Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

Knowledge Transfer in Reinforcement Learning

Tesi di dottorato di:
Alessandro Lazaric

Relatore:
Prof. Andrea Bonarini
Correlatore:
Dott. Ing. Marcello Restelli
Tutore:
Prof. Marco Colombetti
Coordinatore del programma di dottorato:
Prof. Patrizio Colaneri

XX ciclo

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32 I 20133 — Milano

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

Knowledge Transfer in Reinforcement Learning

Doctoral Dissertation of:
Alessandro Lazaric

Advisor:
Prof. Andrea Bonarini
Coadvisor:
Dott. Ing. Marcello Restelli
Tutor:
Prof. Marco Colombetti
Supervisor of the Doctoral Program:
Prof. Patrizio Colaneri

XX - 2008

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32 I 20133 — Milano

To my family...

Acknowledgments

This thesis is the result of three years spent in passionate hard work, but it is also the result of a profitable joint work with valuable people here at AIRLab. First of all, my deepest gratitude goes to my advisor Andrea Bonarini for giving me the opportunity to work on such an exiting topic as Reinforcement Learning and for the encouragement to keep me focused and, at the same, for giving me the freedom to find my way in research. I am really thankful also to my co-advisor and friend Marcello, who attended me step by step in my path as a PhD student with immeasurable patience. I can't count the time he spent discussing, arguing and sharing ideas with me. I don't really know how he could put up with me for so long! A special thank goes also to my world-traveller friend Enrique, for introducing me to the fascinating world of Game Theory.
This thesis was made possible also by my very special family, which represented a reference point in good and bad times. I would like to thank my parents, Ferruccio and Gabriella, for their continued support during these long years of study and because they always gave me the freedom to choose and lead my way, and my brother, Leonardo, for his precious suggestions and with whom no words are needed to make us understood: C TT! Furthermore, I would like to thank also all my relatives, aunt Tamara, aunt Naidi, uncle Franco, and all those who certainly supported me from up above.
And finally, it is obvious that very special thanks go to all my friends: Angela, Daniele L., Daniele G., Davide D., Davide Z., Erika, Francesco, Matteo C., Matteo S., the G.M. Paolo, Pierangelo, Valentina. In all these years they have been a safe reference point to which I always entrusted in hard times.

Alessandro

Sommario

L'argomento centrale di questa tesi di dottorato è il problema del trasferimento di conoscenza e di capacità di apprendimento (*Transfer Learning* (TL)) nel paradigma di apprendimento automatico (*Machine Learning* (ML)) più generale: l'apprendimento per rinforzo (*Reinforcement Learning* (RL)). Le tecniche di ML stanno avendo una crescente diffusione in problemi di classificazione (classificazione di esami medici, stati emotivi, profilazione utente), di previsione (pattern matching, previsione di andamento di serie temporali) e di decisione (problemi di controllo di robot, problemi di gestione di portafogli finanziari). Tuttavia, uno dei maggiori limiti delle tecniche di ML è rappresentato dal fatto che per ogni problema esse richiedono una specifica taratura dei parametri e che, una volta ottenuta la soluzione al problema, esse devono essere riapplicate da zero ogni volta che il problema subisce delle variazioni anche limitate. Al fine di oltrepassare questo limite, alcuni lavori di ML si sono focalizzati sul problema di progettare algoritmi in grado di migliorare le prestazioni di apprendimento trasferendo l'esperienza accumulata risolvendo problemi simili a quello corrente. Nonostante i risultati di queste ricerche siano incoraggianti, esse si sono per lo più concentrate su problemi di apprendimento supervisionato (classificazione e previsione), mentre sono pochi i risultati in problemi di decisione. Per questo motivo, questa tesi si è focalizzata sul paradigma di RL che fornisce modelli e tecniche per affrontare una vasta gamma di problemi di decisione: dal controllo di robot autonomi (navigazione) al controllo di sistemi meccanici (ottimizzazione di controllo attivo delle sospensioni), dall'informatica di intrattenimento (computer games) al supporto per decisioni (gestione di portafogli finanziari). Tuttavia, la gran parte degli algoritmi di RL hanno complessità elevata e anche una volta ottenuta la soluzione del problema, essa deve essere ricalcolata da zero qualora sia necessario risolvere problemi simili.

In questa tesi ci si è dunque concentrati sulla definzione del problema del TL nel paradigma di RL e sono stati proposti algoritmi che consentano un riuso efficace delle soluzioni apprese, al fine di rendere più efficienti le tecniche attuali.

I principali risultati di questa tesi possono essere riassunti come segue.

I. *Sistematizzazione della letteratura di TL.* Il TL è un settore relati-

vamente recente nell'ambito del ML ed in particolare del RL. Per questo motivo, il primo obiettivo della tesi è stata una sistematizzazione ed una classificazione della letteratura ad oggi presente. Grazie a questa classificazione è stato possibile individuare quali obiettivi sono stati perseguiti fino ad ora e attraverso quali tecniche. In questo modo sono state individuate le dimensioni del problema che sono ancora inesplorate e che hanno rappresentato il punto di partenza per il resto della tesi.

II. *Formalizzazione del problema di TL e dei suoi obiettivi.* Una delle difficoltà maggiori nella definizione del problema di TL e dei suoi obiettivi nel RL risiede nel fatto che molteplici aspetti di algoritmo di RL contribuiscono in modo combinato alla prestazione finale. Per questo motivo ci si è concentrati sui recenti approcci di Batch RL che, separando la fase di acquisizione dei dati di esperienza da quella di apprendimento, hanno consentito di formalizzare in modo più rigoroso gli obiettivi del TL e come essi possono essere perseguiti.

III. *Definizione di un algoritmo di trasferimento dell'esperienza.* Nella tesi si è proposto un nuovo algoritmo di TL in grado di riutilizzare i dati raccolti da alcuni problemi "sorgente" per la soluzione di nuovi problemi. L'aspetto più critico è stato lo sviluppo di una tecnica in grado di individuare quali dei dati a disposizione fosse utile trasferire per la soluzione di un nuovo problema. L'algoritmo ha portato significativi vantaggi in termini di velocità di apprendimento rispetto ad algoritmi tradizionali di RL.

IV. *Definizione di un algoritmo per l'apprendimento multi-task.* Uno degli scenari più frequenti del TL, provede la soluzione contemporanea di un insieme di problemi (task) che condividono alcune caratteristiche strutturali. Nella tesi si è proposta l'integrazione di una tecnica utilizzata in apprendimento supervisionato per l'identificazione di questa struttura condivisa con un algoritmo di Batch RL. L'algoritmo così ottenuto è in grado, a partire da dati raccolti da tutti i problemi da risolvere, di individuare soluzioni di qualità superiore a quelle apprese utilizzando algoritmi tradizionali di RL.

La prospettiva seguita in questa tesi nell'affrontare il problema del transfer in RL apre numerose direzioni di ricerca che potranno condurre nel futuro allo sviluppo di tecniche in grado di risolvere problemi complessi in molti domini di interesse.

Summary

The main topic of this thesis is the problem of *Transfer Learning* (TL) in the most general paradigm of *Machine Learning* (ML), *Reinforcement Learning* (RL). ML techniques are gaining more and more interest as effective solutions to classification problems (e.g., clinical classification, user profiling), regression problems (e.g., pattern matching, time series forecasting), and sequential decision problems (autonomous robotics, portfolio management). Nonetheless, one of the main limit of ML techniques is that they often require long parameter-tuning of the parameters and every time the task at hand changes they need to restart learning from scratch. In order to overcome this limitation, many works focused on designing algorithms able to improve the learning performance by transferring the experience retained from the solution of tasks similar to the one at hand. Although these works obtained encouraging results, they mostly focused on supervised learning problems (classification and regression) but still very few results are available in sequential decision problems. Therefore, in this thesis we focused on the paradigm of RL that provides models and techniques to deal with a wide range of decision problems: from autonomous robotics (e.g., navigation) to automatic controls, from entertainment (e.g, computer games) to decision support systems (e.g., portfolio management). Although very general, RL algorithms suffer from the curse of dimensionality, that is, prohibitive temporal and spatial complexity in large problems. Furthermore, even when the solution is finally learned, there is not possibility to reuse it to ease the learning in similar tasks.

Therefore, in this thesis, we focused on the definition of the TL problem in RL and we proposed algorithms able to effectively reuse the knowledge retained from a set of tasks in order to improve the learning performance on similar tasks.

The main achievements of the thesis can be summarized as follows:

I. *Classification of the state of the art.* TL is a relatively novel topic in RL. Therefore, much effort has been devoted to formalize the problem of transfer and to define a classification of the works proposed so far. We focused on three different dimensions: the objectives,

the scenarios, the knowledge retained and transferred across tasks. The analysis of the state of the art under this perspective showed that most of works of transfer in RL focused on a relatively small subset of objectives and techniques. (e.g., transfer of solutions or subpolicies for the improvement of the learning speed). This analysis represented the basis for the development of the rest of the thesis.

II. *Transfer in batch RL.* The performance of RL algorithms depends on many aspects, such as, the exploration policy, the function approximator, the update rule, and so on. As a result, it is often difficult to analyze how transfer solutions affect the final learning performance and which aspects should be considered to achieve the different objectives of transfer. We proposed the batch RL framework as the most suitable perspective to analyze the problem of transfer in RL thanks to the distinction among different aspects of the learning process (i.e., sampling and learning).

III. *Transfer of samples.* We introduced a novel sample-based transfer algorithm pursuing the objective of improving the learning speed. Under the assumption that tasks are somehow related, the idea is to selectively transfer part of the samples collected in the source tasks to augment the set of samples collected in the target task used to feed the batch RL algorithm. The algorithm succeeds in identifying which samples are more convenient to transfer, thus avoiding the negative effects of transferring "wrong" samples. The results show that the transfer of samples can significantly improve the learning performance by avoiding negative transfer.

IV. *Multi-task fitted Q-iteration.* One of the most common scenarios in TL is the multi-task problem, in which a set of related tasks must be solved at the same time. This objective received little attention in works of transfer in RL so far and this is the first attempt of integration of a RL algorithm with a multi-task learning algorithm. Preliminary results show that the proposed algorithm leads to a significant improvement both in terms of reduction of the approximation error and improvement of policy performance.

The perspective followed in this thesis and the analysis of the problem of transfer in RL opens a number of research perspectives that could lead to the development of techniques able to solve complex tasks in a wide range of domains.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview of the Problem

The idea of transfer of knowledge in order to improve the performance of machine learning algorithms stems from psychology and cognitive science research. A vast number of psychological studies shows how the effectiveness of learning a task is strictly related to the knowledge retained from solving similar tasks. For instance, a person who can drive a bicycle learns to drive a motorcycle faster than a person who has never driven anything similar. The reason is that, while learning how to drive a bicycle, the human mind retains abstract knowledge about the problem of driving that can be profitably reused when facing a problem that shares some characteristics with driving a bicycle. Human beings can learn amazingly fast because they effectively *bias* the learning process towards a very limited set of solutions obtained by *transferring* the knowledge retained from solving similar tasks. Similarly, the idea of *transfer learning* is that it is possible to improve the performance of machine learning algorithms by biasing their hypothesis space towards a set of "good" hypotheses according to the knowledge retained from solving other tasks.

The general problem of transfer in machine learning, that is the retention and reuse of knowledge across tasks in order to improve a learning algorithm performance, is a challenging problem and many questions are still open. Research on transfer obtained significant successes in supervised learning problems, such as recommender systems, medical decision making, text classification, and general game playing. On the other hand, the problem of knowledge transfer in reinforcement learning received relatively little attention so far. Although the idea of transfer has been often exploited as methodology for solving complex problems (e.g., transfer of solutions in problems with increasing complexity) and recent works have proposed general solutions that obtained encouraging results, a detailed analysis of transfer in reinforcement learning is still lacking. At the same time, it is widely recognized that the possibility to perform effective transfer in decision-making problems may significantly improve the performance of reinforcement learning algorithms, thus enabling them to solve real-world complex applications.

1.2 Motivation

The motivation for knowledge transfer in reinforcement learning is to reduce the complexity of the learning process by exploiting the knowledge retained from previously solved tasks. Indeed, learning algorithms that can benefit from the transfer of knowledge across related tasks would represent a step towards real autonomous systems. In principle, traditional reinforcement learning already provides mechanisms to learn solutions for any task without the need of human supervision. Nonetheless, the time needed for learning a nearly optimal solution is often prohibitive in real-world problems, even if similar tasks have been already solved (e.g., learning to achieve a location in a room can benefit from the solution of navigation problems in similar rooms). In fact, there is no possibility to transfer knowledge across related tasks and each task must be solved from scratch. Therefore, algorithms able to transfer solutions and to bias a learning algorithm towards a reduced set of solutions could dramatically reduce the learning complexity on a wide range of applications. Furthermore, when multiple related tasks must be solved, learning algorithms simply learn each task independently, thus wasting the information about the relationships among the tasks (e.g., learning an optimal controller for an hybrid car on a given set of cycles). On the other hand, the definition of a joint learning problem on all the tasks at the same time, could significantly improve the generalization performance by exploiting the underlying structures shared across the tasks.

1.3 Research Approach

Instead of focusing only on the formalization of novel algorithms for transfer, in this thesis we follow a synthetic approach with the aim at framing transfer reinforcement learning approaches into a general classification. Furthermore, unlike most of the research in transfer in reinforcement learning, we discuss possible relationships between transfer in supervised learning and in reinforcement learning. In fact, although reinforcement learning has some specific characteristics, we believe that it is possible to derive useful inspiration from supervised learning solutions and, in some cases, to integrate algorithms and techniques with the reinforcement learning paradigm. In particular, we follow two main perspectives, usually adopted in supervised literature: *inductive transfer*, *multi-task learning*. In inductive transfer the idea is to retain knowledge from a set of source tasks and to reuse it so as to reduce the learning

complexity on new target tasks. On the other hand, the multi-task perspective does not distinguish between source and target tasks: given a finite set of related tasks the objective is to exploit the information about all the tasks at the same time in order to improve the generalization performance on each of them.

Following the analysis of current research on transfer in RL, we derive directions of investigation that can represent the basis for the development of novel algorithms for transfer. The objective is to contribute to the identification of promising perspectives for the study of transfer in RL and to propose algorithms that investigate solutions and approaches that received little attention so far.

1.4 Open Questions in Transfer Reinforcement Learning

This thesis investigates the problem of transfer in reinforcement learning. Following the approach summarized in the previous section, we focus on some open questions:

- **How can the problem of transfer in reinforcement learning be formalized?** Although the transfer problem is gaining more and more interest in reinforcement learning community, a formal definition of the objectives and of the applications is still lacking. As a result, it is often difficult to compare the proposed approaches not only in terms of their results but also in terms of the objectives pursued by the algorithms and the scenarios considered in the experiments. In this thesis, we propose a first classification of the transfer approaches to reinforcement learning on the basis of their *objectives*, the *knowledge* they transfer and the *scenarios* they consider. Furthermore, in reinforcement learning it is often difficult to identify how the transferred elements impact on the final performance of the algorithm. In this thesis, we propose batch reinforcement learning as a framework in which the distinction between sampling and learning phases allows to distinguish more precisely among the effects of transfer.

- **What kind of knowledge is convenient to transfer?** A key factor for transfer learning algorithms is to identify what kind of knowledge is more convenient to retain and transfer. Most of the transfer reinforcement learning works focused on transfer of solutions (e.g., value functions, policies) across tasks, while little atten-

tion has been devoted to the analysis of the effect of transferring the most simple form of experience collected during the learning process, that is, the trajectory samples. Another relevant objective that has not been deeply investigated so far, is the improvement of the generalization through the adaptation of the feature space. In this thesis, we propose two different techniques to deal with these two issues.

- **When is it convenient to transfer knowledge?** It is widely recognized that the transfer of knowledge is effective only when tasks are related, but only few works faced the problem of recognizing the tasks in which transfer is likely to improve the learning performance and those in which it is preferable to avoid any kind of transfer. In this thesis, in Chapter 4 we propose a method for measuring the similarity between samples of different tasks so as to choose when to transfer and which part of the retained knowledge is worth transferring. Furthermore, in Chapter 5, we show how the feature space of a function approximator can be adapted so as to discover the similarities among tasks and thus making it possible to learn their solutions through a multi-task learning algorithm.

- **How can reinforcement learning algorithms benefit from transfer results in supervised learning literature?** The study of transfer in supervised learning led to interesting theoretical results and effective algorithms in many applications, but only few works served as inspiration for the study of transfer in reinforcement learning. In this thesis, we focus on the algorithmic integration of batch reinforcement learning with the multi-task learning perspective in order to improve the generalization performance on a finite set of tasks.

1.5 Overview of the Dissertation

The rest of the thesis is organized as follows:

- **Chapter 2**. This chapter provides a brief introduction to reinforcement learning. At first, we review the concept of Markov Decision Process, the definition of (optimal) value function and policy. After the introduction of dynamic programming algorithms, we briefly review temporal difference algorithms, such as TD(0) and Q-learning. Then, we discuss the issues arising from the introduction of function approximation and hierarchical task decompo-

sition in the reinforcement learning paradigm. Finally, we review two main batch reinforcement learning algorithms, Least Squares Policy Iteration and Fitted Q-iteration, that will be at the basis of the transfer algorithms proposed in the rest of the thesis. The reader who is already familiar with these topics can well skip this chapter.

- **Chapter 3**. In this chapter, we review the main approaches of transfer in supervised learning literature and we introduce the distinction between the inductive transfer and multi-task learning perspectives. Furthermore, we propose three dimensions for the analysis of the main approaches to the problem of transfer in reinforcement learning. Finally, the batch reinforcement learning framework is proposed as a suitable model for the implementation of transfer in reinforcement learning.

- **Chapter 4**. This chapter introduces an algorithm for the improvement of learning speed through the transfer of samples from different tasks. In particular, we propose a method for the identification of which source tasks are more convenient to transfer from and we introduce a mechanism for choosing which samples are more likely to be similar with those of the target task.

- **Chapter 5**. In this chapter, we investigate the possibility to integrate multi-task learning algorithms with batch reinforcement learning algorithms. We extend fitted Q-iteration in order to improve the generalization performance of the approximation of the action-value function in a finite set of tasks. In particular, we rely on a recent algorithm of multi-task feature learning for learning the parameters of a linear function approximator together with the feature space.

- **Chapter 6**. In this chapter, we draw the conclusions of the thesis, we summarize the main contributions and we discuss future directions of investigation.

2 Reinforcement Learning

In this chapter, we introduce the basic formulation of reinforcement learning and we review function approximation, hierarchical approaches and batch reinforcement learning. The goal of this chapter is to provide an introduction to the elements that are relevant to enable transfer in Reinforcement Learning. For a more complete overview of RL we refer the reader to [134, 17].

2.1 Introduction

Reinforcement Learning (RL) is a Machine Learning (ML) paradigm resulting from the combination of many different research directions in statistics, computer science, neuroscience, psychology, and optimal control theory. From early 90s, RL has become the standard framework in the artificial intelligence community for studying how agents learn and plan in uncertain environments. In its simplest form, the basic idea of RL is that if an action is followed by an "improvement" in the state of affairs, then the tendency to produce that action is strengthened (i.e., reinforced); otherwise, the tendency to produce that action is weakened [16].

