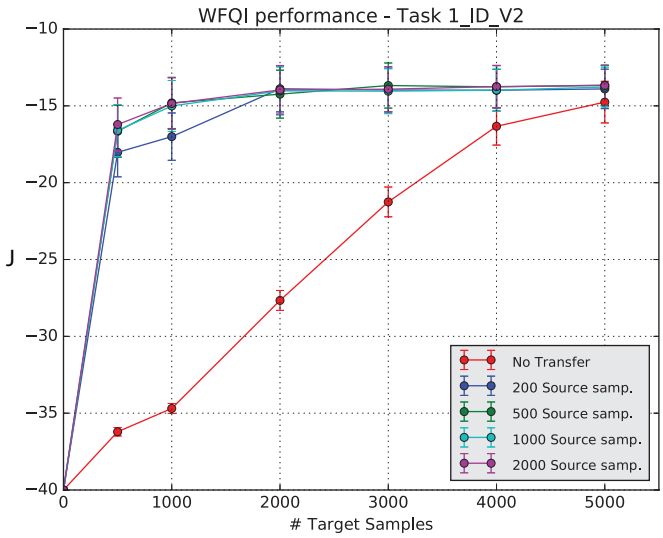

Figure 6.16: *WFQI performance Source 1 / Target task, ideal weights*

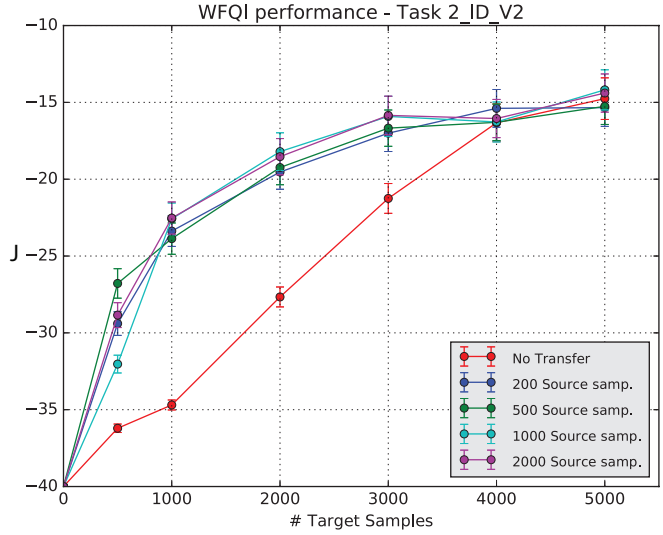

Figure 6.17: *WFQI performance Source 2 / Target task, ideal weights*

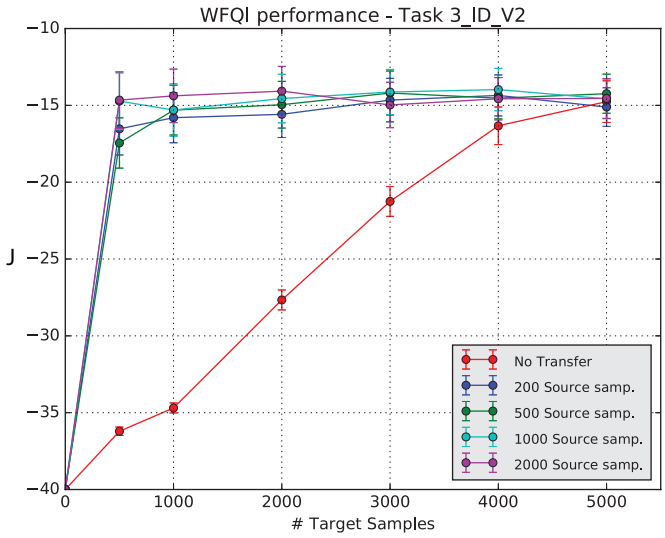

Figure 6.18: *WFQI performance Source 3 / Target task, ideal weights*

In this last set of experiments the samples are transferred from all the source tasks toward the target in equal quantities. Again we highlight the fact that despite the fact that samples coming from the second source task are not effective for the transfer, the algorithm is able to completely eliminate the negative transfer and to reach the maximum performance.

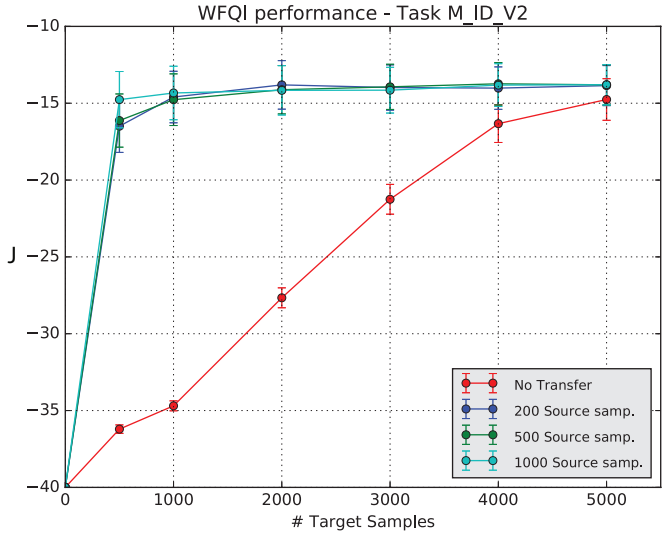

Figure 6.19: *WFQI performance Source 1 / 2 / 3 / Target task, ideal weights*

6.5.2 Estimated Weights

For this set of experiments weights are calculated using algorithms 3 and 4. These algorithms provide only an estimation of the real weights,

therefore the use of the corresponding source samples in the target task is not perfectly unbiased (as in the previous scenario). The performance is not as good as in the ideal case due to the error introduced thanks to the estimation of the weights. Weights are estimated according to Equation 4.20.