RL can be briefly defined as [134]:

> *Reinforcement learning is learning what to do –how to map situations to actions– so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics –trial-and-error search and delayed reward– are the two most important distinguishing features of reinforcement learning.*

The standard RL interaction model between the agent and the environment is depicted in Figure 2.1. At each step of interaction, the agent

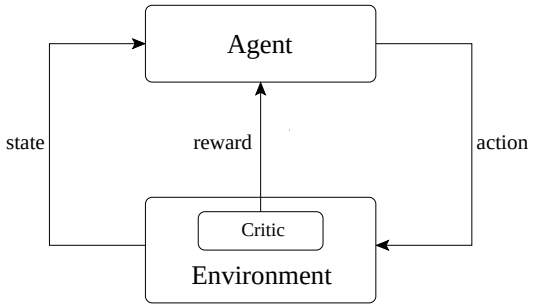


Figure 2.1: The standard reinforcement learning model.

receives an input that describes the current *state* s of the environment and then chooses an *action* a. The action changes the state according to the dynamics of the environment, and the goodness of this state transition is communicated to the agent by means of a scalar *reinforcement signal* r. The component of the environment that provides this signal is usually called the *critic*.

The goal of the agent is to find a *policy* π, that is, a mapping from states to actions, so as to maximize its *expected return*, defined as some specific function of the reward sequence. In *non-episodic* tasks, it is common to measure the *return* of a sequence of actions, as the sum of the expected *discounted reward*:

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma \in [0, 1]$ is a *discount factor*. As γ tends to 0, the agent gets more and more myopic and future rewards are considered not significant. When, instead, γ tends to 1, the agent gets more and more farsighted and takes into account future rewards.

2.2 Markov Decision Processes

In RL, the interaction between the agent and the environment is commonly modeled as a discrete time Markov Decision Process (MDP) [20].

Definition 2.1 *A MDP is a tuple* $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$*, where:*

- $\mathcal{S}$ *is the state space*

- $\mathcal{A}$ *is a the action space*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ *is the transition model that assigns to each state-action pair a probability distribution over* $\mathcal{S}$*,*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathbb{R})$ *is the reward function that assigns to each state-action pair a probability distribution over* $\mathbb{R}$*.*

A process is Markov if it satisfies the *Markov Property*:

$$P\left(s_{t+1}, r_{t+1} \mid s_t, a_t\right) = P\left(s_{t+1}, r_{t+1} \mid s_t, a_t, r_t, ..., r_1, s_0, a_0\right).$$

If the Markov Property holds, what happens at time $t+1$ depends only on what happened at the previous time step (s_t, a_t), and not on the history of the system $(s_t, a_t, r_t, ..., r_1, s_0, a_0)$. The dynamics of the system is thus *one-step*, and this makes it possible to predict the next state s_{t+1} and the next reward r_{t+1} given only the current state-action pair (s_t, a_t).

At each time step, the agent chooses an action so as to maximize the reward obtained from the environment. The choice of the action depends on the current *policy* of the agent. The policy is defined as a function π that maps each state $s \in \mathcal{S}$ to a probability distribution over actions $a \in \mathcal{A}$:

$$\pi : \mathcal{S} \to \Pi(\mathcal{A}). \tag{2.1}$$

Given a generic policy π and a state s, in order to maximize its expected reward the learner must be able to estimate how good it is to follow π in s. This information is given by the *state-value function* $V^\pi(s)$:

$$V^\pi(s) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right],$$

where E_π denotes the expectation given that the agent follows policy π.

It is also possible to define the utility of taking an action a in state s and following policy π thereafter. This is given by the *action-value function* $Q^\pi(s, a)$:

$$Q^\pi(s, a) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right].$$

A key property of value functions is that they satisfy particular recursive relationships, known as *Bellman equations* [20]. The Bellman equation for $V^\pi(s)$ is the following:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}(s'|s, a) \left[R(s, a) + \gamma V^\pi(s') \right], \tag{2.2}$$

where $R(s,a)$ is the expected reward in (s,a), $(R(s,a) = E[\mathcal{R}(\cdot|s,a)])$. This equation expresses a relationship between the value of a state s and those of its possible successor states, weighted by their probability of occurring.

Similarly, the Bellman equation for $Q^\pi(s,a)$ is:

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a) + \gamma V^\pi(s') \right]. \tag{2.3}$$

$V^\pi(s)$ and $Q^\pi(s,a)$ are the *unique* solutions of their respective Bellman equations. Alternatively, it can be shown that Q^π (similarly V^π) is the *fixed point* of the Bellman operator T_π:

$$(T_\pi Q)(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a) + \gamma V^\pi(s') \right]. \tag{2.4}$$

The goal of a RL agent is to maximize the expected sum of discounted rewards, that is, to learn an optimal policy π^* that leads to the maximization of the value functions in each state. It can be proved that in any MDP there exists at least one deterministic optimal policy π^* such that:

$$V^* \geq V^\pi, \ \ \forall \pi \neq \pi^*.$$

The optimal value functions can be computed through specific Bellman equations:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a) + \gamma V^*(s') \right], \tag{2.5}$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a) + \gamma V^*(s') \right]. \tag{2.6}$$

Given the optimal action-value function, the optimal (deterministic) policy π^* can be simply obtained as the action that maximizes the action-value function in each state:

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = arg\max Q(s,a) \\ 0 & \text{otherwise} \end{cases}$$

As we see in the following, most of traditional RL algorithms are built on Bellman equations and learn the optimal policy by estimating the optimal value functions.

Algorithm 1 Iterative policy evaluation algorithm

Input: $\mathcal{P}(\cdot|s,a)$, $\mathcal{R}(\cdot|s,a)$, γ, π
Parameters: ϵ
Output: V^π
for all $s \in \mathcal{S}$ **do**
  Initialize $V_0(s)$
end for
repeat
  $\Delta \leftarrow 0$
  for all $s \in \mathcal{S}$ **do**
    $V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}(s'|s,a)[R(s,a) + \gamma V_k(s')]$, $R(s,a) = E[\mathcal{R}(\cdot,s,a)]$
    $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$
  end for
until $\Delta < \epsilon$

2.3 Dynamic Programming

Dynamic Programming (DP) [20] is a set of techniques for computing the optimal policy in MDPs. DP algorithms use value functions to implicitly perform a search for good policies. All DP algorithms require the transition model and the reward function as input.

2.3.1 Policy Iteration

The *policy iteration* algorithm, starting from a generic policy π_0, finds an optimal policy π^* by iterating steps of *evaluation* and *improvement* of the current policy:

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow ... \rightarrow \pi^* \rightarrow V^*, \tag{2.7}$$

where $(\pi_i \rightarrow V^{\pi_i})$ and $(V^{\pi_i} \rightarrow \pi_{i+1})$ are the i-th policy evaluation and policy improvement step, respectively.

During the i-th evaluation step, the algorithm computes the state-value function V^{π_i} for the current policy π_i by turning the Bellman equation (2.2) into an iterative update rule

$$V_{k+1}(s) = \sum_a \pi_i(a|s) \sum_{s'} \mathcal{P}(s'|s,a)[R(s,a) + \gamma V_k(s')], \quad s \in \mathcal{S}.$$

At each step of the iteration, the algorithm replaces the old value of every state s with a new value obtained from the old values of the successor states of s and the expected immediate rewards, weighted by the

Algorithm 2 Policy iteration algorithm

Input: $\mathcal{P}(\cdot|s,a)$, $\mathcal{R}(\cdot|s,a)$, γ
Output: V^*, π^*
Choose an arbitrary policy π_i
repeat
  Perform policy evaluation on π_i
  Improve π_i at each state:
  $\pi_{i+1} = \arg\max_a \sum_{s'} \mathcal{P}(s'|s,a)[R(s,a) + \gamma V^{\pi_i}(s')]$, $R(s,a) = E[\mathcal{R}(\cdot|s,a)]$
until $\pi_{i+1} = \pi_i$

transition probabilities $\mathcal{P}(s'|s,a)$. The sequence $\{V_k\}$ is guaranteed to converge to the true V^π as $k \to \infty$ [134]. The complete algorithm for the evaluation step is detailed by Algorithm 1.

During the i-th improvement step, the algorithm improves π_i generating a new policy π_{i+1} such that $V^{\pi_{i+1}} \geq V^{\pi_i}$. Let us consider a state $s \in \mathcal{S}$, and let us suppose that π_i is such that in s action a is chosen. In order to determine if another action a' is better than a in s we compute the action-value function $Q^\pi(s,a')$

$$Q^{\pi_i}(s,a') = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a) + \gamma V^{\pi_i}(s')\right],$$

which estimates the utility of taking action a' in s and following policy π_i thereafter. If $Q^{\pi_i}(s,a') > V^{\pi_i}(s)$, the new policy π_{i+1} is an improvement over the original policy π_i. Extending this idea to all states, we obtain the algorithm for the policy improvement step, which finds a new greedy policy π_{i+1} such that:

$$\begin{aligned} \pi_{i+1}(s) &= \arg\max_a Q^{\pi_i}(s,a) \\ &= \arg\max_a E_{\pi_i}[r_{t+1} + \gamma V^{\pi_i}(s_{t+1})|s_t = s, a_t = a] \\ &= \arg\max_a \sum_{s'} \mathcal{P}(s'|s,a)[R(s,a) + \gamma V^{\pi_i}(s')]. \end{aligned} \quad (2.8)$$

This step is guaranteed to produce a new policy π_{i+1} that is at least as good as the original policy π_i [134]. Since in a finite MDP there are a finite number of policies, and the sequence of policies improves at each step, the policy iteration algorithm converges to the optimal policy in a finite number of iterations. Algorithm 2 shows the pseudo-code for policy iteration.

Algorithm 3 The on-line TD(0) learning algorithm

Input: $\mathcal{S}$, $\mathcal{A}$, γ
Parameters: α, π
Output: V^*, π^*
Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
for all episode **do**
  Initialize s_t
  while s_t is terminal **do**
    $a_t \leftarrow \pi(\cdot|s_t)$
    Take action a_t; observe r_{t+1} and s_{t+1}
    $V(s_t) \leftarrow V(s_t) + \alpha_{t+1}\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$
    $t \leftarrow t + 1$
  end while
end for

2.3.2 Value Iteration

The *value iteration* [101] algorithm is obtained by truncating the policy evaluation step of policy iteration just after one sweep in the state set. The reason for this kind of approach is that the policy evaluation step may require multiple sweeps in the state set before converging, thus slowing down the policy iteration algorithm. Moreover, it can be proved [24] that by truncating policy evaluation, the convergence properties of policy iteration are preserved. The update rule for value iteration becomes the following:

$$\begin{aligned} V_{k+1}(s) &= \max_a E[r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} \mathcal{P}(s'|s,a)[R(s,a) + \gamma V_k(s')], \end{aligned} \tag{2.9}$$

It can be proved that this iterative process converges to the optimal value function: $V_k \rightarrow V^*$ as $k \rightarrow \infty$. Like policy iteration, value iteration terminates when the value function improvement is small in a sweep.

2.4 Reinforcement Learning

The main drawback of DP techniques is the assumption that both the transition model and the reward function are available. Since in most RL problems the learner does not have any built-in knowledge of the task to be solved, the use of these algorithms is often impractical. This is the reason because most of the RL algorithms learn the optimal policy

Algorithm 4 The Q-learning algorithm

Input: $\mathcal{S}$, $\mathcal{A}$, γ
Parameters: α, π
Output: V^*, π^*
Initialize $Q(s, a)$ arbitrarily
for all episode **do**
  Initialize s_t
  while s_t is not terminal **do**
    $a_t \leftarrow \pi(\cdot|s_t)$
    Take action a_t; observe r_{t+1} and s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
    $t \leftarrow t + 1$
  end while
end for

solely through the information collected by direct interaction with the environment.

2.4.1 Temporal Difference Learning

Temporal Difference (TD) learning techniques learn directly from experience, without recurring to any model of the environment.

In the simplest TD method, known as $TD(0)$ [131] (Algorithm 3), at each time step t, the agent is in state s_t and takes an action a_t according to its current policy π and the value function estimate in s_t is updated as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha_t[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]. \tag{2.10}$$

where α_t is a learning rate, s_{t+1} is the state achieved by taking action a_t in s_t and r_{t+1} is the reward returned by the critic. Since TD(0) bases its update in part on the existing estimate, it is said a *bootstrapping* method. The algorithm can be shown to converge upon V^π as $t \to \infty$, provided that the rewards are bounded, the process is Markov and the learning rate is declined under the Robbins-Monro conditions:

$$\sum_{t=1}^{\infty} \alpha_t = \infty; \ \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{2.11}$$

Since TD(0) provides an estimation of the value function V^π for a given policy π, it cannot be used in control tasks in which the goal is to learn the optimal policy π^*. In this case, it is necessary to compute an

estimation of the optimal action-value function $Q^*(s,a)$. In Q-learning [159] (Algorithm 4), at each time step t, the agent takes an action a_t in state s_t and the corresponding action-value is update as:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha_t[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)], \quad (2.12)$$

where $r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$ is the target return estimate.

Since the update rule of Q-learning does not take into account the policy used to collect rewards, the algorithm is *off-policy*. It can be proved [160] that the Q-learning algorithm converges to the optimal action-value function $Q^*(s,a)$, under the assumption that the rewards are bounded, the agent selects actions so as to visit every (s,a) pair infinitely often and the learning rate α satisfies the Robbins-Monro conditions.

2.4.2 Exploration-Exploitation Dilemma

Unlike supervised learning, in RL, there is no clear distinction between training and testing phases. In fact, the agent directly collects the samples (i.e., rewards obtained by taking actions in different states) used to adjust its estimation of the value function. Therefore, a key factor for achieving learning is the strategy adopted by the agent to explore the environment. Since the goal is to maximize the rewards, the exploration strategy of the agent should balance the performance obtained by taking greedy actions (i.e., the best action in each state according to the current value function estimation) and the policy improvement that can be obtained through the information gathered by taking exploratory actions. This problem is usually referred to as the *exploration-exploitation dilemma*.

Exploration-exploitation strategies can be roughly divided into two classes [147]: *undirect* and *direct* methods. The former are model-free methods that determine the policy followed by the agent on the basis of the current estimation of the action-value function. ϵ-greedy and Boltzmann strategies [134] are among the most used exploration strategy. The ϵ-greedy strategy is defined as:

$$\pi(a|s) = \begin{cases} 1-\epsilon & \text{if } a = arg\max Q(s,a) \\ \frac{\epsilon}{n-1} & \text{otherwise} \end{cases} \quad (2.13)$$

that is, it performs the greedy action with probability $1-\epsilon$ and with probability ϵ any of the other $n-1$ actions at random.

On the other hand, direct methods usually compute an estimation of the reliability of the value function estimates and/or of the transition

model and reward function approximations. This information is usually used to direct the exploration in regions of the state-action space in which the approximation of the optimal action-value function is poor and the expected advantage of exploration is high [103, 147, 163].

Unfortunately, most of the previous exploration-exploitation strategies are heuristic and they do not provide any guarantee about the learning performance obtained following those policies. Actually, the solution of the problem of optimal exploration [92] is infeasible. Therefore, in recent years many works focused on providing complexity bounds that define the number of samples needed to learn a nearly-optimal policy. The theoretical background of these research directions is the *probably approximately correct* (PAC) framework, that defines the number of samples n such that:

$$P\left(|V^k - V^*| < \varepsilon\right) > 1 - \delta \tag{2.14}$$

that is, the complexity of learning an ε-optimal solution with a probability at least of $1-\delta$. The E^3 [66] algorithm and the $RMAX$ algorithm [30] are the first RL algorithms guaranteed to learn a nearly-optimal policy with polynomial complexity, that is, with a number of samples polynomial in the characteristic parameters of the task (e.g., number of states, number of actions, discount factor). In [126, 127] an empirical and theoretical analysis of the model-based interval estimation exploration strategy is reported. Complexity bounds when the learning algorithm has direct access to a restart distribution are derived in [63, 62]. Finally, recent works [12] framed the problem of exploration-exploitation as a regret minimization problem.

2.5 Function Approximation in Reinforcement Learning

In the previous algorithms, the action-value function is represented as a look-up table that stores a distinct value for each state-action pair. While this approach have strong theoretical foundations and is viable in many applications, it has severe limitations when applied to problems characterized by large (or continuous) state and action spaces. In this case, the amount of memory needed to store a look-up table increases exponentially with the dimensions of the problem and the number of episodes needed to learn the optimal policy makes this model unsuitable for many real-life applications (e.g., robotics). This issue is commonly referred to as the *curse of dimensionality* [134]. Several approaches have

been proposed to overcome this limitation by combining function approximation techniques (e.g., neural networks, radial basis functions, fuzzy sets) to RL algorithms so as to learn an accurate approximation of the action-value function with a limited number of parameters independently from the dimensions of the problem.

The general form of an approximation for the action-value function is:

$$\widehat{Q}(s,a) = f(\phi(s,a),\vartheta), \tag{2.15}$$

where f is the output function, $\phi(\cdot) = [\phi_1(\cdot)\dots\phi_i(\cdot)\dots\phi_N(\cdot)]$ is a vector of features (or basis functions), where feature $\phi_i(\cdot) : \mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$ maps each state-action pair to the corresponding feature value, and $\vartheta = [\vartheta_1\dots\vartheta_i\dots\vartheta_M]$ is the vector of parameters that are adjusted during the learning process. While features ϕ determine the space $\mathcal{Q}$ of functions that can be represented, the learning parameters ϑ define the specific function $\widehat{Q}\in\mathcal{Q}$ actually learned by the agent.

The problem of learning a function approximator from a set of input-output samples has been extensively studied in supervised learning. However, in RL function approximation is harder to implement than in supervised learning, because the training data are not given in advance by a trainer, but they are in part determined by the output of the learned function. In the following, we describe how the value function estimation is updated when learning with function approximation.

2.5.1 Update Rules

During the learning process, the agent receives samples of the target action-value function and the parameters of the function approximator are updated so as to reduce the approximation error. Most of the function approximators update the parameters according to the online gradient descent update rule:

$$\vartheta_i \leftarrow \vartheta_i - \alpha\frac{\partial\mathcal{E}}{\partial\vartheta_i},$$

where $\mathcal{E}$ is the approximation error and α is a learning rate.

Least Mean Square

The most common approximation error used in RL algorithms is the Least Mean Square (LMS) error, that is the squared 2-norm of the difference between the optimal action-value function Q^* and its approximation

$\widehat{Q}$, averaged over the state-action space:

$$\mathcal{E}_{LMS} = \frac{1}{2}\|Q^* - \widehat{Q}\|_2^2 = \frac{1}{2}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left(Q^*(s,a) - \widehat{Q}(s,a)\right)^2.$$

In this case, parameters are updated with the following rule:

$$\vartheta_i \leftarrow \vartheta_i + \alpha(Q^*(s,a) - \widehat{Q}(s,a))\frac{\partial \widehat{Q}(s,a)}{\partial \vartheta_i}.$$

In RL, the optimal action-value function is not available during the learning process. Therefore, $Q^*(s,a)$ is substituted by its estimation:

$$\vartheta_i \leftarrow \vartheta_i + \alpha\left(r + \gamma \max_{a'\in\mathcal{A}} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)\frac{\partial \widehat{Q}(s,a)}{\partial \vartheta_i}. \tag{2.16}$$

Although this update rule (which minimizes the LMS error) has been successfully used in conjunction with many function approximators and obtained valuable results in relevant applications (e.g., [146, 133]), several studies [28, 13, 54, 154] showed that, even in very simple problems, it may lead the approximator to unpredictable results and, in some cases, to divergence.

Bellman Residual

The more straightforward way to guarantee convergence in RL with function approximation is to derive the update rule incorporating the formulation of the estimation of the action-value function directly in the error function, thus obtaining the *mean square Bellman residual*, that is the squared 2-norm of the difference between the approximation and approximation obtained by applying the Bellman operator T:

$$\mathcal{E}_{Residual} = \frac{1}{2}\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left(r + \gamma \max_{a'\in\mathcal{A}} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)^2.$$

Starting from this error, Baird [13] derived the residual gradient update rule:

$$\vartheta_i \leftarrow \vartheta_i + \alpha\left(r + \gamma \max_{a'\in\mathcal{A}} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)\left(\frac{\partial \widehat{Q}(s,a)}{\partial \vartheta_i} - \frac{\partial \gamma \widehat{Q}(s'',a)}{\partial \vartheta_i}\right). \tag{2.17}$$

where s' and s'' are two independent samples drawn from the distribution $\mathcal{P}(\cdot|s,a)$.

2.5.2 Linear Function Approximation

The most common class of function approximator adopted in RL is the linear function approximation:

$$\widehat{Q}(s,a) = \phi(s,a)\vartheta = \sum_{i=1}^{M} \phi_i(s,a)\vartheta_i,$$

where M is the number of features, equal to the number of learning parameters. With linear function approximators, the least mean square update rule becomes

$$\vartheta_i \leftarrow \vartheta_i + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s',a') - \widehat{Q}(s,a) \right) \phi_i(s), \tag{2.18}$$

while the residual gradient update rule becomes:

$$\vartheta_i \leftarrow \vartheta_i + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s',a') - \widehat{Q}(s,a) \right) \left(\phi_i(s) - \gamma \phi_i(s'') \right). \tag{2.19}$$

The most simple example of linear function approximation is state aggregation. The idea of grouping states of a MDP has been adopted—more or less explicitly—in many RL algorithms. In particular, state aggregation is commonly used in domains with continuous state space, in which each state variable is discretized in a finite number of intervals [80, 76]. More sophisticated solutions use *multiple partitions* of the state space, so that each state is covered by a set of macrostates, and the value function is approximated by a linear combination of the values stored in each macrostate. In this category, there are several learning algorithms [35, 2, 94, 170] based on the *multigrid* approach [31] in which multiple partitions, with different resolutions over the state space, are used to accelerate the learning process by exploiting the generalization induced by coarser aggregations. Soft state aggregation [119] adopts a discretization of the state variables into overlapping clusters that are combined together according to their activation degree to compute the action-value estimation.

One of the most commonly used linear function approximation is *CMAC* (or *tile coding*) [1, 133], a sparse coarse-coded function approximator in which each aggregation of states corresponds to a binary feature that is combined with the others by a linear mapping function. An adaptive version of CMAC is proposed in [112], where, given a fixed global resolution, the effect of using a different number of tilings is studied. Furthermore, they proposed a heuristic online criterion that changes the

generalization breadth according to a reliability index based on backup errors. A more sophisticated version of CMAC uses Radial Basis Functions (RBFs), a generalization of coarse coding to continuous-valued features [32, 99]. Each feature is no longer associated to a binary value, but it can take any value in the interval $[0, 1]$; for instance a Gaussian function:

$$\phi_i(s) = e^{-\frac{\|s-c_i\|^2}{2\sigma_i^2}}, \tag{2.20}$$

where $\|s-c_i\|$ is the distance between the state, s, and the feature's center state, c_i, while σ_i is the feature's width. The primary advantage of RBFs over binary features is that they produce approximate functions that are smooth and differentiable. The main drawbacks of RBF networks reside in their great computational complexity and in the fact that they often require a lot of manual tuning before learning becomes robust and efficient.

The main difficulty with linear function approximation is the definition of the feature space ϕ. In fact, the feature space ϕ constrains the space $\mathcal{Q}$ of the action-value functions that can be represented. For instance, CMAC constrains the action-value function to be piece-wise constant and the choice of the number of tiles and tilings defines the resolution of the approximation. Unfortunately, it is often difficult to choose a suitable feature space for each task in advance. As a result, long manual tuning is often necessary to achieve good learning performance. Therefore, recent works [78, 67, 95] tried to develop feature extraction techniques able to adapt the feature space according to the characteristics of the problem at hand.

2.5.3 Theoretical Issues

Unlike function approximation in supervised learning, in RL the optimal action-value function is not available and its estimation is incrementally build on the basis of rewards and of previous estimations of the function (i.e., bootstrapping). This characteristic is particularly critical for the stability of the function approximation process and for the final performance of the learning process. In fact, in [28, 13, 53] simple examples are reported in which function approximation used in conjunction with LMS error obtains very poor performance and even diverges. The Bellman residual update introduced in [13] avoids the problem of loss of stability of function approximation but it has the main drawback that is needs a generative model from which trajectory samples are extracted. In [54] the use of averagers (i.e., approximators that are guaranteed to

be a non-expansion when applied to the Bellman operator) is proved to avoid divergence. Other works with averagers [137, 162] also showed that they can achieve better results than LMS-based approximators.

Another critical aspect for the stability of function approximator is the sampling policy (i.e., exploration strategy) used to collect samples. As shown in [53], current estimation of the action value function affects the sampling policy and the way samples are collected influence the updates to the action value function. This loop between sampling and learning can result in a non-convergent learning process. Therefore, it is often difficult to analyze the performance of an approximator independently from the exploration strategy and viceversa. In [154] convergence is proved for TD learning algorithms when a fixed single trajectory is used, while in [96] convergence is guaranteed for Q-learning and SARSA only when the policy update is continuous.

2.6 Hierarchical Reinforcement Learning

Together with the use of function approximation, task decomposition is usually adopted in RL as a mean to face the problem of the curse of dimensionality. The idea is that each task can be decomposed in a hierarchy of more simple subtasks. The solution of the overall task is the result of the composition of the solutions of each subtask. If the decomposition is effective, the learning time is significantly shortened. In the following we briefly review only MAX-Q value function decomposition, the option framework and subgoal discovery techniques. Indeed, these are the most common frameworks of HRL used in transfer learning. A more complete review of Hierarchical Reinforcement Learning (HRL) is available in [17].

2.6.1 MAXQ Value Function Decomposition

The MAXQ decomposition [38] is one of the most complete models of HRL. The basic assumption is that the designer has enough prior knowledge about the task to identify useful subgoals and define subtasks for each subgoal. The idea is that the optimal value function can be decomposed into an additive combination of a hierarchy of value functions. The MAXQ algorithm starts with a given decomposition of a task T (i.e., an MDP) into a hierarchy of subtasks $\{T_0, T_1, \ldots, T_n\}$ where T_0 is the root subtask. Each subtask T_i is defined by a policy μ_i, which can either execute primitive actions or other subtasks, a termination condition, and a pseudo-reward function that returns reward according to the subgoal

of T_i. The overall policy π is obtained by hierarchically executing the sub-policies μ_i. A value function can be defined for each subtask T_i by introducing a Semi-MDP (SMDP) model [100], thus taking into account that the execution of a subtask can take more than one time step to finish. As a result, it is possible to define Bellman equations for the value functions at each level of the hierarchy and thus, the hierarchical value function corresponding to policy π.

2.6.2 Options Framework

A common solution of HRL is the temporal abstraction of the MDP through the augmentation of the action space with temporally extended actions, that is macro-actions that perform a sequence of primitive actions and that terminate only when a specific condition is met. Usually, the termination condition is limited to the achievement of a particular state that coincides with one of the subgoals determined by the task decomposition. The option framework [136] is the most complete and theoretically sound model for temporally extended actions. An option o is defined by the tuple $\langle \mu, I, \rho \rangle$, where μ is a policy defined on actions in $\mathcal{A}$ (i.e., set of primitive actions), $I \subseteq \mathcal{S}$ is the set of states where the option can be selected, and ρ is a probability distribution over $\mathcal{S}$ that represents the probability of the option to terminate in a state s. The set of primitive actions of the MDP is augmented by the introduction of a set of options, thus obtaining the option space $\mathcal{O}$. Formally, the introduction of the options leads to the definition of a SMDP. The value function definition can be changed in order to take into account the options as follows

$$V^{\pi}(s) = \sum_{o \in \mathcal{O}} \pi(o|s) \sum_{s'} \mathcal{P}(s'|s,o) \left[R(s,o) + \gamma V^{\pi}(s') \right], \tag{2.21}$$

where $R(s,o)$ is the expected reward accumulated by following the policy of option o until it terminates and $\mathcal{P}(\cdot|s,o)$ is the transition model defined as a probability distribution over $\mathcal{S}$

$$\mathcal{P}(s'|s,o) = \sum_{k=1}^{\infty} P\left(s', k\right) \gamma^k,$$

where $P\left(s', k\right)$ is the probability of option o to terminate in s' after k steps. Starting from the Bellman equations for the value functions, it is possible also to extend Q-learning to SMDP Q-learning as:

$$Q(s_t, o_t) = Q(s_t, o_t) + \alpha[r_{t+1} + \gamma \max_{o} Q(s_{t+1}, o) - Q(s_t, o_t)], \tag{2.22}$$

where r_{t+1} is the reward accumulated until the option o_t terminates. There is a wide empirical evidence that if the set of options available solves the subgoals of the overall task, then the convergence time is significantly reduced.

2.6.3 Subgoal Discovery

One of the main drawbacks of HRL is that the task decomposition should be done by hand on the basis of prior knowledge about the task. Furthermore, if the task decomposition is not suitable for the task, then the learning performance can even get worse with respect to learning without task decomposition. Therefore, many works focused on the development of techniques able to automatically discover subgoals, so as to design an effective hierarchy of subtasks specific for the task at hand.

In [81] a statistical approach to the identification of bottlenecks is proposed. States often traversed by successful trajectories (i.e., trajectories that achieve the goal) are set as termination states for options that are incrementally defined during the learning process. In [83, 91] a topological analysis of partial models of the environment is used to identify states that are more likely to be relevant for the achievement of the goal of the task at hand. Once identified, these states are used to learn new options. The HEXQ algorithm [57] builds a MAX-Q hierarchical decomposition of the task by identifying *exit* states, that is, passage states that connect different regions of the state space. Finally, intrinsically motivated approaches [92, 118, 26] frame the problem of subgoal discovery into a psychologically founded framework in which the identification of subgoals and the development of new skills is directly driven from an intrinsic motivation of the agent.

2.7 Batch Reinforcement Learning

One of the main limit of RL when applied to real-world problems is the large amount of experience needed before a nearly-optimal solution is achieved. The main reason for this drawback is an inefficient use of the experience. For instance, in model-free algorithms, such Q-learning, when the agent achieves the goal only the previous state-action pair is updated and this information takes much time before being propagated back to initial states. On the other hand, model-based algorithms (e.g., prioritized sweeping [87]) are computationally expensive and need a large amount of data before the model approximation becomes reliable.

In order to overcome these drawbacks, batch approaches have been proposed. The main idea is to distinguish between the exploration strategy that collects samples from the task at hand, and the learning algorithm that, on the basis of the samples, computes the approximation of the action value function. In the following, after a brief introduction to early works on batch RL algorithms, we focus on two main recent algorithms: LSPI and fitted Q-iteration.

2.7.1 Early Works in Batch Reinforcement Learning

The idea of effectively using the experience collected through direct interaction with the environment has been exploited in many works in RL.

Monte Carlo methods [121] represent the first example of the batch use of experience. Given a policy π, the idea is to estimate the action value function in a state-action pair by recording all the rewards collected along one trajectory obtained following π and by averaging the corresponding actual return over different attempts. The update of the action value function is performed in a batch mode when the episode finishes.

The experience replay introduced in [73] is used in order to speed up the learning process by saving trajectory samples and repeatedly replaying their corresponding updates. The main advantage is that in this way better use of a small amount of expensive real experience can be made when training the RL agent. Although very simple, this method has been successfully applied in complex problems such as Job Shop scheduling [168] and mobile robot navigation [123].

Finally, Dyna architectures [132, 130] effectively integrate online reinforcement learning and model-based algorithms into a process that alternately learns a model of the environment, updates the action-value function with DP algorithms, performs the corresponding greedy policy and so on. The idea is that it is more effective to use the experience collected through direct interaction to learn a model of the environment and to use DP algorithms to estimate the action-value function than learning online. Furthermore, the action-value function computed by the DP algorithm can be used to direct the agent to more rewarding regions of the environment and to improve the approximation of the model.

2.7.2 Least-Squares Policy Iteration

Least-Squares Policy Iteration (LSPI) [70] is an approximate dynamic programming algorithm that does not need an explicit approximation of

Algorithm 5 The LSTDQ algorithm

Input: set of samples D, feature space ϕ, policy π
Parameters: γ
Output: w^π

Initialize A, b
for all $\langle s, a, s', r\rangle \in D$ **do**
  $A \leftarrow A + \phi(s,a)(\phi(s,a) - \gamma\phi(s', \pi(s')))^\mathsf{T}$
  $b \leftarrow b + \phi(s,a)r$
end for
$\vartheta^\pi = A^{-1}b$

the transition model and the reward function. Starting from a generic linear function approximator $\widehat{Q}(s,a) = \phi(s,a)\vartheta$, the idea is to compute the least-squares solution of the system of Bellman equations. Given a matrix $\mathbf{\Pi} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}||\mathcal{A}|}$ defined according to policy π, the stochastic matrix $\mathbf{P} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}|}$ defined according to the transition model $\mathcal{P}(\cdot|s,a)$, and the vector $R \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ defined as the expected value of $\mathcal{R}(\cdot|s,a)$, the approximated Bellman equations becomes the overconstrained system

$$\widehat{Q}^\pi \approx R + \gamma\mathbf{P}\mathbf{\Pi}\widehat{Q}^\pi. \tag{2.23}$$

Recalling the approximator definition and solving the corresponding least-squares problem, the learning parameter vector is

$$\vartheta^\pi = \Big((\phi - \gamma\mathbf{P}\mathbf{\Pi}\phi)^\mathsf{T}(\phi - \gamma\mathbf{P}\mathbf{\Pi}\phi)\Big)^{-1}(\phi - \gamma\mathbf{P}\mathbf{\Pi}\phi)^\mathsf{T}R. \tag{2.24}$$

Once the learning parameters are computed for policy π, a policy improvement step is performed by taking the action that maximizes the action-value function in each state. Thereafter, policy evaluation and policy improvement steps are alternated until a termination condition is met, as in the DP policy iteration algorithm (Section 2.3.1). The main drawback in the computation of this solution is that a generative model of the environment is still required in order to draw independent samples from $\mathcal{P}(\cdot|s,a)$. An alternative formulation is to force the Bellman operator to have a fixed point when applied to the feature space, thus obtaining the well-formed system:

$$\widehat{Q}^\pi = \phi(\phi^\mathsf{T}\phi)^{-1}\phi^\mathsf{T}(R + \gamma\mathbf{P}\mathbf{\Pi}\widehat{Q}^\pi). \tag{2.25}$$

The corresponding learning parameter vector becomes:

$$\vartheta^\pi = \Big(\phi^\mathsf{T}(\phi - \gamma\mathbf{P}\mathbf{\Pi}\phi)\Big)^{-1}\phi^\mathsf{T}R. \tag{2.26}$$

Algorithm 6 The LSPI algorithm

Input: set of samples D, feature space ϕ, policy π
Parameters: γ, ϵ
Output: $\widehat{\pi}^*$

Initialize π_0, ϑ_0
while $\|\vartheta_i - \vartheta_{i-1}\| < \epsilon$ **do**
  $\vartheta_i \leftarrow LSTDQ(D, \phi, \pi_i)$
  $\pi_{i+1} \leftarrow \text{improve}(\pi_i, \vartheta_i)$
end while

Although in this formulation the transition model and the reward function are still present, using LSTDQ, an extension of LSTD [29] to action-value functions, the complexity of the computation of the solution does not depend on the dimensionality of the MDP model but only on the dimensionality of the feature space. Equation (2.26) is the solution of $(M \times M)$ linear system of the form:

$$Aw^\pi = b, \tag{2.27}$$

where $A = \phi^\mathsf{T}(\phi - \gamma \mathbf{P}\mathbf{\Pi}\phi)$ and $b = \phi^\mathsf{T}R$. The value of A and b can be incrementally estimated starting from a set of samples $D = \{\langle s, a, s', r\rangle_i\}_{i \in \mathbb{N}_m}$ [1] as shown in Algorithm 5. It is interesting to notice that no estimation of the transition model is needed and only the feature ϕ is evaluated in the state-action pairs of D. Finally, the overall policy iteration algorithm is summarized in Algorithm 6. Extensions of LSPI with kernel spaces are introduced in [58, 61].

2.7.3 Fitted Q-iteration

The idea of fitted solutions is to reformulate the learning of the value functions as a sequence of regression problems. Given a set of samples $D = \{\langle s, a, s', r\rangle_i\}_{i \in \mathbb{N}_m}$, either collected from the environment or drawn from a generative model, the goal is to learn an estimation of the optimal action-value function by iteratively extending the optimization horizon. Let us consider a MDP with horizon limited to one step. The optimal action-value function becomes:

$$Q_1^*(s, a) = E[\mathcal{R}(\cdot|s, a)] = R(s, a) \tag{2.28}$$

The problem of the approximation of Q_1^* is now a typical regression problem of supervised learning. In fact, from D we can define the training set

[1] $\mathbb{N}_m$ denote the set of integer numbers $\{1, \ldots, m\}$.

Algorithm 7 The Fitted Q-iteration algorithm

Input: set of samples D
Parameters: γ, N, regressor
Output: Q_N

Initialize Q_0
for $k = 1$ to N **do**
  Generate input set $x_i = (s_i, a_i)$, with $i = 1 \dots m$
  Generate output set $y_i = r_i + \max_{a'} Q_k(s'_i, a')$, with $i = 1 \dots m$
  $Q_{k+1} \leftarrow \text{regressor}(x, y)$
end for

$\{x_i, y_i\}_{i \in \mathbb{N}_m}$, where $x_i = (s_i, a_i)$ and $y_i = r_i$. This regression problem can be solved using any kind of regressor, e.g., neural-networks, regression trees, and so on. The result is an approximated action-value function $\widehat{Q}_1^*$. If the horizon is extended by one, the optimal action-value function becomes:

$$\begin{aligned} Q_2^*(s,a) &= E[\mathcal{R}(\cdot|s,a)] + \max_{a'} \sum_{s'} \mathcal{P}(s'|s,a)\gamma E[\mathcal{R}(\cdot|s',a')] \\ &= R(s,a) + \gamma \max_{a'} \sum_{s'} \mathcal{P}(s'|s,a) R(s',a'), \end{aligned} \tag{2.29}$$

that is, the expected reward of (s, a) plus the highest expected reward in the next step. Exploiting the approximation of the action-value function of the 1-step MDP, we can formulate the approximation of Q_2^* as a regression problem with a training set defined as $x_i = (s_i, a_i)$ and $y_i = r_i + \max_{a'} \widehat{Q}_1^*(s'_i, a')$. [2] This process can be iteratively repeated up to the horizon of the task at hand, thus obtaining an approximation of the optimal action-value function Q^*. The overall algorithm, usually referred to as *fitted Q-iteration* (FQI), is summarized in Algorithm 7, where *regressor* is any regression algorithm.

In [52, 90] it is proved that FQI in conjunction with kernel-based regressors (i.e., averagers) avoid the stability problems of polynomial regression and back-propagation regressors (Section 2.5.3). A detailed theoretical and empirical analysis of FQI with regression trees is carried out in [41]. Finally, empirical results of FQI with CMAC and neural networks are reported in [151] and [105].

[2] In case of samples drawn from a single trajectory $s'_i = s_{i+1}$.