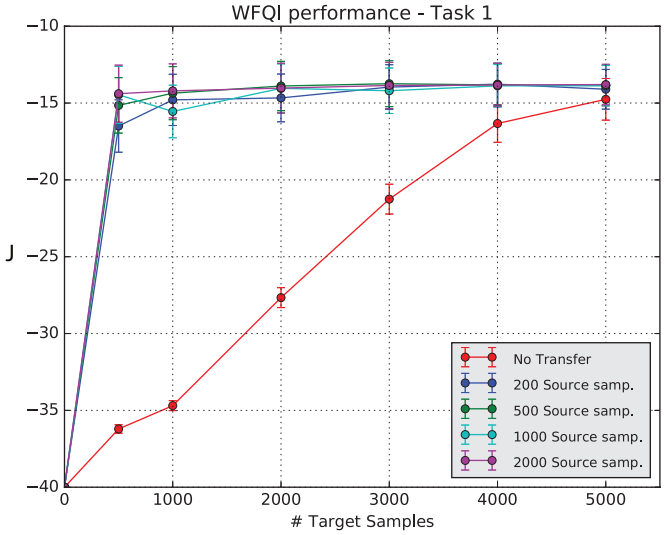

Figure 6.20: *WFQI performance Source 1 / Target task, est. weights*

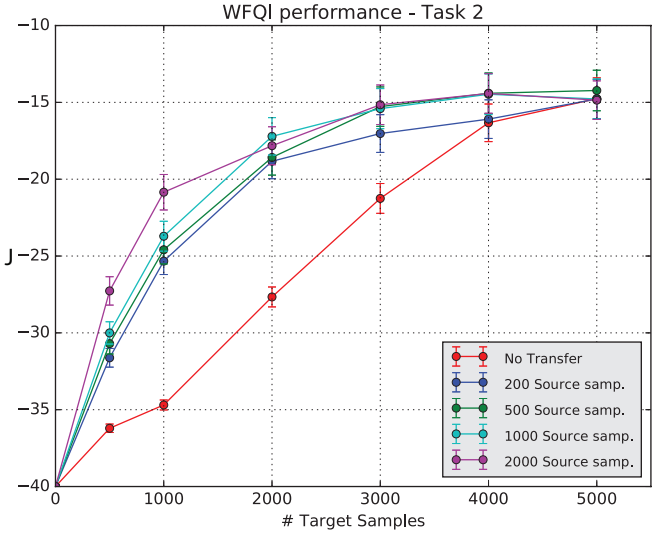

Figure 6.21: *WFQI performance Source 2 / Target task, est. weights*

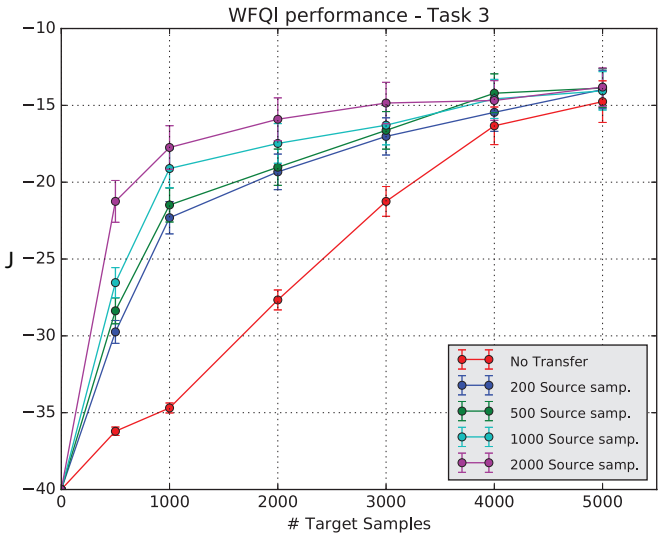

Figure 6.22: *WFQI performance Source 3 / Target task, est. weights*

In this last set of experiments the samples are transferred from all the source tasks toward the target in equal quantities. Also in this case, despite the fact that samples coming from the second source task are not effective for the transfer, the algorithm is able to eliminate the negative transfer effect and to achieve an satisfactory performance.

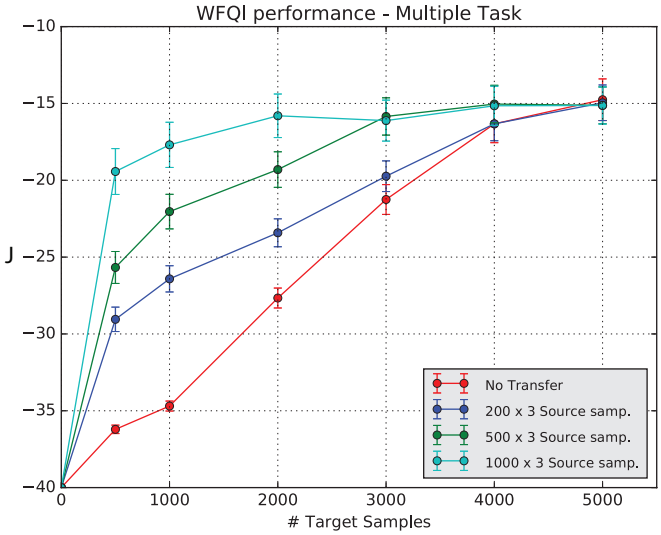

Figure 6.23: *WFQI performance Source 1 / 2 / 3 / Target task, est. weights*

In addition we also report the results when weights are estimated according to Formula 4.13 using respectively, in Figure 6.24 $\sigma_r^2 = 0.01$ and $\sigma_p^2 = 0.04$ and in Figure 6.25 $\sigma_r^2 = 0.09$ and $\sigma_p^2 = 0.36$.