3 Knowledge Transfer in Reinforcement Learning

In this chapter, we review the main approaches to transfer in supervised learning. Then, we review the most influential works of transfer in reinforcement learning, analyzing their goals, the scenarios of application, and the elements transferred across the tasks. Finally, we introduce an analysis of transfer in the context of batch reinforcement learning algorithms.

3.1 Introduction

Machine Learning (ML) algorithms are traditionally designed to learn one task at a time. In recent years, many research directions focused on designing techniques to extend ML algorithms to transfer solutions learned in one task to other tasks drawn from the same domain, in order to improve the learning performance. More precisely,

> *Transfer learning* refers to the problem of retaining and applying the knowledge learned in one or more tasks to efficiently develop an effective hypothesis for a new task. (*from NIPS "Inductive Transfer: 10 Years Later" Workshop* [117])

This challenging goal has been pursued following many different perspectives (e.g., meta-learning, multi-task learning, learning to learn, continual learning) and many empirical and theoretical results showed that learning algorithms can actually benefit from the transfer of knowledge across related tasks. Nonetheless, most of the research in transfer learning focused on the Supervised Learning (SL) paradigm and little attention has been devoted to the application of transfer to the Reinforcement Learning (RL) paradigm. Recent results show that the re-use of knowledge (e.g., policies, value functions) can greatly improve the performance of RL algorithms, but a systematic analysis about the way to obtain effective transfer in RL and about how the transferred knowledge affects the learning process is still lacking. The reason is that, in RL, many different aspects may contribute to the overall performance and, thus, it is

often difficult to analyze how the transferred knowledge actually impacts on the performance. For instance, options, a framework often used as a means for knowledge transfer across tasks, may affect the exploration strategy, the update of the action-value function estimate and, in case of function approximation, the stability and the convergence point of the approximator.

In this Chapter, we review the literature of transfer both in supervised and reinforcement learning paradigms, and we introduce the elements used in the rest of the thesis. More specifically, this Chapter has three main objectives: *(i)* by analyzing the literature of transfer in SL we introduce the two perspectives adopted in Chapters 4 and 5, that is, inductive transfer and multi-task learning, *(ii)* we provide a general formulation of the problem of transfer in RL and we propose a classification inspired by similar formalizations usually adopted in SL, *(iii)* we discuss how transfer learning can be achieved in the batch RL framework.

3.2 Transfer in Supervised Learning

A comprehensive overview of transfer in SL is beyond the scope of this thesis. For partial reviews of transfer learning and related topics we refer the reader to [114, 149, 157, 169].[1] In this section, we focus on some of the main works in transfer learning, with the aim of providing a brief historical perspective of the topic and some basic elements for the comparison with transfer in RL.

3.2.1 Early Works of Transfer Learning

The idea of retaining and reusing knowledge in order to improve the performance of learning algorithms dates back to early stages of ML [86]. In fact, it is widely recognized that a good representation (or *bias*) is the most critical aspect for the performance of any learning algorithm, and the development of techniques that automatically change the representation according to the task at hand is one of the main objectives of large part of the research in ML. Most of the research in transfer learning [44] identified the single-problem perspective usually adopted in ML as a limit for the definition of effective methods for the inductive construction of good representations. On the other hand, taking inspiration from studies in psychology and neuroscience [50, 49], the multi-problem

[1] A comprehensive bibliography is also available at:
http://www.cs.berkeley.edu/~russell/classes/cs294/f05/all-papers.html.

point of view, in which knowledge is retained from previously solved problems and reused in new problems, is considered as the most suitable perspective to design techniques of inductive bias [155].

Starting from this idea, different approaches to the problem of transfer have been proposed. *Meta-learning* [157] studies how learning systems can increase in efficiency through experience. The objective is to learn how the learning itself can be modified according to the task under study. Meta-learning studies how to dynamically choose the right bias, as opposed to single-problem learning where the bias is fixed a priori, or user-parameterized. As a result, while different applications of single-problem algorithms over the same data always produce the same result, independently of the performance, meta-learning aims at discovering ways to dynamically search for the best learning strategy as the number of tasks increases. Other solutions refer to the so-called *learning to learn* approach [149, 19], a research direction that generalizes the ML traditional perspective. Following the definition of ML in [86], given a set of tasks, training sets for each task and a set of performance measures, a learning algorithm is said to learn to learn if its performance in each task improves with the experience and the number of tasks. Finally, *lifelong learning* approaches [116] focused on the development of techniques of inductive bias. A lifelong learning system is focused on designing effective methods for the retention of domain knowledge and for reusing such knowledge as inductive bias for learning algorithms.

Although the previous research directions are different in perspectives and scenarios, they all share the idea that the performance of a learning algorithm can be improved through the *transfer* of knowledge across a set of related tasks.

3.2.2 Inductive Transfer and Multi-Task Learning

In recent years, research on transfer learning gained increasing relevance in the SL community and obtained relevant theoretical results and successfully solved a wide range of applications. A complete formalization and analysis of the transfer problem is introduced in [18]. Along with an extension of the traditional single-task approach to take into account training examples coming from different tasks, generalization bounds of learning theory [156] are re-formulated, showing that learning multiple tasks from the same domain can give better generalization than learning independently on each task. In [22] a notion of relatedness among tasks is implicitly introduced using a data generating framework and the complexity bounds obtained by a multi-task learning algorithm are proved

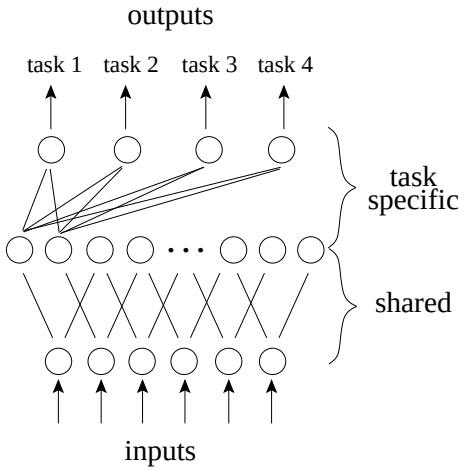


Figure 3.1: The structure of a neural network for multi-task learning [34].

to be more tight (i.e., smaller sample sets per task) than those in the single-task learning case.

From an algorithmic point of view, the implementation of transfer learning followed different directions. Many works [18, 34, 114] focused on the use of neural networks to transfer knowledge across tasks. Instead of using as many neural networks as the tasks to be solved, the idea is to design one neural network divided in two different layers (Figure 3.1). Under the assumption that all the tasks share the same input space, the first layer is meant to learn the features shared across the tasks, while the second layer is specific for each task and computes the prediction. Another framework commonly adopted in transfer learning problems is the Hierarchical Bayes model [14, 166, 42, 149, 167]. According to this model, the problem of inductive bias is defined as the problem of learning the hyper-distribution common to all the tasks together with the models of each task. Finally, recent works [3, 43, 9] focused on the extension of regularized methods to the transfer problem. While regularization is adopted in single-task problems to force the learning of smooth functions, in multi-task problems regularization is defined to force the tasks to share the same underlying representation.

Although the previous works share the idea that in many applications learning on multiple tasks is preferable than learning on each single task separately, they often focus on different objectives. We can roughly

classify the previous methods in two main approaches: [2]

- *Inductive Transfer.* Inductive Transfer is concerned with learning to predict in one task (*target* task) based on training samples coming from other tasks (*source* tasks). Therefore, the objective is to reduce the number of samples needed to learn solutions in unknown tasks.

- *Multi-task Learning.* Multi-Task Learning is concerned with utilizing training data from different tasks to help predicting in all of them. Therefore, the focus is on a given set of tasks and the objective is to improve the generalization performance of the learning algorithm on the very same tasks.

The difference between these two perspectives reflects also on the difference in their corresponding applications. Inductive transfer algorithms are usually applied to highly structured problems in which the main problem is how to map solutions previously learned in one domain to a new domain. Examples of application of inductive transfer are image recognition [56], planning problems [71], general game playing [15]. On the other hand, multi-task learning obtained relevant results in classification and regression problems in which the tasks are actually known in advance and few examples for each of them are available. Examples of application of multi-task learning are collaborative filtering [165], text classification [3], medical decision making [34].

Although strictly related, a unifying framework able to formalize both inductive transfer and multi-task learning approaches is still lacking. In the rest of the thesis, we investigate the possibility to follow these two perspectives in RL.

3.3 Transfer in Reinforcement Learning

The problem of transfer in RL shares many issues with transfer in SL. For instance, the problem of improving the generalization performance of a learning algorithm on a limited set of tasks can be directly generalized to the RL paradigm in case of function approximation algorithms (see Chapter 5). On the other hand, the are issues that are specific for the RL paradigm. For instance, the problem of exploration strategies for

[2] Unfortunately, at the moment, there is not a commonly accepted distinction among different perspectives in transfer learning. The distinction followed here is inspired by [18, 21, 6].

collecting samples is not present in supervised algorithms, as samples are given in advance.

In this section, we propose a general formulation of the problem of transfer in RL. Furthermore, we provide a perspective view on the most relevant results of transfer research in RL. Here, the objective is not to provide a detailed analysis of each work (more thorough descriptions are postponed to Chapters 4 and 5) but to analyze which aspects are usually considered in transfer in RL and discuss which aspects are still unexplored.

3.3.1 Problem Formulation

Transfer learning is a very general problem and it is difficult to provide a formal definition that takes into account all the possible perspectives and approaches to the problem. In general, a SL transfer algorithm is formalized as a mapping from samples and hypotheses coming from the solution of a set of tasks, to the learning algorithm used to learn the solution of new tasks. Thus, the learning algorithm is biases so as to improve its performance when applied to tasks that are somehow related with those used as input for the transfer algorithm. Here, we adapt the formalisms introduced in transfer in [18, 114] to the RL paradigm and we introduce general definitions and symbols used throughout the rest of the thesis.

Definition 3.1 *A* task *T is a MDP defined by the tuple $\langle \mathcal{S}_T, \mathcal{A}_T, \mathcal{P}_T, \mathcal{R}_T \rangle$, in which the state and action spaces define the* context*, the transition model $\mathcal{P}_T$ defines the* dynamics*, and the reward function $\mathcal{R}_T$ defines the* goal*. The* task space *is denoted by $\mathcal{T}$.*

Definition 3.2 *An* environment *E is defined by the tuple $\langle \mathcal{T}, \Omega \rangle$, where $\mathcal{T}$ is the task space and Ω is a* task distribution *defined on $\mathcal{T}$ that provides the probability of a task $T \in \mathcal{T}$ to occur.*

The characteristics of a problem of transfer defined on an environment E depends on the definition of the distribution $\Omega(T)$.

Definition 3.3 *A problem in which all the tasks share the same context (i.e., state and action space) and the same transition model, is a* goal transfer *problem ($\Omega(T) = \Omega_{\mathcal{R}}(T)$). A problem in which tasks share the same context (i.e., state and action space) and the same reward function, is a* dynamics transfer *problem ($\Omega(T) = \Omega_{\mathcal{P}}(T)$). Finally, a problem in which $\Omega(T)$ is a joint distribution on both the reward function and the*

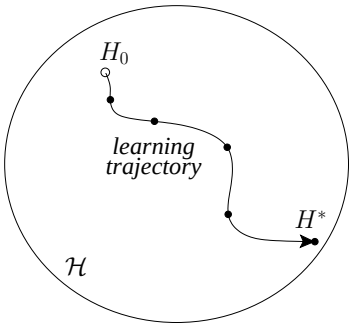


Figure 3.2: Representation of the learning process of a RL algorithm (inspired by [114]). The learning algorithm moves from an initial hypothesis H_0 to an optimal hypothesis H^*.

transition model, is a domain transfer *problem without any constraint on the state-action space of the tasks.*

A general RL algorithm can be qualitatively defined as a mapping from the task space to the hypothesis space

$$A_{Learning} : \mathcal{T} \rightarrow \mathcal{H}, \tag{3.1}$$

where $\mathcal{H}$ is a hypothesis space that can be either the space of (action) value functions or the space of policies. Depending on the learning algorithm, the input task $T \in \mathcal{T}$ can be interpreted differently. For instance, when using a model-based algorithm (e.g., value iteration), T is the exact definition of the MDP. On the other hand, in an on-line algorithm (e.g., Q-learning with ϵ-greedy exploration), T is a process sampled by the learning algorithm according to its exploration strategy. As shown in Figure 3.2, a learning algorithm is initialized with a hypothesis $H_0 \in \mathcal{H}$ that is changed during the learning process until an optimal hypothesis H^* is learned. As it can be noticed, the hypothesis space $\mathcal{H}$ defines the hypothesis that can be learned, while $A_{Learning}$ determines the *learning trajectory* followed during the learning process. [3]

In the general inductive transfer problem [4] (Figure 3.3), given an environment E, we sample $n-1$ source tasks according to the task distribu-

[3] More precisely, since machine learning algorithms are stochastic, the learning trajectory in Figure 3.2 is to be considered as averaged over all the possible learning trajectories.

[4] The scenario in case of multi-task learning is almost the same except that the transfer algorithm gets as input all the tasks and computes their solutions as output.

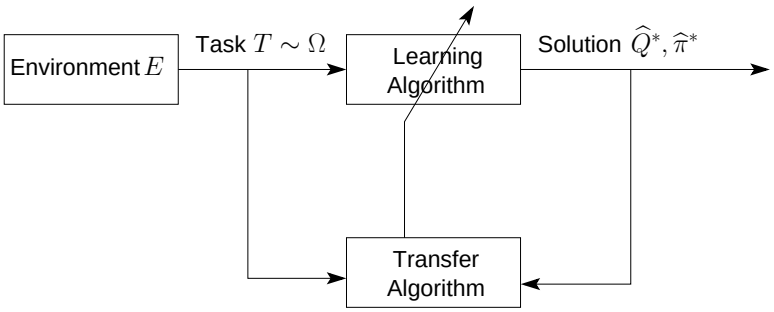


Figure 3.3: Qualitative description of a transfer process. The transfer algorithm biases the learning algorithm on the basis of the knowledge retained from other tasks.

tion Ω, and the objective is to optimize the learning process on any other target task drawn from Ω. On the basis of the information retained from the source tasks and the information available for the target task, the goal of a transfer algorithm is to modify the definition and the structure of a RL algorithm so as to improve its learning performance. In general, a transfer algorithm $A_{Transfer}$ is a mapping from the knowledge retained so far to the task space and the hypothesis space:

$$A_{Transfer} : \mathcal{H}^{n-1} \times \mathcal{T}^n \rightarrow \mathbb{H}, \mathbb{T}, \tag{3.2}$$

where $\mathbb{H}$ is the family of all the possible hypothesis spaces and $\mathbb{T}$ is the family of all the possible task spaces. In order to make this notation clearer, we consider two transfer algorithms (A_{T1} and A_{T2}) that modify the hypothesis space and the task space respectively. Both the algorithms get as input the information collected from $n-1$ source tasks and about the target task itself, that is hypotheses $\{H_i\}_{i \in \mathbb{N}_{n-1}}$, where $H_i \in \mathcal{H}$, and task information $\{T_j\}_{j \in \mathbb{N}_n}$, where $T_j \in \mathcal{T}$. On the basis of this information, A_{T1} builds a set of options $\mathcal{O}$. As a result, the output of the transfer algorithm is a new task space $\mathcal{T}' \in \mathbb{T}$ in which the action space of all the tasks is augmented with the set of options ($\mathcal{A}' = \mathcal{A} \cup \mathcal{O}$). On the other hand, A_{T2} changes the hypothesis space in order to improve the performance of the learning algorithm. In particular, let us consider the case in which $\mathcal{H} = \mathcal{Q}$, that is, the hypothesis set is the set of the action value functions that can be learned by a given function approximator. The transfer algorithm adapts the structure of the approximator according to the characteristics of the tasks at hand. As a result, the initial hypothesis space $\mathcal{H}$ is changed into a new hypothesis

space $\mathcal{H}' \in \mathbb{H}$.

In general, which information the transfer algorithm uses and how it modifies the learning algorithm depends on the objectives, the scenario and the structure of the algorithm. A classification of these elements is proposed in the next section.

3.3.2 The Dimensions of Transfer

In this section, we propose three dimensions used in the following to classify algorithms for transfer in RL. In particular, we focus on the *objectives* of a transfer algorithm, the *scenarios* in which it is applied and the *knowledge* it transfers across the tasks.

Objectives

The objectives of transfer in RL can be adapted from the objectives suggested for the general problem of transfer in [71] (Figure 3.4):

(I) *Learning speed improvement.* This objective is about the reduction of the complexity of the learning algorithm in terms of the experience needed to learn the solution of the task at hand. As new tasks are sampled from Ω, the knowledge retained from a set of previously solved tasks can be used to bias the learning algorithm towards a limited set of solutions, so as to reduce its learning time. The complexity of a learning algorithm is usually measured by the number of samples needed to achieve a desired performance. In SL, this scenario is usually referred to as the inductive transfer problem. In RL, this objective is pursued following two different approaches. The first approach is to make the algorithm more effective in using the experience collected from the exploration of the environment. For instance, as shown in [64] and [55], the use of options can improve the efficiency of value iteration backups by updating value function estimates with the total reward collected by an option, and thus reducing the number of iterations before converging to a nearly optimal solution. In this case, the transfer problem is to identify the set of options that leads to the greatest speedup [64]. The second aspect is about the strategy used to collect the samples. While in SL samples are given in advance, in online RL algorithms the agent follows a given exploration strategy and samples are obtained from direct interaction with the environment. The experience collected by solving a set of tasks can lead to the definition of better exploration strategies for new related

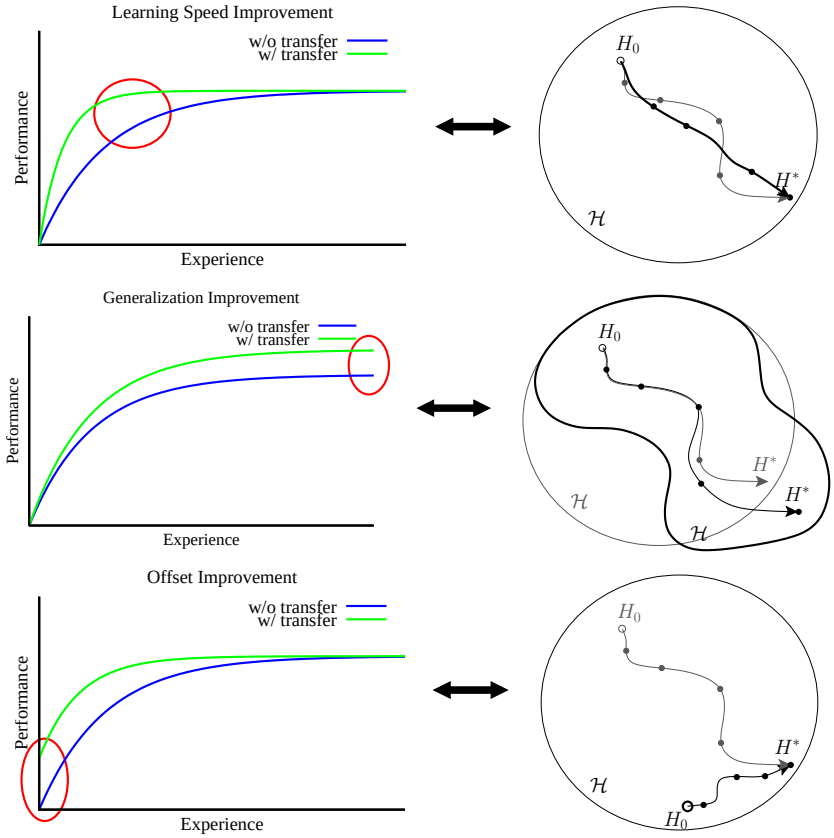


Figure 3.4: The three objectives of transfer learning [71]: learning speed improvement, generalization improvement, initial offset improvement. On the left: qualitative learning curves with and without transfer. On the right: the corresponding changes to the learning trajectory.

tasks. For instance, if all the tasks have goals in a limited region of the state space, an exploration strategy that frequently visits that region will lead to more informative samples that can be used to reduce the learning time.
Recalling the general definition in (3.2) this objective is usually achieved by transfer algorithms such as

$$A_{Transfer} : \mathcal{H}^{n-1} \times \mathcal{T}^n \rightarrow \mathbb{T}, \tag{3.3}$$

that is, algorithms that bias the definition of the task (e.g., augmentation of the action space with options) so as to make the

learning algorithm to solve the task faster than without transfer. In principle, also the hypothesis space $\mathcal{H}$ could be changed by reducing the number of hypotheses that the learning algorithm can learn (e.g., the reduction of the actions in the action space proposed in [113]), but this approach is rarely considered in transfer literature. It is worth noting that the reduction of $\mathcal{H}$ has the secondary effect to change the generalization performance. In fact, it may happen that the hypotheses removed from $\mathcal{H}$ through transfer are actually optimal for tasks that will eventually occur.

(II) *Generalization improvement.* In most of the problems of practical interest, an exact estimation of the value function is not possible (e.g., problems with continuous state-action spaces) and the use of function-approximation techniques is mandatory. The more accurate the approximation, the better the generalization (and the performance). In many cases, the difference between the objective of learning speed and of generalization improvement is subtle. In fact, the improvement in learning speed at a given point of the learning process can be seen as an improvement in the performance due to a better approximation of the optimal action-value function. In order to make this difference clearer, we consider the objective of generalization improvement as the improvement in the performance at convergence. As discussed in Section 2.5, in value-based algorithms, the accuracy of the approximation is strictly dependent on the structure of the approximator, which defines the hypothesis space (i.e., the functions) that can be represented. In this case, the hypothesis space $\mathcal{H} \in \mathbb{H}$ is the space of action-value functions $\mathcal{Q}$ and the objective of a transfer algorithm $A_{Transfer}$ is to adapt the structure of the approximator (i.e., $\mathcal{Q}$), so as to obtain the best approximation of the optimal value functions of the tasks in Ω.
In general, a transfer algorithm pursuing the objective of generalization improvement can be defined as

$$A_{Transfer} : \mathcal{H}^{n-1} \times \mathcal{T}^n \rightarrow \mathbb{H}. \tag{3.4}$$

Therefore, $A_{Transfer}$ biases the solution space with the aim of improving the approximation accuracy of the optimal solutions of the tasks at hand.

(III) *Offset improvement.* The learning process usually starts from a random hypothesis drawn from the hypothesis space. The initial hypothesis may affect two aspects of the learning performance: *(i)*

the initial performance, *(ii)* the learning speed. In fact, if the initialization is "near" to the optimal hypothesis, the learning algorithm is expected to converge faster than with other initialization hypothesis. At the same time, we expect to have a high initial performance if the initial hypothesis is similar to the optimal solution of the task at hand. It is worth noting that these two aspects are not necessarily related. Let us consider a source task whose optimal policy is significantly different from the optimal policy of the target task but that, at the same time, it achieves only a slightly suboptimal performance (e.g., two goal states with different final positive rewards in different regions of the state space). In this case, the improvement in the initial performance can be obtained by initializing the learning algorithm to the optimal policy of the source task, but this may lead to worsen the learning speed. In fact, the initial policy does not provide samples of the actual optimal policy of the task at hand, thus slowing down the learning algorithm. On the other hand, it could be possible that the policy transferred from the source task is an effective exploration strategy for learning the optimal policy of the target task, but that it also achieves very poor performance.Therefore, in the case of transfer of policies, an effective transfer algorithm should learn a suitable trade-off between the transfer of rewarding policies and the transfer of policies that improve the information about the optimal policy.

More in general, the objective of a transfer algorithm is to use the knowledge retained from the solution of a set of tasks to define an initial hypothesis that is likely to improve the learning performance on new tasks drawn from the same distribution Ω [164]. The transfer algorithm can be defined as

$$A_{Transfer} : \mathcal{H}^{n-1} \times \mathcal{T}^n \rightarrow \mathcal{H}. \tag{3.5}$$

Unlike the previous objectives, here the transfer algorithm directly affects the initial hypothesis $H_0 \in \mathcal{H}$ used to initialize the learning algorithm. It is interesting to notice that H_0 can be either the policy or the action-value function used to initialize the algorithm.

Scenarios

The possible scenarios of transfer depend on the differences between the tasks at hand (see Definition 3.3):

(I) *Goal Transfer.* In case of goal transfer, the tasks share all the same dynamics but have different goals. In RL, this means that the transition model is shared across all the tasks, while the reward function, that specifies the goal of the task, is different. This scenario is typical in all the applications in which the interaction between the agent and the environment is constant, while it is up to the designer to specify the goal. For instance, in most of the robotic navigation problems the dynamics of the robot is always the same, while the goal position varies.

(II) *Dynamics Transfer.* In case of dynamics transfer, each task has a different dynamics while the goal is always the same. This scenario is typical in many control applications in which the goal is intrinsic in the application (e.g., the stabilization of a pendulum, the achievement of a set point) while the dynamics can be affected from parameters of the environment or characteristics of the object to be controlled (e.g., friction, masses). Another wide class of applications is game playing. Depending on the opponent strategy, the dynamics of the environment can significantly change, while the goal of the game (the condition of victory) is always the same. For instance, in the Texas Hold'em Poker [25], the outcome of the actions of the agents (e.g., fold, bet) does not depend only on the rules of the game, but also on the opponent strategy. On the other hand, the goal, i.e., the reward function, is always the same independently from the opponent.

(III) *Domain Transfer.* In the case of domain transfer, each task may have different dynamics and goals. Furthermore, the tasks may also be defined on different state-action spaces. Therefore, the transfer of knowledge (e.g., a policy) requires the mapping between the state-action spaces of the source and target tasks. The problem of domain transfer is the most general and complex problem of transfer.

Transferred Knowledge

A general classification of the knowledge retained and transferred across tasks in SL is proposed in [114]. Here, we propose a different classification specific for transfer in RL. In particular, we do not distinguish between retained and transferred knowledge, since in most of the transfer RL algorithms these two elements coincide. According to the qualitative definition of transfer algorithm in Equation 3.2, the main elements that

can be retained and transferred are: experience collected during the learning process and structural knowledge about the environment.

(I) *Experience.* During the learning process the algorithm collects experience and learns a suitable solution for the task at hand. This experience can then be reused to improve the learning performance in related tasks. The experience an algorithm obtains from learning on a task can be roughly divided into three groups: *(i)* samples, *(ii)* value functions, *(iii)* policies. For instance, the transfer of trajectory samples can be used to simplify the estimation of the model of new tasks [129]. On the other hand, the transfer of solutions (either policies or value functions) is commonly adopted to initialize the learning algorithm and to speed-up the learning [141, 145].

(II) *Structural knowledge.* After learning on different tasks, transfer algorithms often perform an abstraction process that leads to the definition of general knowledge about the environment from which the tasks are drawn. As a result, they change the definition of the problem in order to take into account the knowledge about the structure of the tasks and of their solutions. We further divide this type of transfer in two categories: *(i)* task representation, *(ii)* solution representation. In the former, we consider algorithms that modify the definition of the tasks ($\mathcal{T}$), such as reward shaping [68], MDP augmentation through options [118], bisimulation [47]. The latter contains algorithms that modify the structure of the solution ($\mathcal{H}$). For instance, in [78] the basis function of a linear function approximator are extracted so as to improve the approximation accuracy on a wide range of value functions.

3.3.3 State of the Art in Transfer Reinforcement Learning

In this section, we summarize the main works of transfer in RL (see Table 3.2) following the dimensions introduced in the previous section. Some of the works are present in more than one category because they consider multiple objectives or merge different techniques in one algorithm.

The idea of transferring solutions dates back to early research in RL. Transfer of solutions has been often adopted to simplify the learning process in robotic applications. A number of works [10, 77, 111] showed that transfer from simple to complex tasks can be an effective experimental methodology in order to reduce the learning complexity and solve real-world problems. Nonetheless, the process of transfer proposed in these

		Experience			**Structure**	
		Samples	*Value Function*	*Policy*	*Task Representation*	*Solution Representation*
	Dynamics			[27]		
Convergence Time	*Goal*		[120] [139]	[46]	[23] [26] [77] [118] [150]	[48] [82]
	Domain	[129] [140] **[This thesis]**	[74] [135] [141] [143]	[75] [145] [152]	[11] [47] [64] [68] [69] [74] [97] [104] [113] [158]	[11] [39] [102] [124] [145]
	Dynamics					[45]
Generalization	*Goal*					[78]
	Domain					[106] [142] **[This thesis]**
	Dynamics		[15]		[15]	
Offset	*Goal*					
	Domain		[141] [143]	[138] [145] [153]	[124] [164]	[142]

Table 3.1: Classification of the main works of transfer in RL.

works is often carried out by hand (mainly transferring hand-coded solutions) and it cannot be easily extended to other scenarios.

As it can be noticed in Table 3.2, most of the recent research in transfer RL focused on the objective of improving the learning speed. This objective is mainly pursued by transferring task representation or transferring solutions across tasks. In particular, many works [118, 93, 26] focused on the augmentation of the action space of tasks with a set of options suitable for the solution of a wide range of tasks sharing the same domain but with different goals. A different use of options is proposed in [69], in which the options are defined in a general space (i.e., the *agent space*) composed by a set of features shared across the tasks. These options are more general and can be used even when the domain and the context of the tasks are different. The task representation is changed also in [68], where a automatic mechanism of reward shaping is used to estimate intermediate rewards on the basis of previous tasks, thus reducing the learning time. Transfer of the solution representation is performed in [82], where the task decomposition of MAXQ is used to transfer subgoals and subpolicies across tasks sharing the same domain but with different goals. The improvement in learning speed can also be obtained through direct transfer of a policy from source to target task. In [46] a library of policies is used and the agent can exploit previously learned policies in order to reduce the number of episodes to learn the solution of the current task. In [145] the more general case of transfer of solutions between tasks with different state and action spaces is considered.
Although the previous approaches cover many different aspects of the transfer problem for the learning speed improvement, it is very difficult to compare their effectiveness and results because the performance metrics are often different and a common formal definition of the objective is still lacking. Furthermore, there are still aspects of the problem that can be investigated. A relevant topic that is rarely considered is the issue of negative transfer. It is widely recognized that transfer is effective only if tasks are related, otherwise the learning performance can even get worse than without transfer. Therefore, it is important to define a suitable measure of distance between tasks and to design algorithms able to avoid transfer whenever the distance is too high. From the algorithmic point of view, whereas a common practice in many multi-task works [34, 114] is to transfer training samples, only few works [129, 140] address the problem of transfer in RL by reusing the experience directly collected by the agent from direct interaction with the environment, that

is, the trajectory samples.

Although the objective of improving the offset is in general distinct from the improvement in learning speed, many algorithms do not distinguish between these two aspects and the transfer is designed to achieve both these results. The most relevant works for this objective are focused on the definition of suitable methods for the transfer of solutions across tasks with different state and action spaces. Indeed, this is the most challenging problem in transfer. In fact, in many applications of interest the "nature" of the problem is fixed across the tasks and the optimal solution for one task is a nearly-optimal solution for the others. In this case, the main problem is to map the solution learned in one task to the state and action space of another task, thus initializing the learning algorithm to a convenient solution. There are many different aspects of a reinforcement learning algorithm that can be initialized: value function [74, 143, 141], policy [145], structure [142, 15]. While the mapping between source and target tasks is usually designed by hand, in recent works [124, 138, 74, 145] algorithms that learn the most suitable mapping between state and action spaces so as to improve the effectiveness of the transfer are proposed. A different perspective is followed in [164], where a hierarchical Bayesian approach is followed in order to learn a prior on the distribution of the tasks and, thus, to initialize the learning system to the solution of the most probable task.
The initialization of learning algorithms by mapping solutions from source to target task obtained interesting results even in complex problems (e.g., simulated soccer keepaway). Similarly to the learning speed improvement, the main open problem is the possibility to estimate the expected improvement the can be obtained from initializing the learning algorithm. Furthermore, almost no work distinguishes between the two aspects of this objective, that is, the initial performance and the learning speed. As discussed in Section 3.3.2, a transferred solution can be very effective in terms of performance (i.e., it is *near* to the optimal solution) but, at the same time, it can increase the complexity of learning the actual optimal solution. Conversely, it may happen that the learning algorithm needs only few episodes before converging to the optimal solution when initialized with the solution of a source task, but it has a very poor initial performance. Therefore, it is important to investigate when transfer actually achieves to initialize the learning algorithm so as to obtain a good initial performance and, at the same dime, not to prevent from learning the optimal solution in few learning episodes.

Finally, the objective of generalization improvement, although of pri-

mary concern in many transfer algorithms in SL, is still relatively unexplored in RL and, as it can be noticed from Table 3.2, most of the solutions focused on the transfer of structure representation. In Section 3.3.2 we defined the objective of generalization improvement as the improvement of the performance at convergence. While the improvement of the performance during the learning process can be achieved both by transferring experience and by changing the structure of the problem (see the algorithms for speed improvement), at convergence the performance is strictly related to the hypothesis space and, thus, only transfer algorithms that directly change $\mathcal{H}$ can achieve the objective of generalization improvement. This is the reason for the lack of solutions that transfer experience, since it is not possible to design an algorithm able to improve the performance of the learning algorithm by either transferring samples or solutions. In [142] the problem of transferring some parts of the structure of an approximator is considered. The generalization of the proto-value function framework to the problem of transfer between tasks with different domains is faced in [45]. Finally, in [106] a preliminary formal model for the multi-task problem in the context of RL algorithms is proposed.
Since in SL the problem of generalization improvement is usually pursued by following a multi-task perspective, the most promising investigation direction in RL is to adapt SL solutions to the context of RL problems. In particular, value-based RL algorithms could be integrated with hierarchical Bayesian approaches and regularized algorithms so as to change the hypothesis space ($\mathcal{Q}$ in this case) in order to improve the approximation accuracy of the approximator at hand in a number of tasks.

As far as the frameworks are concerned, most of the previous works rely on value-based, TD algorithms and often use hierarchical frameworks for task decomposition. Little attention has been devoted to other approaches. For instance, only few works [145] focused on knowledge transfer in policy search algorithms. Similarly to solution representation transfer for value functions, a particularly interesting aspect of transfer with policy search algorithms would be the analysis of how the policy space could be adapted in order to achieve generalization improvement. Another approach rarely considered is the use of batch algorithms, as usually done in SL. For instance, in the proto-value function framework [45, 78] LSPI is used only for the approximation of the action-value function but it is not related with the process that actually generate the proto-value functions.

3.3.4 Alternative Models of Transfer Learning

Besides the perspective introduced in the previous sections, other models of transfer have been proposed in literature. Some of the scenarios described in the following have been considered only in the context of transfer in SL and can represent interesting fields for future investigation in RL.

Online Multi-Task Learning

Multi-task learning is defined as the problem of learning multiple tasks in parallel. Most of the works in multi-task learning follow a batch perspective in which all the samples from all the task are given in advance. In [37] an online perspective is considered. On each online round, the learning algorithm receives one sample and makes a prediction for each of the tasks. The basic assumption about the relatedness among the tasks is captured by using a single global loss function to evaluate the quality of the multiple predictions made on each round. In RL, this perspective can be followed in case of goal transfer with an online algorithm. In fact, while the agent explores the environment, several value functions can be simultaneously updated using a set of reward functions.

Transfer Across Tasks on Hidden Variables

This scenario is a specific case of the general model introduced in Section 3.3.1. In many environments, the characteristics of the tasks are determined by few parameters that cannot be directly observed. In this case, a transfer algorithm should estimate the functional relationship between the parameters and the solution of the corresponding task. Therefore, the transfer algorithm could simply "transform" the solution of one task to the solution of another task by simpling estimating the value of the parameters. Similar scenarios are considered in [23, 144].

Sequential Transfer

In the general definition of transfer introduced in Section 3.3.1, Ω is a distribution over the task space $\mathcal{T}$ and it does not provide a specific sequence of tasks. On the other hand, transfer techniques have been often applied to sequence of tasks with progressive difficulty [75]. The use of dummy tasks is a common practice in robotic applications in which it is possible to progressively exploit more sensors and actuators, so as to smoothly increase the complexity from one task to the other.

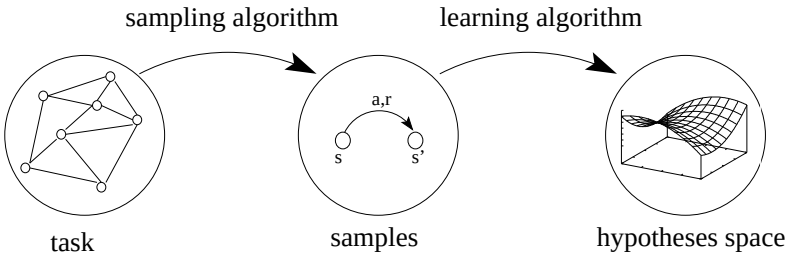


Figure 3.5: The learning process of a generic batch RL algorithm.

Tracking

A recent work by Sutton *et al.* [135] proposes tracking as the most suitable perspective in which transfer learning should be stated. The problem of learning on a sequence of learning problems is similar to the problem of tracking in non-stationary environments. In the paper, they highlight that transfer learning algorithms usually work only for some sets of tasks but not for others, and it is not clear where the tasks would come from in real applications. Therefore, they propose to ground transfer learning algorithms in the scenario of learning on a sequence of tasks generated by a stationary environment and thus inherently related by being parts of an overall problem.

3.4 Transfer in Batch Reinforcement Learning

Batch RL algorithms are gaining more and more interest in RL community because of their interesting theoretical properties [70, 4] and experimental success [65]. The main feature of batch RL algorithms is an effective use of experience obtained by dividing the RL process in two distinct phases of sampling and learning. As discussed in the previous sections, batch RL algorithms have been rarely adopted in transfer learning up to now. In this section, we discuss how batch RL can also be used as a suitable framework for transfer in RL.

Batch RL is an appealing framework for achieving transfer in RL for two main reasons: *(i)* the distinction between sampling and learning allows a more thorough analysis about how transfer objectives can be achieved, *(ii)* the learning phase can be stated as a supervised problem and, thus, transfer solutions in SL can be easily extended to transfer in RL.