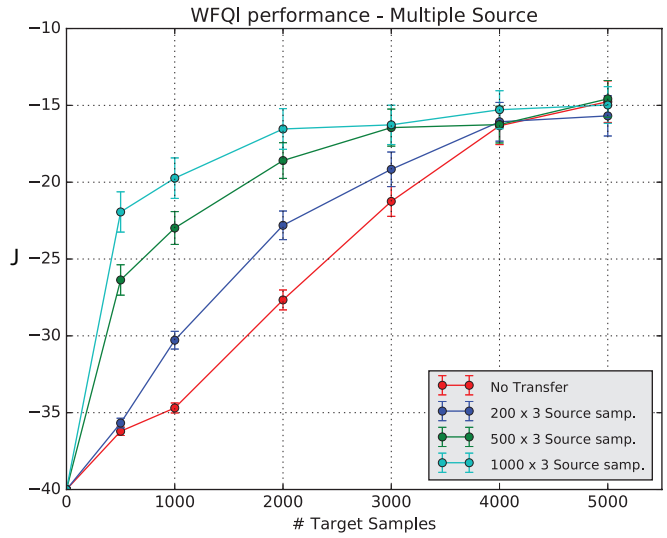

Figure 6.24: *Puddle-World performance, multiple source tasks, Equation 4.13* ***without*** σ^2 *overestimation*

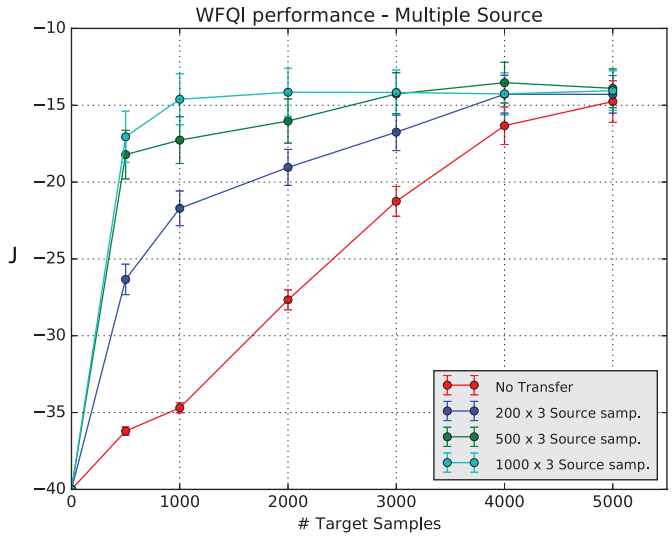

Figure 6.25: *Puddle-World performance, multiple source tasks, Equation 4.13* ***with*** σ^2 *overestimation*

We observe, in the first case, poor performances, mainly due high number of discarded samples. In the second graph we notice a much better performance due to the overestimation of σ^2 for both reward and transition model, in this case even better than the heuristic approach.

6.6 Acrobot Environment

Acrobot is a two-link underactuated robot, we refer to the model described by [5]. A graphical representation is depicted in figure 6.26.

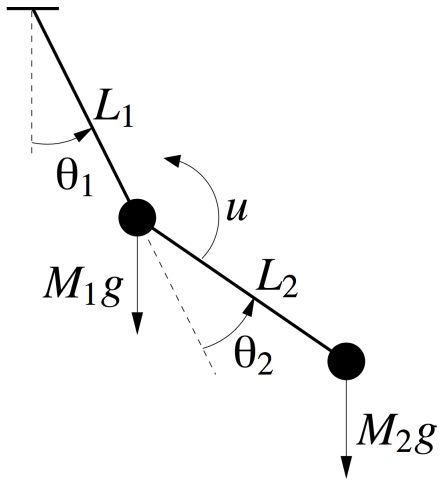

Figure 6.26: *A graphical representation of the Acrobot environment*

The state space of the problem is continuous and described by the tuple $[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$, where θ_1 and θ_2 are the angles of the two links and $\dot{\theta}_1$ and $\dot{\theta}_2$ are the two corresponding angular speeds. The action space is discrete and the only two possibility are to apply a torque of either -5 or 5.
The goal is to bring as fast as possible the two links in the upright position. The reward signal is -1 for every action and 0 when the two links are in the neighbourhood of the goal position.

The dynamical system equations of the Acrobot are given by:

$$\begin{cases} d_{11}\ddot{\theta}_1 + d_{12}\ddot{\theta}_2 + c_1 + \phi_1 = -\mu_1\dot{\theta}_1 \\ d_{12}\ddot{\theta}_1 + d_{22}\ddot{\theta}_2 + c_2 + \phi_2 = u - \mu_2\dot{\theta}_2 \\ d_{11} = M_1 L_1^2 + M_2(L_1^2 + L_2^2 + 2L_1 L_2 \cos(\theta_2)) \\ d_{22} = M_2 L_2^2 \\ d_{12} = M_2(L_2^2 + 2L_1 L_2 \cos\theta_2) \\ c_1 = -M_2 L_1 L_2 \dot{\theta}_2(2\dot{\theta}_1 + \dot{\theta}_2)\sin(\theta_2) \\ c_2 = M_2 L_1 L_2 \dot{\theta}_1^2 \sin(\theta_2) \\ \phi_1 = -(M_1 L_1 + M_2 L_1) g \sin(\theta_1) - M_2 L_2 g \sin(\theta_1 + \theta_2) \\ \phi_2 = -M_2 L_2 g \sin(\theta_1 + \theta_2) \end{cases} \tag{6.5}$$

$g = 9.81 m/s^2$ is the gravitational acceleration coefficient while μ_1 and μ_2 are the friction coefficients for the corresponding two joints.
The discount factor γ of the corresponding MDP is 0.95.

6.6.1 Tasks

For the Target task we set $L_1 = 1.0, L_2 = 1.0, M_1 = 1.0, M_2 = 1.0, \mu_1 = 0.1, \mu_2 = 0.1$.
For the Source task 1 we set $L_1 = 0.9, L_2 = 0.95, M_1 = 0.95, M_2 = 1.0, \mu_1 = 0.1, \mu_2 = 0.1$
For the Source task 2 we set $L_1 = 0.8, L_2 = 0.6, M_1 = 0.9, M_2 = 1.0, \mu_1 = 0.1, \mu_2 = 0.1$
For the Source task 3 we set $L_1 = 0.85, L_2 = 0.85, M_1 = 0.9, M_2 = 0.9, \mu_1 = 0.1, \mu_2 = 0.1$

Tests are performed singularly, $Source \rightarrow Target$ and in ensemble, $Source1/2/3 \rightarrow Target$.

6.6.2 Results - Acrobot

In this section we report the results relative to the Acrobot environment, tests are performed with the following parameters: $Estimators = 50, RandomSplit = 5, Min.Samplesperleaf = 2$, WFQI is executed over 50 iterations, the horizon of the MDP is set to 100. Results are averaged of 30 runs.
It is important to observe the these settings represents a much more difficult than the previous experiments: we are no more in a gaussian settings (while in Puddle World the dynamics and rewards models are assumed to be normally distributed), this means that the use of GPs introduces a further error (in addition to the bias of the estimated weights);

despite this further difficulty again we are able to obtain interesting advantages from our approach. Weights are estimate according to Equation 4.20.

Figure 6.27: *WFQI performance Source 1 / Target task, est. weights*

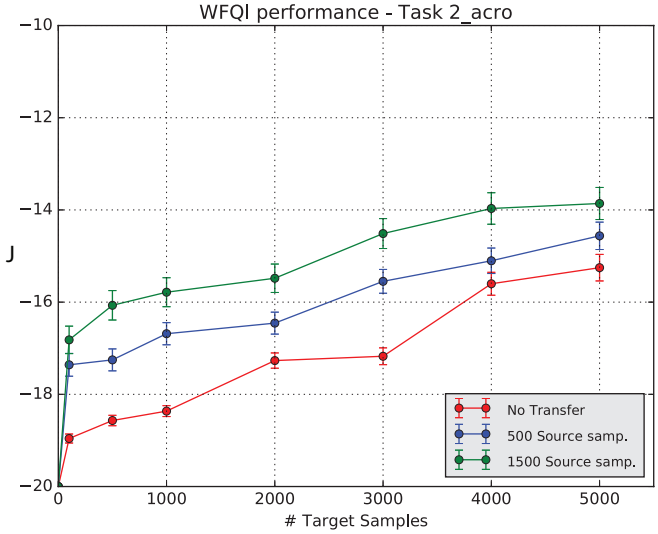

Figure 6.28: *WFQI performance Source 2 / Target task, est. weights*

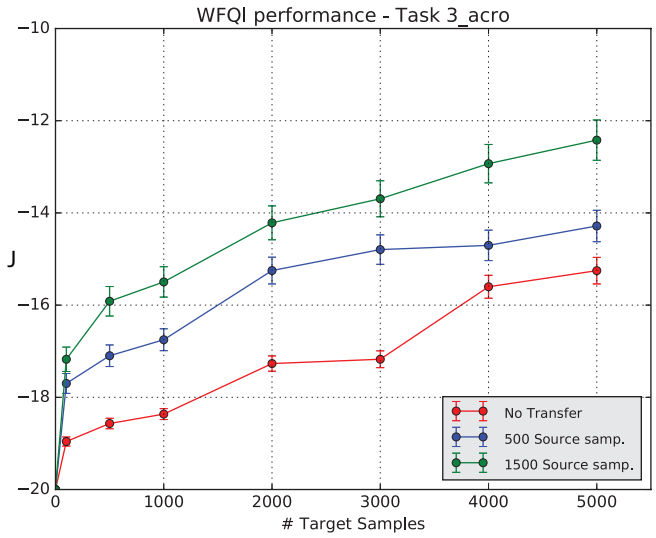

Figure 6.29: *WFQI performance Source 3 / Target task, est. weights*

Note that in Source task 2 the performance are lower given that it is very different from the target task.
In the last set of experiments we transfer from all the three source tasks to the target. Again our algorithm reveals to be robust toward the effect of the negative transfer (ie. samples from source task 2) and maintains an optimal performance using the information coming from the other two.

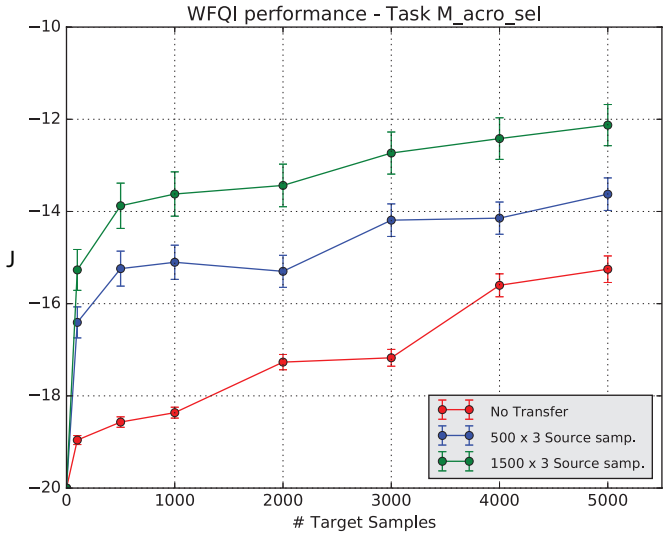

Figure 6.30: *WFQI performance Source 1 / 2 / 3 / Target task, est. weights*

In addition we also report the results when weights are estimated according to Formula 4.13 using respectively, in Figure 6.31 $\sigma_p^2 = 0.09$ and

in Figure 6.32 $\sigma_p^2 = 0.81$.