3.4.1 Batch Reinforcement Learning

The learning process of a batch RL algorithm can be represented as in Figure 3.5. The *sampling algorithm* defines the way the agent explores the task and collects trajectory samples, used to form a set of samples. Once the sampling algorithm is finished, the set of samples is used as input for a *learning algorithm* that computes an approximation of the optimal action-value function (Q^*). As the number of samples tends to infinity the learning algorithm learns the best approximation of Q^* according to the space of functions that can be represented by the function approximator. [5]

In the following, we denote a set of m samples drawn from T as $\widehat{T} = \{\tau_i\}_{i \in \mathbb{N}_m}$, where $\tau_i = \langle s_i, a_i, s'_i, r_i \rangle$. In general, sample sets $\widehat{T}$ belong to a space $\widehat{\mathcal{T}}$ that contains all the sample sets that can be generated from the tasks in the task space $\mathcal{T}$. Following the general definition of a RL algorithm in (3.1), we define:

Definition 3.4 *A* sampling algorithm $A_{Sampling}$ *is a mapping from the space of tasks* $\mathcal{T}$ *to the space of set of samples* $\widehat{\mathcal{T}}$

$$A_{Sampling} : \mathcal{T} \rightarrow \widehat{\mathcal{T}}, \tag{3.6}$$

where the sample set $\widehat{T} \in \widehat{\mathcal{T}}$ *contains samples drawn from* $T \in \mathcal{T}$*.*

Definition 3.5 *A* learning algorithm $A_{Learning}$ *is a mapping from the space sample sets* $\widehat{\mathcal{T}}$ *to the space of action-value functions* $\mathcal{Q}$

$$A_{Learning} : \widehat{\mathcal{T}} \rightarrow \mathcal{Q}, \tag{3.7}$$

where $\mathcal{Q}$ *defines the action-value functions that can be learned by the learning algorithm.*

The objective of the sampling algorithm is to collect samples that are informative (e.g., samples of the goal states) for the achievement of the task, while the objective of the learning algorithm is to exploit the samples collected by the sampling algorithm in order to compute the most accurate approximation of the action-value function. It is interesting to notice, that, unlike on-line algorithms, in batch RL, sampling and learning algorithms are not strictly related. In fact, the value function approximated by the learning algorithm does not impact on the sampling strategy and, at the same time, the sampling algorithm does not directly affect the actual possibility for a function approximator to converge, as it may happen in on-line algorithms [53].

[5] The function approximator can represent only a limited space of functions that, in general, does not contain the optimal action-value function.

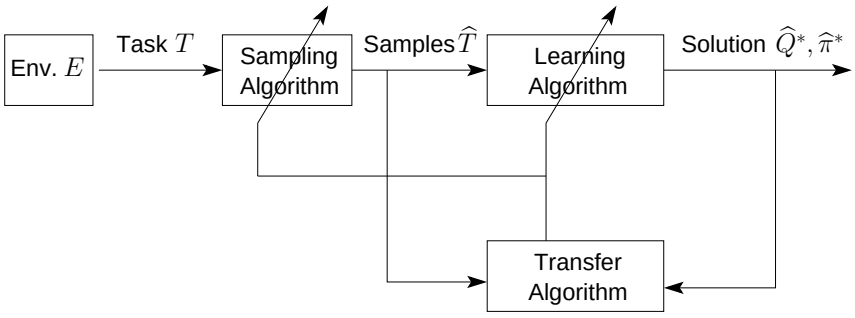


Figure 3.6: A qualitative representation of a transfer algorithm in batch RL. The transfer algorithm biases both the sampling and the learning algorithms according to the knowledge retained from other tasks.

3.4.2 A Batch Reinforcement Learning Approach to Transfer

Building on the previous definitions, we analyze how a transfer algorithm could affect the performance of a batch RL algorithm. A general model of the transfer problem in batch RL, is depicted in Figure 3.6. Unlike the model in Section 3.3.1, here we have two distinct algorithms and we consider the case in which the transfer algorithm can either modify the sampling algorithm or the learning algorithm. Furthermore, the transfer algorithm has not direct access to the model of the task T but it can only receive information about the task from the sampling algorithm.

Thanks to the distinct effects of the two phases of a batch RL system, a transfer algorithm can pursue the objectives described in Section 3.3.2 by modifying the definition of each algorithm separately. Elements that can affect the performance of a batch RL algorithm are:

- *Set of samples.* The complexity of a batch RL algorithm is usually measured as the number of samples directly collected from the task. The "quality" of samples significantly impacts on the learning performance of batch RL algorithms, the more informative the samples, the better the final approximation of the optimal action-value function. Unlike on-line approaches, in batch RL only the sampling algorithm is responsible for the collection of samples and it does not depend on the learning algorithm. Therefore, the objective of learning speed improvement can be achieved by directly biasing the sampling algorithm with the aim of reducing the number of samples needed to learn the optimal solution. Following the classification in Section 3.3.2, a transfer algorithm can impact

on the effectiveness of the set of samples used as input for the learning algorithm by transferring either structural knowledge or experience. In particular, while transfer of structural knowledge can impact on the definition of the task, the transfer of experience can either bias the exploration strategy towards specific regions of the state space (e.g., transfer of policy) or integrate the samples collected from the task with samples saved from the exploration of previous tasks (see Chapter 4).

- *Action value function space.* In Section 3.3.2, we discussed how the objective of generalization improvement can be achieved by biasing the hypothesis space. In case of batch RL, only the learning algorithm determines how the approximation of the optimal action-value function is computed. Therefore, a transfer learning should bias the value function space of the learning algorithm towards the a suitable space for the approximation of the value functions of the tasks in the environment. Since the learning algorithm learns on a given set of training samples, it is similar to a SL problem and it could be integrated with transfer solutions in SL (see Chapter 5).

- *Initialization.* As discussed in Section 3.3.2, the initialization of a learning algorithm can either result in an improvement in the learning speed or of the initial performance. The advantage of batch RL algorithm is that it is possible to map these two effects to the initialization of the sampling and learning algorithm, respectively. The initialization of the sampling algorithm means that the exploration strategy is biased towards a specific policy determined according to the knowledge retained from the solution of previous tasks. The expected result is an improvement in the learning speed, since the biased exploration policy should collect samples more informative than those collected by a random policy. On the other hand, the initialization of the learning algorithm, that is, the choice of an initial hypothesis, affects the initial performance of the agent.

Although batch RL received little attention as a framework for transfer in RL, the previous analysis shows how transfer objectives could be effectively achieved by biasing either the sampling or the learning algorithm.

Although many different algorithms can be designed for transfer in batch RL, in this thesis we focus on two aspects of transfer: the transfer of samples and the learning of features of function approximators.

According to the analysis in Section 3.3.3, these aspects have been addressed by very few works but in Chapter 4 and 5 we highlight the motivations for designing algorithms able to achieve transfer following these two perspectives. Furthermore, by focusing on sample transfer and on feature learning, we follow both the inductive transfer and the multi-task learning perspectives.

More specifically, the first transfer algorithm is aimed at the improvement in the learning speed from an inductive transfer perspective. The idea is to bias the output of the sampling algorithm by modifying the space of the sample sets

$$A_{Transfer} : \widehat{\mathcal{T}} \times \widehat{\mathcal{T}}^{n-1} \rightarrow \widehat{\mathbb{T}}. \tag{3.8}$$

That is, given a set of samples collected from a target task and $n-1$ sets of samples collected from source tasks, the transfer algorithm modifies the space of the sample sets by adding a subset of the source task samples. Under the assumption that some tasks are related in terms of domains and goals, the batch RL algorithm benefits from these additional samples and converges to a better approximation of the optimal value function even when a limited set of samples is actually collected from the target task.

The second transfer algorithm is meant to improve the generalization performance from a multi-task perspective. The idea is to change the hypothesis space of the learning algorithm by identifying the features shared across the task in the environment

$$A_{Transfer} : \widehat{\mathcal{T}}^{n} \rightarrow \mathbb{Q}. \tag{3.9}$$

Starting from a set of samples collected from n tasks, the transfer algorithm biases the hypothesis space towards functions that provide an accurate approximation of the optimal value functions of the tasks at hand, thus improving the generalization performance.

In the rest of the thesis, we restrict the set of transfer problems we focus on. In particular, we require all the tasks to share the same state and action space. This requirement may exclude some relevant applications (e.g., robotic problems in which sensors and/or actuators can change among different tasks). Nonetheless, the algorithms proposed in the next chapters can be extended to the general case of domain transfer if a mapping among the state and action spaces of the tasks is available (e.g., using inter-task mapping algorithms [144]). A recent work [140] proposed a mechanism for transferring samples from source to target tasks by mapping each sample of the source task into a sample defined in the state-action space of the target task.

4 Transfer of Samples in Batch Reinforcement Learning

In this chapter, we propose a technique to achieve the first objective of transfer, the improvement of the learning speed, in a batch RL algorithm. In particular, we introduce a method to transfer samples from a set of source tasks to a target task, thus reducing the complexity of the learning process in terms of the number of samples actually collected through direct interaction with the target task.

4.1 Introduction

It is widely recognized that the main objective of transfer in RL is to reduce the learning time. In fact, the solution of a set of *source* tasks can provide useful information about how to solve a related *target* task, thus reducing the amount of experience needed to learn its solution. For instance, if two tasks (source and target) share almost the same optimal policy, the solution of the source task can be used to bias the learning on the target task towards policies that are similar to the source task optimal policy. As a result, the algorithm will converge in a number of episodes smaller than that needed when learning from scratch. Similarly, other elements characterizing a learning problem could be transferred from source to target tasks, such as value functions, action spaces, reward functions, and so on. Nonetheless, when transferring from a source task that is not related to the target task, *negative* transfer may occur, that is, the learning algorithm may be slowed down because biased towards solutions that are completely different from the optimal one. Therefore, when defining a transfer algorithm for the improvement of learning speed, two critical aspects must be taken into account: *what* to transfer and *when* to transfer.

In this Chapter, we focus on a perspective that received very little attention so far, that is the transfer of experience in terms of samples (i.e., experience tuples of the form $\langle s, a, s', r \rangle$). We propose a mechanism that selectively transfers samples from source to target tasks on the basis of the likelihood (*compliance*) of source tasks with the samples collected

in the target task and that can be applied to any of the batch RL algorithms reviewed in Section 2.7. Furthermore, we introduce a criterion (*relevance*) to select from which source tasks samples should be transferred and, within a single source task, which samples are more likely to speed up the learning process.

Unlike the transfer of policies, the transfer of samples require only local similarity between tasks. In fact, even local and small variations to the task can lead to very different policies, thus preventing from transferring policies at all. On the other hand, samples in regions that are similar in source and target tasks can be always transferred independently from the similarity of their corresponding solutions. As a result, through sample transfer, we reduce the number of samples that must be actually collected in the target task to achieve convergence.

4.2 Background

In general, in RL algorithms, many aspects can affect the amount of experience needed to achieve convergence to the optimal or nearly optimal solution. The problem of improving learning speed is usually studied in RL under the perspective of the exploration/exploitation dilemma, that is, the problem of finding the optimal trade-off between high rewarding actions and explorative actions in order to achieve high performance in few learning episodes (Section 2.4.2). In particular, this problem has been framed within the PAC framework [62], that defines the problem of bounding the number of samples needed to achieve a ϵ-suboptimal solution with at least $1 - \delta$ probability. Although a thorough theoretical analysis is now available for a number of algorithms [62, 127, 125, 66, 30], it is still difficult to understand how different elements and parameters of traditional learning algorithms actually affect these bounds. In fact, complexity bounds in exploration/exploitation literature contain information about the state-action space, the maximum performance loss ϵ, the probability of being nearly-optimal δ, and the mixing time, but they rarely provide reliable estimations of the actual learning time of an algorithm in specific problems. Furthermore, at the moment, a rigorous theoretical analysis of the relationship between the learning speed and the techniques usually adopted to speedup RL algorithms (e.g., options, shaping rewards) is still lacking. On the other hand, there exists a large empirical evidence of the effectiveness of these techniques in improving the learning speed of RL algorithms and in Section 3.3.3 we reviewed how they are applied to transfer problems.

Most of these works identified the transfer of policies as the main technique to achieve effective transfer from source to the target task. The augmentation of an MDP with the inclusion of options "relevant" to learn the solution of the target task, leads to a significant bias to the exploration towards the goals of the options that can ease the learning. Similarly, the automatic shaping of the reward function [68] reduces the time needed to achieve high rewarding regions of the state space, thus reducing the learning time. Other approaches [48, 120] transfer value functions that can be used to initialize the learning algorithm to solutions near to the optimal one. Finally, a recent work [140] proposes a technique for the transfer of samples from one source task to a target task that share the same goal but are defined on different state-action spaces. It is interesting to notice that all these approaches affect the learning speed of the learning algorithm, but do not change the accuracy of the approximation of the optimal value function. This objective will be analyzed in Chapter 5.

Although these approaches study how the transfer of different elements from source to target tasks can impact on a RL algorithm, they often rely on the assumption that the tasks are somehow related and they do not address the problem of negative transfer [107]. Some works focused on the definition of relatedness measures between tasks that can be used to select from which source tasks it is actually convenient to transfer so as to avoid negative transfer. Carroll and Seppi [33] provide an experimental analysis of different similarity measures that estimate the expected speed-up on the basis of elements such as policy overlapping, Q-values, and reward structure. Unfortunately, it is often difficult to compute these measures before actually learning the solution of the target task and, thus, they are mainly used as a mean of analysis of the effectiveness of a transfer mechanism. In [113], although no relatedness measure is adopted, the loss of performance caused by the reduction of the action space of the target task to the optimal actions of the source task is bounded. Finally, in [47], different metrics for the distance between tasks are proposed and a theoretical analysis about how the distance between tasks is related to the difference between the performance of the corresponding optimal value functions is reported.

4.3 Motivating Example

Let us consider the domain shown in Figure 4.1, in which the general task is to hit a ball into a hole. The main factors characterizing the dynamics

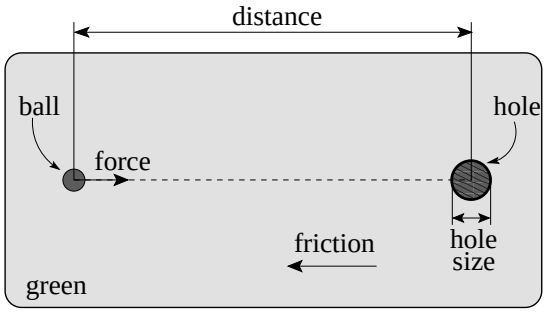


Figure 4.1: The golf problem. The agent hits the ball with a given force. Depending on the initial velocity of the ball, the friction of the ground and the size of the hole the ball can either enter in the hole, overcome the hole or remain at a given distance from the ball.

of the problem are the friction of the ball on the green and the size of the hole [1]. While the former impacts on the transition model, the latter affects the reward function and the termination of episodes. Any batch RL algorithm requires the sampling of samples $\langle s, a, s', r \rangle$ from the task at hand and the offline computation of an approximation of the optimal policy. Let us consider the case of a problem of goal transfer from one source to a target task, in which the friction is exactly the same and the size of the hole in the source task is greater than in the target task. Although there are regions of the state space in which the optimal policy of the two tasks could be different, a large amount of samples collected in the source task is exactly the same as those in the target task. In fact, since the dynamics does not change, all the samples in which the next state s' is not in or beyond the hole, are common to the source and to the target task. Therefore, if only few samples are collected from the target task, the batch RL algorithm can greatly benefit from re-using the samples collected in the source task, thus improving the accuracy of the approximation of the optimal policy and its corresponding performance. On the other hand, if we consider tasks in which the frictions are very different it may happen that the transfer of samples from source to target task worsens the performance of learning in the target task because of misleading samples. Therefore, it is important to design a mechanism able to identify which samples are actually informative for the target task so as to improve the learning performance.

[1] A more detailed description of the dynamics of the problem is reported in Section 4.5.

4.4 Transfer of Samples in Batch Reinforcement Learning

In this section, we outline a novel mechanism for sample transfer.

4.4.1 Problem Formulation

As illustrated in Figure 3.6, in a batch RL algorithm, the main element that can directly affect the learning speed is the set of samples. Here, we focus on the way the set of the samples used to feed the learning algorithm can be augmented by the inclusion of samples drawn from a set of source tasks. The basic intuition underlying this idea is that, since the tasks are related through the task distribution Ω, some of the source tasks are likely to contain samples similar to samples of the target task. Therefore, a batch RL algorithm could greatly benefit from additional samples extracted from the source tasks when learning the solution of the target task. Under this perspective, we define the *complexity* of a batch RL algorithm as the sample complexity of the sampling algorithm, that is, the number of samples actually drawn from the task at hand through direct interaction.

Formally, we define this transfer algorithm as a mapping from the samples collected from source and target tasks to an augmented set of samples for the target task

$$A_{Transfer} : \widehat{\mathcal{T}} \times \widehat{\mathcal{T}}^{n-1} \rightarrow \widehat{\mathbb{T}}, \tag{4.1}$$

where n is the total number of tasks at hand (one target task and $n-1$ source tasks), $\widehat{\mathcal{T}}$ is the space of the sets of samples, and $\widehat{\mathbb{T}}$ is the family of all the possible sample sets spaces $\widehat{\mathcal{T}}$. Before transfer, the space of sample sets $\widehat{\mathcal{T}} \in \widehat{\mathbb{T}}$ contains all the possible sets of samples that can be obtained from the target task T. When the transfer takes place, the new space of sample sets $\widehat{\mathcal{T}}'$ augments the samples sets in $\widehat{\mathcal{T}}$ with a given set of samples that are transferred from the source tasks. Thus, given the set of samples collected from the target task $\widehat{T} \in \widehat{\mathcal{T}}$ and the $n-1$ sets of samples of the source tasks $\widehat{S}_k \in \widehat{\mathcal{T}},\ k \in \mathbb{N}_{n-1}$, the objective is to bias the space of the sets of samples by building a set of samples $\widetilde{T} \in \widehat{\mathcal{T}}$ obtained by transferring samples from the source tasks to the target task. Samples in $\widetilde{T}$ are finally used as input for the learning algorithm. It is worth noting that the transfer algorithm that follows is completely independent from the specific learning algorithm used to compute the approximation of the optimal action-value function.

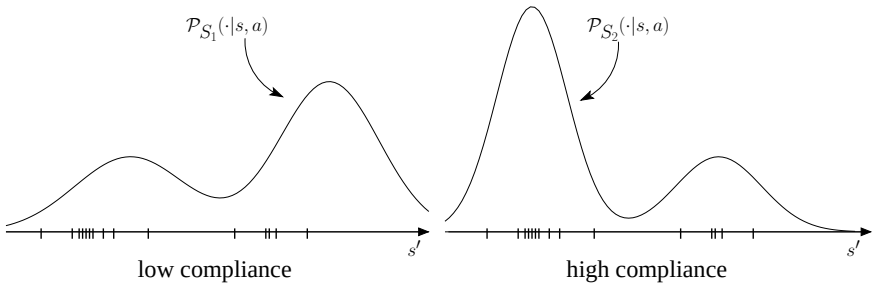


Figure 4.2: Comparison of the probability of two source tasks (S_1 and S_2) to be the models from which the samples of target task are drawn. Here we report only the transition model, that is the probability distributions $\mathcal{P}_{S_1}(\cdot|s,a)$ and $\mathcal{P}_{S_2}(\cdot|s,a)$, the tics on the x-axis represent the samples collected in state-action pair (s,a) from the target task T.

4.4.2 Task Compliance

The main problem of transferring samples from one task to another is to reduce the probability negative transfer, that is the transfer of samples drawn from a source task that is significantly different from the target task. In order to avoid negative transfer, we need to identify which source tasks are more likely to have samples similar to those available for the target task.

Alternatively, this problem can be stated as a *model identification problem*. Let us consider the following scenario. The task space $\mathcal{T}$ contains n tasks, and m samples have been already collected from each task. Let T be a new task drawn according to Ω and $\widehat{T}$ the set of samples collected from it, with $|\widehat{T}| = t \ll m$. Since the transfer of samples from all the tasks in $\mathcal{T}$ can make the learning performance on T worse, we need to identify which of the previously solved tasks is actually T according to the available samples. Starting from a uniform prior over the tasks in $\mathcal{T}$, we compute the posterior distribution as the probability of a task to be the model from which samples in $\widehat{T}$ are drawn. As the number of samples t increases, the posterior distribution is updated accordingly until the probability is concentrated only on the task equal to T. Then, the m samples previously collected in the task equal to T can be added to $\widehat{T}$ and used to feed the batch RL algorithm, thus improving its learning performance.