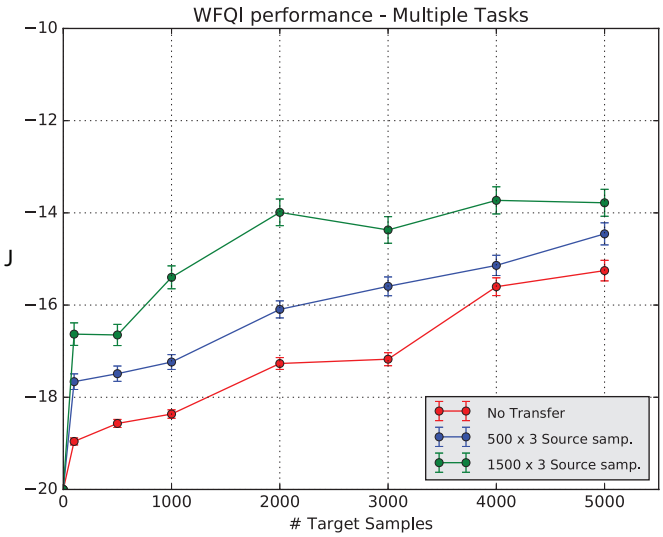

Figure 6.31: *Acrobot performance, multiple source tasks, Equation 4.13* ***without*** σ^2 *overestimation*

Figure 6.32: *Acrobot performance, multiple source tasks, Equation 4.13* ***with*** σ^2 *overestimation*

We observe, in the first case, poor performances, mainly due high number of discarded samples. In the second graph we notice a much better performance due to the overestimation of σ^2 for both reward and transition model. However notice, in this case, that the performance is not as good as in Figure 6.30. This is mainly due to the fact that Acrobot is a

non-gaussian environment and σ^2 is just the result of an educated guess and may be not the best choice for this specific scenario. Better results could be obtained by using some estimation procedures for σ^2.

6.7 Comparison

In this section we compare our approach with the one presented in [10] (abbreviated as BATCH). This last approach is based on the calculation of measures between tasks and samples. When the target and source tasks share some similarities (in this case the authors assume the tasks to be drawn from the same probability distribution Ω) it is possible to measure the distance (called task compliance) between a target and a source task, moreover within each source we can measure the likelihood of each sample (sample relevance) to be generated in the target task. The algorithm first calculate for each source task the compliance and then for each sample of each source, it calculates the relevance. Samples are drawn from each task proportionally to the calculated compliance and relevance. For more detail we remand to [10].

We compare the performance of WFQI and BATCH over the two environments defined before (Puddle-World and Acrobot).
For Puddle-World, we use a total of 1000 source samples extracted from the three source tasks at hand. For WFQI the parameters are identical to the ones presented in the previous section. For BATCH we maintain the same parameters reported in [10]. For WFQI calculate weights using the overestimation of σ^2 (in particular we take $\sigma^2 = 0.09$).

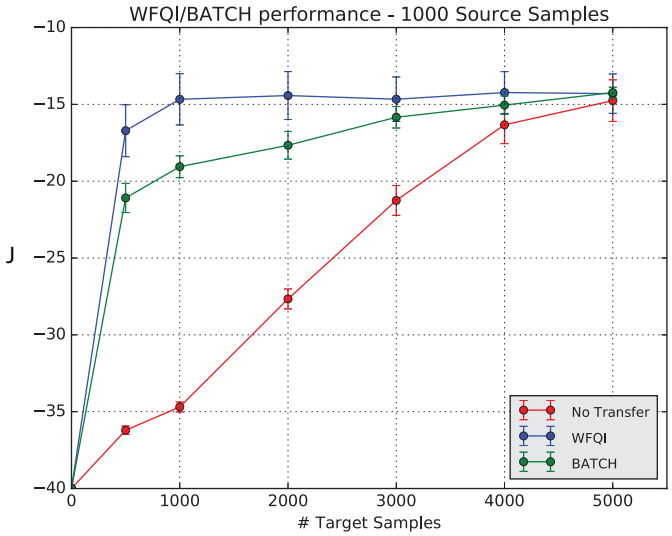

Figure 6.33: *Puddle World - Performance comparison BATCH vs WFQI*

Notice how our approach is able to gain a better jumpstart performance over BATCH when the number of target samples is low.

Moreover we propose a similar comparison also over the Acrobot environment. In this case 1000 source samples are used, parameters for WFQI are unchanged with respect to the previous case. For BATCH we use ($\delta_{s,a} = 0.3, \delta_r = 0.5, \delta_p = 0.3, \mu = 0.5$) all the other parameters regarding FQI are unchanged with respect the ones presented in the previous set of experiments.

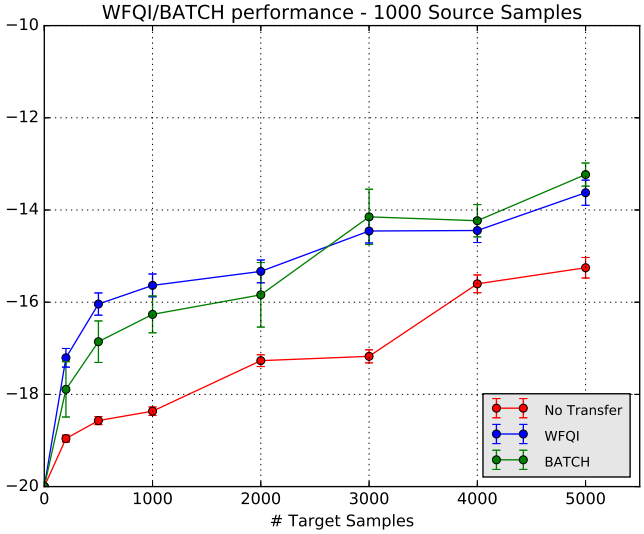

Figure 6.34: *Acrobot - Performance comparison BATCH vs WFQI*

Again we notice how WFQI is able to gain an advantage in the first part of the test, when the number of target samples is low, but in the final part the performance are identical.