In the case in which $\mathcal{T}$ is infinite or contains many tasks, the probability to have one source task identical to the target task is negligible, but

we can still identify the source tasks that are more likely to generate samples similar to those of the target task. Bayesian models in RL [36, 128] usually define the likelihood of a model S as the joint probability of S given the samples in $\widehat{T}$. Under the assumption that all the samples are independent, this results in the product of the likelihood $P\left(S|\widehat{T}_{\langle s,a\rangle}\right)$ in all the state-action pairs. Unfortunately, this perspective is not suitable for our scenario. Let us consider the case in which all the samples in $\widehat{T}$ but one perfectly fit S, that is there exists one sample τ_i for which the compliance λ_i is very low. As a result, the global likelihood is very low. Since our objective is to identify the source task that is more likely to contain samples that is worth transferring, if a task S contains most of the samples with high-compliance and only one low-compliant sample, the transfer from S should take place. Therefore, instead of the probability of a source task to generate *all* the samples collected in the target task, we compute its *compliance* with T by averaging the probability of generating the samples in $\widehat{T}$. Then, we transfer samples of source tasks proportionally to their compliance with the target task.

In Figure 4.2 we report a qualitative scenario with two source tasks S_1 and S_2 and a set of samples collected from T in a generic state-action pair $\langle s, a\rangle$. The two distributions represent the probability to achieve a state s' taking a in s. As it can be noticed, S_2 is more likely to have generated the samples than S_1.

Let us consider a generic source task S and the samples extracted from the target task $\widehat{T}$. Given a state-action pair $\langle s, a\rangle$, the probability of S to be the model from which the samples of the target task in $\langle s, a\rangle$ are extracted, that is the likelihood of the model, can be computed by applying Bayes theorem as

$$\begin{aligned} P\left(S|\widehat{T}_{\langle s,a\rangle}\right) &\propto P\left(\widehat{T}_{\langle s,a\rangle}|S\right)P\left(S\right) \\ &= \prod_{\tau_i \in \widehat{T}_{\langle s,a\rangle}} P\left(\tau_i|S\right)P\left(S\right) \\ &= \prod_{\tau_i \in \widehat{T}_{\langle s,a\rangle}} \mathcal{P}_S(s'_i|s_i,a_i)\mathcal{R}_S(r_i|s_i,a_i)P\left(S\right), \end{aligned} \quad (4.2)$$

where $\widehat{T}_{\langle s,a\rangle}$ is the set of samples $\{\tau_i \in \widehat{T} | \langle s_i, a_i\rangle = \langle s, a\rangle\}$ and $P\left(S\right)$ is a *prior* on the source task S. Probability $P\left(S|\widehat{T}_{\langle s,a\rangle}\right)$ is a *posterior* distribution over the source tasks in $\langle s, a\rangle$.

Unfortunately, the posterior probability cannot be immediately computed, since we do not have the exact model of the source task S. On the

other hand, we have a set of m samples $\widehat{S}$ previously collected in S, from which an approximation of the continuous model can be computed. In the following, with an abuse of notation, with $\widehat{T}$ and $\widehat{S}$ we denote both the sets of samples and the model approximations. Let $\tau_i = \langle s_i, a_i, s'_i, r_i \rangle$ be a generic transition sample in $\widehat{T}$, the probability of this transition to be generated by S given the set of samples $\widehat{S}$ is

$$P(\langle s_i, a_i, s'_i, r_i \rangle | \widehat{S}) = \mathcal{P}_{\widehat{S}}(s'_i | s_i, a_i) \mathcal{R}_{\widehat{S}}(r_i | s_i, a_i), \tag{4.3}$$

where $\mathcal{P}_{\widehat{S}}$ and $\mathcal{R}_{\widehat{S}}$ are respectively the approximated transition model and reward function. In discrete tasks the transition model can be easily approximated using the maximum likelihood estimator obtained by counting the number of occurrences of the transitions. In the general case of tasks with continuous state and action spaces, this approach cannot be adopted because the probability to have many samples in the same state-action pair $\langle s, a \rangle$ is negligible. Therefore, in order to estimate the probability of a transition in a state-action pair, it is necessary to generalize the maximum likelihood approach by taking into consideration all the samples close to $\langle s, a \rangle$. In particular, we build on the kernel-based solution proposed in [60, 58]. The estimation of the probability of a transition τ_i to occur depends on all samples $\sigma_j \in \widehat{S}$. The contribution of σ_j to the probability of τ_i is divided in two parts: *(i)* the closeness of the two samples in the state-action space, *(ii)* the similarity of the outcome (either next state s' or reward r). The closeness of two samples can be computed by applying a kernel function $\varphi(\cdot)$ to a given distance metric d defined on $\mathcal{S} \times \mathcal{A}$. More specifically, we adopt a Gaussian kernel $\varphi(\cdot) = \exp\left(\frac{\|\cdot\|}{\delta}\right)^2$, where δ is the bandwidth of the kernel (i.e., the standard deviation of the Gaussian).

First of all, we define the similarity (*compliance* in the following) between the trajectory of τ_i and the trajectories of samples $\sigma_j \in \widehat{S}$ in terms of dynamics and reward. We define the compliance of τ_i with respect to σ_j for the transition model as

$$\lambda^{\mathcal{P}}_{ij} = w_{ij} \cdot \varphi\left(\frac{d(s'_i, s_i + (s'_j - s_j))}{\delta_{s'}}\right),$$

where

$$w_{ij} = \frac{\varphi\left(\frac{d(\langle s_i, a_i \rangle, \langle s_j, a_j \rangle)}{\delta_{sa}}\right)}{\sum_{l=1}^{m} \varphi\left(\frac{d(\langle s_i, a_i \rangle, \langle s_l, a_l \rangle)}{\delta_{sa}}\right)}.$$

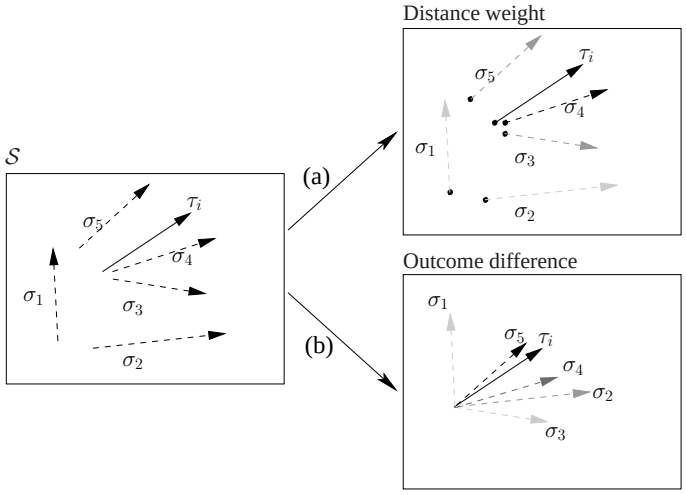


Figure 4.3: Computation of the approximated transition model of a task S for the transition of a sample τ_i according to the samples in $\widehat{S}$. We consider six samples $\sigma_j \in \widehat{S}$ such that $a_j = a_i$. The transition probability is obtained by weighting the difference in the outcome (b) by the distance between the samples in the state space (a). The darker the arrow the higher the weight (value).

The first term weighs the difference between the outcome of taking action a_i in s_i and the outcome of a_j taken in s_j. Instead of a simple comparison of s'_i and s'_j, the second element computes the distance between the state achieved in τ_i and the state that it would be achieved if a_j were taken in s_i. As shown in the example of Figure 4.3, σ_5 gets a relatively small weight because of its distance from the state-action pair of τ_i (Figure 4.3-(a)) but the outcome of action a_j is almost the same as a_i (Figure 4.3-(b)). On the other hand σ_3 is next to τ_i but its outcome is completely different. As a result, compliance $\lambda^{\mathcal{P}}_{i3}$ is less than $\lambda^{\mathcal{P}}_{i5}$.

Similarly, the compliance for the estimation of the reward function is defined as

$$\lambda^{\mathcal{R}}_{ij} = w_{ij}\varphi\left(\frac{|r_i - r_j|}{\delta_r}\right).$$

where $d(r_i, r_j)$ is the difference between the two rewards. The approximated transition model and the reward model are the average of the

compliance between τ_i and all the samples in $\widehat{S}$

$$\mathcal{P}_{\widehat{S}}(s'_i|s_i,a_i) = \frac{1}{Z^{\mathcal{P}}}\sum_{j=1}^{m}\lambda_{ij}^{\mathcal{P}};\;\; \mathcal{R}_{\widehat{S}}(r_i|s_i,a_i) = \frac{1}{Z^{\mathcal{R}}}\sum_{j=1}^{m}\lambda_{ij}^{\mathcal{R}},$$

where $Z^{\mathcal{P}}$ and $Z^{\mathcal{R}}$ are normalization terms [2].

Finally, we define the compliance of a sample τ_i to a source task S approximated using its samples in $\widehat{S}$ as

$$\lambda_i = P(\langle s_i,a_i,s'_i,r_i\rangle|\widehat{S}) = \frac{1}{Z^{\mathcal{P}}Z^{\mathcal{R}}}\left(\sum_{j=1}^{m}\lambda_{ij}^{\mathcal{P}}\right)\left(\sum_{j=1}^{m}\lambda_{ij}^{\mathcal{R}}\right).$$

Recalling the probability in (4.2), given the compliance of each sample in $\langle s,a\rangle$, the likelihood of the model in $\langle s,a\rangle$ becomes

$$P\left(S|\widehat{T}_{\langle s,a\rangle}\right) \propto \prod_{\tau_i\in\widehat{T}_{\langle s,a\rangle}} \lambda_i P(S). \tag{4.4}$$

Starting from the probability of the model in each state-action pair, we compute a global measure of the probability for the task to contain samples similar to those in the target task. We define the compliance of a task S as the average likelihood computed over each state-action pair experienced in the target task.

Definition 4.1 *Given the set of samples of the target task $\widehat{T}$ and of a source task $\widehat{S}$, the* task compliance *of S with the target task samples is*

$$\Lambda = \frac{1}{|\widehat{U}|}\sum_{\langle s,a\rangle\in\widehat{U}} P\left(S|\widehat{T}_{\langle s,a\rangle}\right), \tag{4.5}$$

where $\widehat{U}$ contains all the distinct state-action pairs in the samples of $\widehat{T}$.

Since the probability to have two samples in the very same state-action pair is negligible, we can consider $|\widehat{U}| = |\widehat{T}| = t$ and the definition of task compliance reduces to

$$\Lambda = \frac{1}{t}\sum_{i=1}^{t}\lambda_i P(S), \tag{4.6}$$

where $P(S)$ is a prior on the source task. When n source tasks with m samples each are available, and t samples are collected from T, the computation of the task compliance has a time complexity of $\Theta(nmt)$.

[2] For the transition model the normalization term Z is the integral of the kernel function on the state space , while for the reward function is the integral on $\mathbb{R}$.

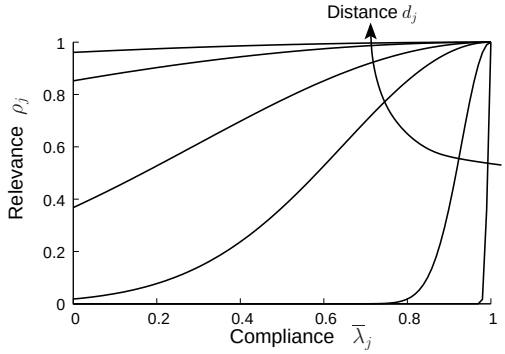


Figure 4.4: The relevance function ρ_j for different values of average distance d_j.

4.4.3 Sample Relevance

Although the measure of compliance is effective in identifying which sources, in average, are more convenient to transfer samples from, it does not provide any suggestion about which samples in $\widehat{S}$ are actually better to transfer. In the following, we introduce the concept of *relevance* of each sample $\sigma_j \in \widehat{S}$. The idea is to use the compliance of σ_j with the target task. Unfortunately, in this case, the measure of compliance is often unreliable because of a poor approximation of the target task. In fact, while each source task contains m samples, only $t \ll m$ samples are available for the target task. As a result, it may happen that the compliance of a sample σ_j is computed according to samples τ_i that are significantly far in the state-action space. Therefore, we need a formulation of relevance strictly related to the compliance whenever the number of samples in $\widehat{T}$ close to σ_j is sufficient, while tending to a default value when the compliance is not reliable.

Given the definition of compliance $\lambda_{ji}^{\mathcal{P}}$ and $\lambda_{ji}^{\mathcal{R}}$ of σ_j with a sample τ_i, the compliance of σ_j with the approximated model of the target task $\widehat{T}$ is

$$\lambda_j = P(\sigma_j|\widehat{T}) = \frac{1}{Z^{\mathcal{P}} Z^{\mathcal{R}}} \left(\sum_{i=1}^{t} \lambda_{ji}^{\mathcal{P}} \right) \left(\sum_{i=1}^{t} \lambda_{ji}^{\mathcal{R}} \right). \tag{4.7}$$

Let the samples τ_i be sorted in descending order according to the compliance λ_{ji}. Given the distance between the two samples in the state-action space $d(\langle s_j, a_j \rangle, \langle s_i, a_i \rangle)$, we compute the average distance

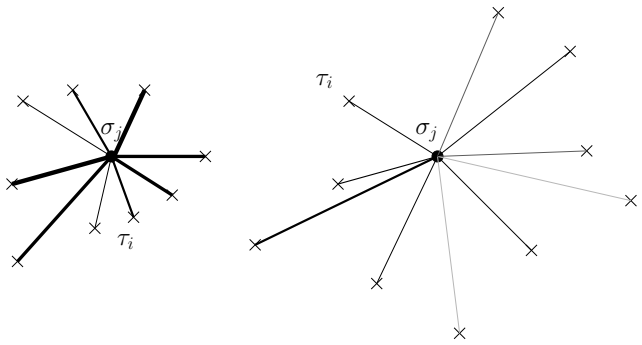


Figure 4.5: Two configurations in which sample σ_j has high relevance. The samples are represented in the state-action space and the thickness of the lines represents the compliance λ_{ji}. In the first case, there is a number of samples τ_i near to σ_j with high compliance, thus the relevance function returns a high value. In the second case, although the compliance is very low, the samples are very far from σ_j, thus its relevance is high by default.

between σ_j and the samples $\tau_i \in \widehat{T}$ as

$$d_j = \frac{1}{h_j} \sum_{i=1}^{h_j} d(\langle s_j, a_j \rangle, \langle s_i, a_i \rangle), \tag{4.8}$$

where h_j is such that $\sum_{i=1}^{h_j} w_{ji} < \mu$, where $\mu \in (0; 1]$ determines the fraction of the total number of samples considered in the computation of the average distance. The value of fraction μ is not particularly critical for the performance of the system and it is used only to limit the samples considered in the computation of the average distance.

Definition 4.2 *Given the compliance λ_j and the average distance d_j, the* relevance *of σ_j is defined as*

$$\rho_j = \rho(\overline{\lambda}_j, d_j) = e^{-\left(\frac{\overline{\lambda}_j - 1}{d_j}\right)^2}, \tag{4.9}$$

where $\overline{\lambda}_j$ is the compliance normalized over all the samples in $\widehat{S}$.

The relevance function is shown in Figure 4.4 for different values of distance d_j. As it can be noticed, sample σ_j may have a high relevance in two distinct cases (Figure 4.5). In the first case, σ_j can be highly relevant

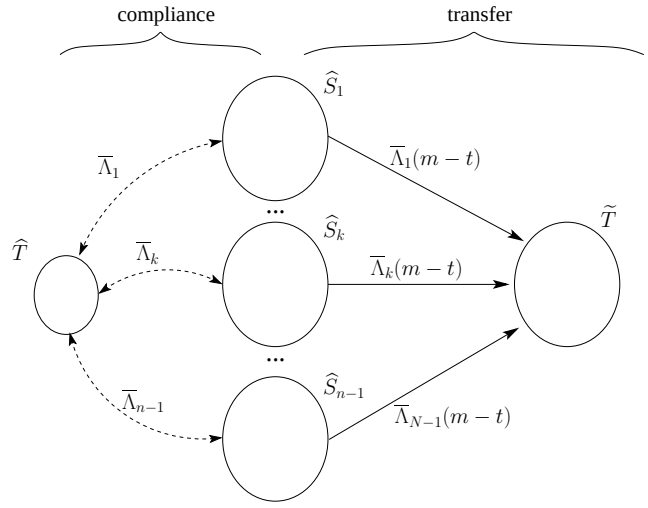


Figure 4.6: Sample transfer process. After the computation of the compliance Λ_k of the source task S_k with the samples available in $\widehat{T}$, $\overline{\Lambda}_k(m-t)$ samples are drawn with a probability proportional to their relevance from $\widehat{S}_k$ and transferred to the set of samples $\widetilde{T}$ used to feed the batch learning algorithm.

because of a number of close samples τ_i which it is compliant with. On the other hand, in case there are no close samples, independently from the compliance, we assume a high relevance for sample σ_j. The assumption underlying the definition of relevance is that, whenever there is no evidence against the transfer of a sample, it is convenient to transfer it to the target task. During the transfer process, the probability of a sample σ_j to be transferred to $\widetilde{T}$ is proportional to its relevance ρ_j.

4.4.4 Task Transfer

Given the definition of compliance of a source task with the samples collected from the target task, we need to define a suitable mechanism to choose the source tasks from which the samples are actually transferred. For sake of simplicity, we bound the number of samples used by the learning algorithm to m. Since $|\widehat{T}| = t$ samples are already available in $\widehat{T}$, $m-t$ samples need to be extracted and transferred from the source tasks, so as to obtain a set of samples $\widetilde{T}$ that contains the samples actually collected from T and the samples transferred from the source tasks. The most straightforward criterion is to draw all the $m-t$ samples from the

Algorithm 8 The sample transfer algorithm

Input: source tasks $\{S_k\}_{k\in\mathbb{N}_{n-1}}$, target task T
Parameters: bandwidth δ_{sa}, $\delta_{s'}$, δ_r, t number of samples of task T, m number of samples of tasks S_k
Output: transferred sample set $\widetilde{T}$

for $k = 1$ to $n-1$ **do**
  Collect m samples from source task S_k and form the sample set $\widehat{S}_k$
end for
Collect t samples from the target task T and form the sample set $\widehat{T}$
for $k = 1$ to $n-1$ **do**
  Compute compliance Λ_k of source task S_k with samples $\widehat{T}$
  Compute relevance ρ_j of samples $\sigma_j \in \widehat{S}_k$
  Sample $\overline{\Lambda}_k(m-t)$ samples from $\widehat{S}_k$ with probability proportional to ρ_j
end for
Put all the additional samples in $\widehat{T}$ and form the transferred sample set $\widetilde{T}$

most compliant source task. Thus, the probability to extract samples similar to those in the target task is maximized. Another criterion is to draw the $m-t$ samples from all the source tasks proportionally to their compliance Λ_k as shown in Figure 4.6.

If transfer occurs only on the basis of the task compliance, there is no theoretical reason to actually prefer one criterion over the other (an empirical comparison is reported in Section 4.5.3). However, if we consider the region transfer mechanism based on sample relevance introduced in the previous section, there are situations in which transferring only from the most compliant task is not efficient. Let us consider an environment in which each source task has only one specific region similar to the target task while it is significantly different elsewhere and all the tasks have similar compliance (an example of this scenario is reported in Section 4.6). In this case the transfer from the most compliant task would transfer only samples from a limited region of the state-action space. On the other hand, the proportional criterion would equally draw samples from all the tasks. As a result, set of samples $\widetilde{T}$ would contain samples from many different regions, thus significantly improving the learning performance.