Chapter 7

Conclusions and Future Works

THe goal of our work was to provide an efficient way to perform the transfer of samples from a set of source tasks to a target task with common state-action space. We focus on the context of Batch Reinforcement Learning, and in particular in Fitted Q-Iteration. We propose a transfer procedure based on the idea of importance sampling. We calculated a pair of weights w_p and w_r for each source samples (s, a, s', r) involved in the transfer. These weights, according to the theory of importance sampling, were defined as the ratio between the likelihood of observing a given reward/transition in the target task and the likelihood observing a given reward/transition in the corresponding source task. A low importance weight indicates a sample unlikely to be generated in the target task vice versa for a high importance weight.
We studied different possibilities to inject these weights inside FQI: the first idea was to consider an overall weight $w = w_p w_r$ and use a weighted regression inside FQI, weighting each sample with w. This approach appeared to work but resulted to be very inefficient in many scenarios. With inefficient we mean that the approach does not maximize the number of transfer samples. For example, consider a scenario where the dynamics are shared between the tasks but the reward models appear to be very different, thus the overall weight w will, in general, be very low and therefore limiting the amount of transferred samples. The considered

final, and better, approach was to consider w_r and w_p separately and inject them separately inside FQI. We used the reward weights only for the first iteration and for the remaining ones only the transition weights, thus maximizing the number of transferred samples also in situation similar to the one described before.
Finally, we proposed a set of benchmarks to validate the performance of WFQI. We tested our approach on a single and multiples source task scenarios where samples are transferred first from each source task separately and then from all the source task at the same time. We compared the expected return in the cases where we transfer different amount of samples and when we transfer no sample at all. Our approach resulted to be successful in different configurations environment being able to improve the performance using samples selected from similar tasks. We also provided a theoretical analysis of WFQI in which we bounded the error introduced by the use of biased (source) samples and we motivated our criterion to obtain an accurate estimation of the importance weights.

In the remaining part of this chapter we propose several guide lines for possible future works:

- **Weights Selection**: In Chapter 5 we proposed a possible way to deal with the estimation of the importance weights, in particular we came up with the following estimator:

  $$\hat{w}(x) = \frac{\mathcal{N}(x; \mu_{GP,T}, \sigma^2 + \sigma^2_{GP,T})}{\mathcal{N}(x; \mu_{GP,S}, \sigma^2 + \sigma^2_{GP,S})}$$

  while this estimator appears to work very well in practice (and converges to the true weights under ideal conditions) we would like to obtain stronger theoretical properties on $\hat{w}$. A possible idea would be to understand some properties of the real distribution of the reward/transition weights for a given sample. In particular it would be interesting to understand the role of $\hat{w}$ in this distribution.
  Moreover the flexibility of our approach allow for very different approaches for the weights estimation, for example by setting up an optimization problem with the objective of choosing the weight of each sample minimizing the overall bias.

- **Errors Propagation**: In Chapter 8 we discussed some of theoretical properties of Weighted Fitted Q-Iteration. In particular we propose a bound on the bias on the loss function introduced by the use of source samples. One limitation of this result is that it is local to one iteration of our learning procedure. A possible development would be to understand how this local error propagates among different iterations of WFQI and if this error indeed converges to zero when the number of source samples and the accuracy of the Gaussian Processes increases.

- **Algorithm Performance**: One of the main practical (from an algorithms perspective) limitations of WFQI is the use of Gaussian Processes to get estimations of the reward and transition models. While, in practice, this approach reveals to be very effective it does not permit our algorithm to scale when a large number of samples is required to solve the task at hand (for example in possible application to Deep Reinforcement Learning, see [12]). A possible way to proceed would be to study the use of alternative regressors with better time performance also when a high number of samples is required.

Bibliography

[1] Christopher G Atkeson and Juan Carlos Santamaria. “A comparison of direct and model-based reinforcement learning”. In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE. 1997, pp. 3557–3564 (cit. on p. 23).

[2] Justin A Boyan. “Technical update: Least-squares temporal difference learning”. In: *Machine Learning* 49.2 (2002), pp. 233–246 (cit. on p. 27).

[3] Paul Bromiley. “Products and convolutions of gaussian probability density functions”. In: *Tina-Vision Memo* 3.4 (2003) (cit. on pp. 53, 54).

[4] Damien Ernst. “Near optimal closed-loop control. Application to electric power systems”. PhD thesis. University of Liege, Belgium, 2003 (cit. on p. 27).

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. “Tree-based batch mode reinforcement learning”. In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 503–556 (cit. on pp. 27, 28, 30, 32, 84).

[6] Geoffrey J Gordon. “Approximate solutions to Markov decision processes”. In: *Robotics Institute* (1999), p. 228 (cit. on p. 27).

[7] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. “Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance”. In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. Vol. 1. IEEE. 2005, pp. 85–89 (cit. on p. 24).

[8] Alessandro Lazaric. “Transfer in reinforcement learning: a framework and a survey”. In: *Reinforcement Learning*. Springer, 2012, pp. 143–173 (cit. on pp. 17, 35, 36).

[9] Alessandro Lazaric and Marcello Restelli. “Transfer from multiple MDPs”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 1746–1754 (cit. on p. 35).

[10] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. “Transfer of samples in batch reinforcement learning”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 544–551 (cit. on pp. 16, 17, 31, 35, 45, 89).

[11] Xuejun Liao, Ya Xue, and Lawrence Carin. “Logistic regression with an auxiliary data source”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 505–512 (cit. on p. 34).

[12] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533 (cit. on p. 93).

[13] Gerhard Neumann and Jan R Peters. “Fitted Q-iteration by advantage weighted regression”. In: *Advances in neural information processing systems*. 2009, pp. 1177–1184 (cit. on p. 29).

[14] Dirk Ormoneit and Saunak Sen. “Kernel-based reinforcement learning”. In: *Machine learning* 49.2 (2002), pp. 161–178 (cit. on p. 26).

[15] Art B. Owen. *Monte Carlo theory, methods and examples*. 2013 (cit. on p. 47).

[16] David Pardoe and Peter Stone. “Boosting for regression transfer”. In: *Proceedings of the 27th international conference on Machine learning (ICML-10)*. 2010, pp. 863–870 (cit. on p. 34).

[17] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 99).

[18] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. “Off-policy temporal-difference learning with function approximation”. In: *ICML*. 2001, pp. 417–424 (cit. on p. 23).

[19] Satinder P Singh, Andrew G Barto, and Nuttapong Chentanez. “Intrinsically Motivated Reinforcement Learning.” In: *NIPS*. Vol. 17. 2. 2004, pp. 1281–1288 (cit. on p. 17).

[20] Satinder Singh et al. “Convergence results for single-step on-policy reinforcement-learning algorithms”. In: *Machine learning* 38.3 (2000), pp. 287–308 (cit. on p. 23).

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998 (cit. on p. 23).

[22] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063 (cit. on p. 24).

[23] Liu Yang, Steve Hanneke, and Jaime Carbonell. “A theory of transfer learning with applications to active learning”. In: *Machine learning* 90.2 (2013), pp. 161–189 (cit. on p. 34).

Appendices

Appendix A

Implementation

In this appendix we describe in details the practical implementation that we used to obtain the results of the experiments chapter. The main procedures are described in pseudo-code in order to obtain a more compact and abstract representation of the main ideas.

For reference the complete implementation, in Python 3.4, is available at:
`https://github.com/Sessa93/ifqi`.

In the implementation there are 3 main important part:

1. **WFQI** (Algorithms 1 and 2): The class (W)FQI provides a general interface for a FQI implementation that abstracts from the specific regressor used. In this case a pseudocode implementation of (W)FQI has been already proposed in chapters 4 and 6. This abstraction permits to reuse the same interface to test different regressor implementation, this idea is not particularly exploited in this thesis but may be particularly useful for more advanced implementation. Again the only differences in the implementation of FQI and WFQI is the use, in the latter, of importance weights. These differences reflects in the structure of the single sample that now needs to include the weights for reward and transition model.
2. **Action Regressor** (algorithm 5): The idea behind the use of an action regressor is, when the set of action $\mathcal{A}$ is discrete and finite, to use a separate regressor for each action $a \in A$. Every time (W)FQI asks for a prediction over a given sample, depending on the action specified by the sample the correct regressor is used.

3. **Regressor**: This class specify a generic interface for a regressor implementation. In our particular case this is specified as an Extremely Randomized ensemble regressor (for the details refer to chapter 4). The interface provides two methods: fit and $predict$, the former permits to train the the regressor, the latter permits to obtain predictions for new data points.

Algorithm 5 Action Regressor

1: $regressors \leftarrow \{\textbf{new} \quad regressor_a \quad \forall a \in \mathcal{A}\}$
2: **procedure** FIT(X, Y)
3: **for** $a \in \mathcal{A}$ **do**
4: $i \leftarrow X[1].indexOf(a)$
5: $regressors[i].fit(X[i,:], Y[i,:])$
6: **procedure** PREDICT(X)
7: **for** $a \in \mathcal{A}$ **do**
8: $i \leftarrow X[1].indexOf(a)$
9: $regressors[i].predict(X[i,:])$

For the implementation of the Extra Trees and GP regressors we refer to the standard implementation available in the Scikit-learn [17] library:

- `http://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html`
- `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html`

Appendix B

Gaussian Process

GAussian Process (sometimes abbreviated as GP) is a supervised learning model that found large application in the domain of regression, classification and clustering. In this appendix we focus on application of GPs over a generic regression problem.
There are many ways to interpret this type of model, for this section we will focus on the *weight-space view* and on the *function-space view*. The latter interpret a gaussian process as an entity defining a distribution over function, hence the name, and inference taking place directly inside the space of the functions. On the other hand the weight-space view is simpler to grasp but at the same time completely equivalent with the first interpretation.

The appendix is divided into two main section, representing the two way to interpret a GP model. The main results known in literature are given. It also important to remark that this appendix is of fundamental importance in our work given that the entire weight estimation procedure for WFQI relies on gaussian processes.

B.1 Weight-Space View

We start our discussion with the weight-space view given that it can be obtained starting from a simple linear regression model.

B.1.1 Bayesian Linear Regression

We initially consider the usual linear regression model

$$f(\boldsymbol{x}) = \boldsymbol{x}^t \boldsymbol{w} \qquad y = f(\boldsymbol{x}) + \epsilon$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the input variable, f is true function we are trying to fit, $\boldsymbol{w}$ is the vector of the weight and $y \in \mathbb{R}$ is the observed target value.
ϵ is the additive **gaussian** noise we suppose is present in the observed samples. In particular we have

$$\epsilon \sim \mathcal{N}(0, \sigma_e^2)$$

Under independence assumption of the observation, we can write the likelihood function of our model (ie. the probability that our model has generated the observed target):

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_e} e^{-\frac{(y_i - \boldsymbol{x}_i^t \boldsymbol{w})^2}{2\sigma_e^2}} = \mathcal{N}(\boldsymbol{X}^t \boldsymbol{w}, \sigma_e^2 \boldsymbol{I}) \tag{B.1}$$

where $\boldsymbol{I}$ indicates the identity matrix of order n.

As in every bayesian setting we need to specify a prior distribution over the space of the variables we are going to fit, in our case we need to specify a probability distribution over $\boldsymbol{w}$ and for this discussion we assume:

$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_p)$$

To make inference possible in this context we need the posterior distribution, that, given the prior over the parameters can be easily obtained using the Bayes theorem:

$$p(\boldsymbol{w}|\boldsymbol{y}, \boldsymbol{X}) = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})} \tag{B.2}$$

Note that $p(\boldsymbol{y}|\boldsymbol{X})$, known as marginal distribution, is independent from $\boldsymbol{w}$ and acts only as a normalizing factor.

With some simple algebraic manipulation we obtain the following result for the posterior distribution:

$$\begin{aligned} p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) &= \mathcal{N}(\bar{\boldsymbol{w}}, A^{-1}) \\ \bar{\boldsymbol{w}} &= \frac{1}{\sigma_e^2} A^{-1} \boldsymbol{X} \boldsymbol{y} \\ A &= \frac{\boldsymbol{X}\boldsymbol{X}^t}{\sigma_e^2} + \Sigma_p^{-1} \end{aligned} \tag{B.3}$$

Given that in gaussian settings the mode coincide with the mean of the distribution this is, by construction a Maximum a Posteriori (MAP) estimator.

Finally we want to make predictions: this is possible by averaging all the possible parameters value *weighted* by their posterior probability. With

other words the prediction for a new test input is given by averaging the output of all the possible linear model with respect to the gaussian posterior distribution. In formula:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}) d\boldsymbol{w} = \mathcal{N}(\frac{1}{\sigma_e^2} \boldsymbol{x}_*^t A^{-1} \boldsymbol{X}\boldsymbol{y}, \boldsymbol{x}_*^t A^{-1} \boldsymbol{x}_*)$$

It is important to notice that the predictive distribution is gaussian with a mean given by the posterior mean of the weights from eq. B.3 multiplied by the test input.

The model obtain is a simple version of a linear gaussian process. In the next section we describe an extended version of the previous model by projecting the input space in the feature space.

B.1.2 Kernel Trick

In the previous paragraph we have described a simple linear model which suffers from very limited expressiveness. To expand the previous model we introduce a new function ϕ that maps the n-dimensional vector $\boldsymbol{x}$ to an N-dimensional feature space.
Our model is now

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^t \boldsymbol{w} \tag{B.4}$$

All the equations stated in the previous section do not change except by substituting $\boldsymbol{X}$ with $\Phi(\boldsymbol{X})$ where the latter expression is an abbreviated formula to indicate the columns $\phi(\boldsymbol{x})$ for all the possible choices of $\boldsymbol{x}$.

A computationally efficient expression for the predictive distribution that takes in account the projection is given by the following equation:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(\phi_*^T \Sigma_p \Phi (K + \sigma_p^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \phi_*^T \Sigma_p \phi_* - \phi_*^T \Sigma_p \Phi (K + \sigma_p^2 \boldsymbol{I})^{-1} \phi_*^T \Sigma_p \phi_*) \tag{B.5}$$

where $K = \Phi^T \Sigma_p \Phi$.
It is know important to notice the the recurrence of the term $\phi(\boldsymbol{x}) \Sigma_p \phi(\boldsymbol{x})$. We define $\psi(\boldsymbol{x}) = \Sigma_p^{1/2} \phi(\boldsymbol{x})$. The *kernel trick* consists in defining $k(\boldsymbol{x}, \boldsymbol{x}') = \psi(\boldsymbol{x}) \psi(\boldsymbol{x}')$. k is known as *covariance function* or simply as *kernel function*. This substitution permit to use an arbitrary kernel function and does not require to compute the product between the features vectors which in general results to be computationally expensive.

B.2 Function-Space View

The function-space view represents an equivalent way to look at Gaussian Process. First we give a formal definition

Definition 7. *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

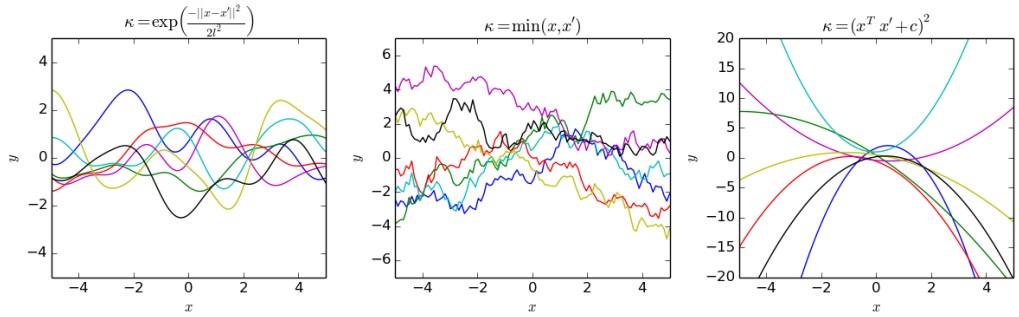

Figure B.1: *Prior samples with different kernels*

In this setting a GP is completely specified by a *mean and covariance function*. Let $f(\boldsymbol{x})$ be a GP than

$$\begin{cases} m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \\ k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))] \end{cases} \tag{B.6}$$

It is important to notice that this definition is completely in accordance with the weight-space view. Indeed our simple linear bayesian regression model can be obtained by defining:

$$\begin{cases} m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] = \phi(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}] = 0 \\ k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \Sigma_p \phi(\boldsymbol{x}') \end{cases} \tag{B.7}$$

The covariance function represents implies a distribution over the space of functions. Figure B.1 shows, for different choice of the kernel function, samples of function drawn from the prior distribution. Again the choice of the covariance function is not arbitrary and requires some educated guesses and domain specific knowledge.

Prediction, in case of noise-free observations, for new test input happens in a symmetrical way with respect the bayesian model

$$\begin{aligned} \boldsymbol{f}_* | \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{f} \sim \mathcal{N}(&K(\boldsymbol{X}_*, \boldsymbol{X}) K(\boldsymbol{X}, \boldsymbol{X})^{-1} \boldsymbol{f}, \\ &K(\boldsymbol{X}_*, \boldsymbol{X}_*) - K(\boldsymbol{X}_*, \boldsymbol{X}) K(\boldsymbol{X}, \boldsymbol{X})^{-1} K(\boldsymbol{X}_*, \boldsymbol{X}_*)) \end{aligned} \tag{B.8}$$

The previous equation can be extended also for the case of noisy observation by taking into account also the variance of the error (σ_e^2) associated with the observed data.