Therefore, in the general case, the transfer process is as follows. For each source task S_k, the number of samples transferred to the sample set $\widetilde{T}$ of the new target task is proportional to its normalized compliance $\overline{\Lambda}_k = \frac{\Lambda_k}{\sum_{l=1}^{n}\Lambda_l}$. Then, in each source task, samples are drawn according to their relevance, thus avoiding to transfer samples that are quite dissimilar

from those in the target task. The whole process of sample transfer is summarized in Algorithm 8.

It is worth noting that the number of samples m used to feed the learning algorithm constrains the fraction of transferred samples in $\widetilde{T}$. While this constraint could limit the effectiveness of the transfer, it assures that as the number of samples t collected from the target task increases, the use of source samples decreases and when $t = m$ the performance depends only on samples from the target task. As a consequence, the value of m is chosen to balance the use of source and target samples. While low values of m lead to exploit the target samples at most, thus possibly reducing the advantage of transfer, high values of m make the transferred samples more relevant. In the experiments, m is usually set to the number of samples needed by the learning algorithm to achieve convergence when using samples coming only from the target task.

4.5 Experiments: the Golf Problem

4.5.1 Definition and Settings

The first domain we consider is a variant of the mini golf game (Figure 4.1) [72]. The agent has to shoot a ball inside a hole with the minimum number of strokes. Given the distance x_0 of the ball from the hole (x_0 assumes value in $[-2000; 0]cm$), the agent must determine the initial velocity to put the ball in the hole in one stroke. Let $d_{ball} = 4.5cm$ and d_{hole} be the diameters of the ball and of the hole and f the coefficient of friction between the ball and the ground. The agent selects the initial velocity a in a limited interval $[100; 200]cm/s$ discretized by $20cm/s$. The chosen velocity a is perturbed by a uniform noise $U[-5; 5]$. The resulting dynamics is

$$x_f = x_0 + v_0 t - 0.5 f t^2, \tag{4.10}$$

where x_0 is the initial position, $v_0 = a + U[-5; 5]$ is the initial velocity applied to the ball and $t = \frac{v_0}{f}$ is the time before the ball stops. If the final position x_f is greater than 0, we need to verify whether, when the ball arrived at $x = 0$, its velocity is low enough to let the ball fall in the hole. The maximum velocity the ball can have to fall in the hole when it arrives at $x = 0$ is

$$v^{max} = \frac{d_{hole}}{\sqrt{\frac{d_{ball}}{2}\frac{2}{g}}}. \tag{4.11}$$

Parameter	Value
m	1000
μ	0.8
δ_{sa}	5.0
δ_r	0.1
$\delta_{s'}$	10.0

Table 4.1: Parameters for sample transfer used in all the experiments in the golf problem.

Transfer	Source Task	Definition
Goal	S_1	$f = 40cm/s^2 \quad d_{hole} = 11.5cm$
Dynamics	S_2	$f = 70cm/s^2 \quad d_{hole} = 8.0cm$
Domain	S_3	$f = 80cm/s^2 \quad d_{hole} = 9.0cm$
Domain	S_4	$f = 20cm/s^2 \quad d_{hole} = 6.0cm$

Table 4.2: Source tasks used to analyze single-source sample transfer.

As a result, the minimum and maximum initial velocity to make the ball to enter in the hole are $v_0^{min} = \sqrt{2gkx_0}$ (the ball stops exactly at the hole position) and $v_0^{max} = \sqrt{2gkx_0 + (v^{max})^2}$ (the velocity allows the ball to fall in the hole), where g is the universal constant of gravity. At the beginning of each episode the ball is placed at random, between $20cm$ and $2000cm$ far from the hole. When the ball enters the hole the episode ends with reward 1. If $v_0 > v_0^{max}$, the ball is lost and the episode ends with reward -10. Finally, if $v_0 < v_0^{min}$ the episode goes on and the agent can try another stroke with reward -1. Since the problem is episodic we set the discount factor γ to 1.0. The performance of the learned policy is tested on a finite set of 50 states evenly distributed between $-2000cm$ and $-10cm$.

In the following experiments, we use Gaussian kernels and Mahalanobis distance (see Section 4.4.2) and the parameters summarized in Table 4.1. The results are obtained by averaging 100 runs. In FQI, we use extra-randomized trees [41] with 50 trees, 2 random splits, and 30 minimum sample size for each node, trained on 20 iterations. Samples are obtained through random sampling run on independent episodes of maximum 10 steps each.

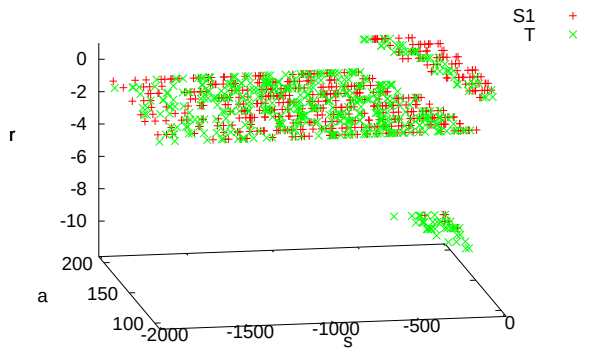


Figure 4.7: The $\langle s, a, r \rangle$ part of samples from T and S_1.

4.5.2 Results: One Source Task Transfer

In this section, we analyze the results obtained by transferring samples from one source task to a target task. The following experiments are meant to illustrate the effectiveness of sample transfer when tasks are similar and to show the negative effects of transfer when tasks are significantly different. We consider a space of tasks $\mathcal{T} = \{T, S_1, S_2, S_3, S_4\}$ and a uniform distribution Ω. The target task T has a friction $f = 40cm/s^2$ and a hole size $d_{hole} = 8cm$, while the different source tasks are summarized in Table 4.2 with the corresponding type of transfer. Tasks S_1, S_2, S_3 has higher friction and bigger holes but are relatively similar to T and we expect positive effect from the transfer of samples in terms of an improved initial performance and a reduction of the number of samples collected from the target task in order to obtain a nearly-optimal performance. On the other hand, task S_4 has a friction much less than that of the target task and the hole has a smaller diameter. Therefore, samples from S_4 are likely to cause negative effects on the learning performance in the target task.

In order to clarify the reasons for either positive or negative transfer, before studying the learning performance, we analyze the differences between the samples coming from different tasks. In particular, we compare the samples of T, S_1 and S_4. In the case of transfer from S_1, the transition model does not change (the friction is the same as in T) and only the size of the hole is different. As a result, most of the samples collected from S_1 can be profitably reused in T. However, a larger hole may cause a significantly different outcome when the ball reaches the

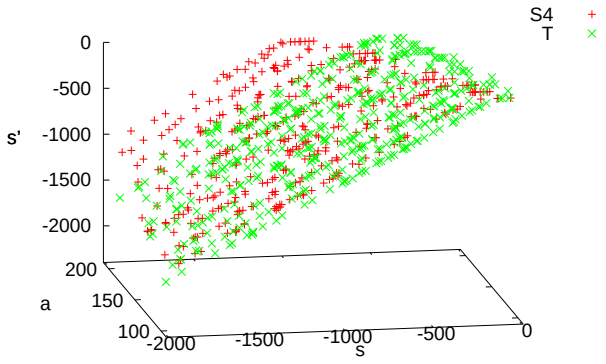


Figure 4.8: The $\langle s, a, s' \rangle$ part of samples from T and S_4.

hole with the limit velocity v^{max}. In fact, as shown in Figure 4.7, while in S_1 this results in a positive reward, in T the agent receives a negative reward of -10. Nonetheless, even few samples with negative reward from T are enough to counter-balance the effect of the transfer of "wrong" positive samples from S_1. Furthermore, most of the samples are exactly the same in the two tasks and, thus, the learning algorithm is likely to greatly benefit from the additional samples, in particular at the beginning of the learning process. On the other hand, we expect samples from S_4 to cause negative transfer. As shown in Figure 4.8 the dynamics of samples in S_4 is completely different from T because of different friction. In this case, there is no sample that provides useful information about the target task. Furthermore, the optimal policy in S_4 selects low-velocity actions because of the low friction and this leads to accumulate negative rewards when used in T. As a result, the transfer of samples in S_4 is likely to worsen the learning performance in T.

Figure 4.9 shows the performance obtained in the target task when either *original* or *transferred* samples are used. As it can be noticed, transferring samples from S_1 obtains the best performance. In fact, the difference between S_1 and T is limited to few samples of the goal region and this does not lead to any worsening of the performance. At the beginning of the learning process, the performance is significantly improved thanks to the samples from S_1, while as the learning progresses the number of samples from T becomes more relevant with respect to the transferred samples (when $t = m$, no transfer occurs), thus allowing the learning to achieve the optimal policy.

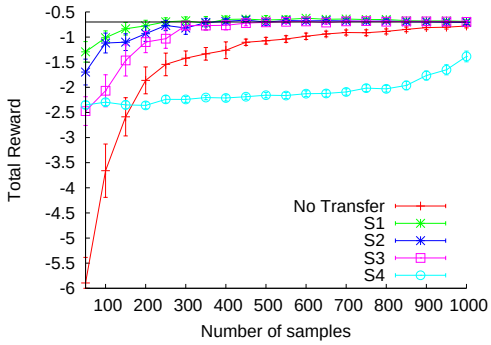
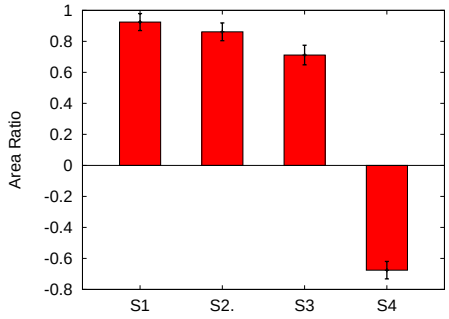


Figure 4.9: Total reward in the Golf problem with or without transfer from one source task.



Transfer	Area Ratio
$S_1 \rightarrow T$	$92.4\% \pm 5.47$
$S_2 \rightarrow T$	$86.1\% \pm 5.73$
$S_3 \rightarrow T$	$71.1\% \pm 6.31$
$S_4 \rightarrow T$	$-67.5\% \pm 5.62$

Figure 4.10: Area ratios in the Golf problem for transfer from one source task.

Since in task S_2, the goal (the hole size) is the same, but the friction is greater, the policy initially learned on the transferred samples uses velocities that are greater than the optimal ones in the target task and the risk is to go beyond the hole and get a negative reward. Similarly to transfer from S_1 this difference does not actually lead to negative because it is counter-balanced by samples from T. Although samples collected from task S_3 are different both in transition model and reward function, the transfer still leads to a significant improvement to the learning performance. As a result, transfer from both S_2 and S_3 is successful and reduces the learning time. On the other hand, the effects of transfer from S_4 is completely different. In fact, S_4 greatly differs from T both in transition model and reward function and almost no sample is useful for learning the optimal policy of the target task.

In order to evaluate the improvement in terms of learning speed, we analyze the cumulative reward in all the four configurations. Although other measures of learning speed can be used (e.g., the number of samples before achieving a desired performance), this measure takes into account the global performance of the learning algorithm and it is often used to compare different exploration strategies [126]. More precisely, we consider the area ratio measure introduced in [144]:

$$r = \frac{\text{area of curve w/ transfer} - \text{area of curve w/o transfer}}{\text{area of curve w/o transfer}}, \quad (4.12)$$

where we measure the area of the learning curve as the area between the curve and the convergence value. Figure 4.10 shows the area ratio of sample transfer from the source tasks. As it can be noticed, transfer from S_1, S_2, S_3 significantly improves the global performance of the learning algorithm, while S_4 causes negative transfer.

This illustrative experiment shows that, depending on the structure of the source task, the transfer of samples can result in either positive or negative transfer. It is interesting to notice that in case of single source transfer, the performance obtained by transferring samples is conceptually equivalent to the transfer of policies in the case of online learning. In fact, the improvement in the learning speed is mainly due to a better initial performance strictly related to the performance of the optimal policy of the source task when transferred to the target task. Since at the beginning of the learning process almost all the samples in $\widetilde{T}$ are drawn from the source task and only a few are actually collected from the target task, the learned policy is exactly the optimal one for the source task. As the learning progresses the number of transferred samples decreases and the performance gets similar to that of learning on target task samples. On the other hand, whenever the source tasks are more than one and, in particular, when the region transfer is adopted (see Section 4.6.2), the sample transfer technique is radically different from policy transfer, since samples are obtained from different regions of different tasks and, thus, the result is not the same of policy transfer from one single source.

Finally, we report the compliance of S_1 with the samples in $\widehat{T}$ together with the confidence ($p = 0.01$) averaged over 10 runs (Figure 4.11). As it can be noticed, after about 200 samples, the value of the compliance is almost constant. Therefore, for $t > 200$ the compliance becomes reliable, thus allowing to determine whether the transfer will be effective or not. While with only one source task, the compliance could be used only as an indirect measure of the expected speed-up (the higher the compliance the higher the expected speed-up), when multiple source tasks are available,

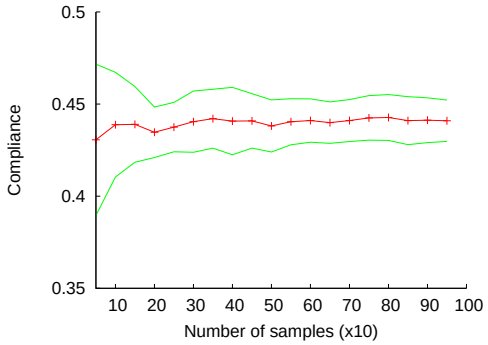


Figure 4.11: Compliance in the Golf problem of the source task S_1 with the target task T.

Transfer	Source Task	Definition
Goal	S_1	$f = 40cm/s^2 \quad d_{hole} = 7.5cm$
Dynamics	S_2	$f = 42.5cm/s^2 \quad d_{hole} = 8.0cm$
Domain	S_3	$f = 15cm/s^2 \quad d_{hole} = 8.0cm$
Domain	S_4	$f = 20cm/s^2 \quad d_{hole} = 4.5cm$
Domain	S_5	$f = 80cm/s^2 \quad d_{hole} = 6.0cm$
Domain	S_6	$f = 10cm/s^2 \quad d_{hole} = 3.5cm$

Table 4.3: Source tasks used to analyze n-source sample transfer.

it can be used to identify from which task is more convenient to transfer samples.

4.5.3 Results: n-Source Task Transfer

In this section, we analyze whether the task compliance is effective in avoiding the transfer of samples from tasks that are dissimilar from the target task and, at the same time, identify those which are more convenient to transfer from. We consider a task space $\mathcal{T} = \{T, S_1, S_2, S_3, S_4, S_5, S_6\}$ and a uniform distribution Ω. As summarized in Table 4.3, we use tasks with different changes with respect to the target task. In particular, S_1 and S_2 are very similar to the target task, while S_3, S_4, S_5, S_6 significantly differ from T in both the transition model and the reward function.

Besides the analysis of the task compliance, we also report the comparison between three different transfer policies: *best*, *proportional*, *random*.

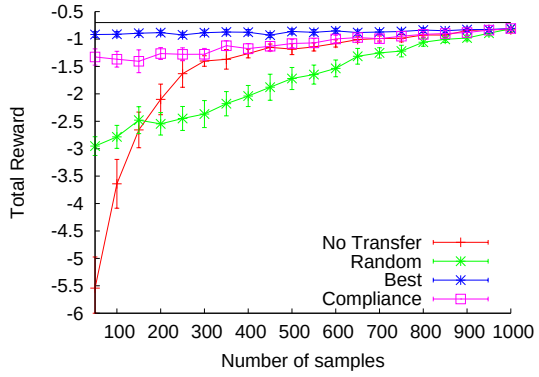
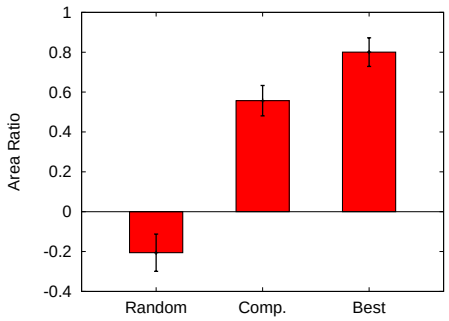


Figure 4.12: Total reward in the Golf problem with or without transfer from n source tasks.



Transfer	Area Ratio
Random	$-20.5\% \pm 9.33$
Best	$80.0\% \pm 7.63$
Comp.	$55.6\% \pm 7.18$

Figure 4.13: Area ratios in the Golf problem with or without transfer from n source tasks.

In the first case, the transfer occurs only from the most compliant source task, while in the second case, samples are extracted from all the tasks proportionally to their compliance (see Figure 4.6). Finally, with the *random* policy, the compliance is ignored and tasks are chosen at random.

Figure 4.12 shows the learning performance on the target task and the three transfer algorithms. The transfer of samples from the most compliant task obtained the best performance with a significant speed-up with respect to the version without transfer (Figure 4.13). Also the transfer proportional to the task compliance achieved a good result, with an improvement of the $55.6\% \pm 7.18$ of the performance without transfer. On the other hand, the random sampling from all the tasks obtained a very poor performance caused by the presence in $\mathcal{T}$ of many tasks whose

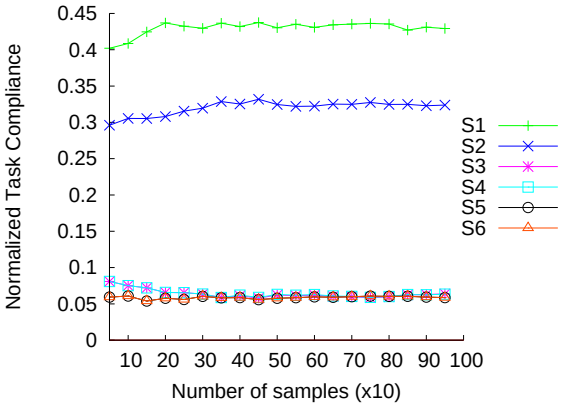


Figure 4.14: Compliance of different source tasks as the number of samples in $\widehat{T}$ increases.

samples lead to negative transfer.

In Figure 4.14, we report the normalized task compliance for each task, as the number of samples collected from T increases. After few tens of samples the compliance stabilizes and the transfer algorithm successfully identifies that the most compliant task is S_1, while $S_3, \ldots, S_6$ have a very low compliance. Finally, S_2, that is pretty similar to the target task, has an intermediate value of compliance.

4.6 Experiments: the Car on the Hill Problem

4.6.1 Definition and Settings

The second problem we consider is the Car on the Hill problem (or mountain car problem), in which a car must achieve the top of a hill in the minimum time. The dynamics model we adopt is the same as in [41]. Given the two state variables, position p and velocity v, the shape of the hill $Hill(p)$, and a control variable $a = \{-4, 4\}$, the dynamics of the car is:

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= \frac{a}{M(1 + Hill'(p)^2)} - \frac{gHill'(p)}{1 + Hill'(p)^2} - \frac{s^2 Hill'(p) Hill''(p)}{1 + Hill'(p)^2} \end{aligned}$$

Parameter	Value
m	3000
μ	0.8
δ_{sa}	0.1
δ_r	0.5
$\delta_{s'}$	0.1

Table 4.4: Parameters for sample transfer parameters used in the hill car experiments.

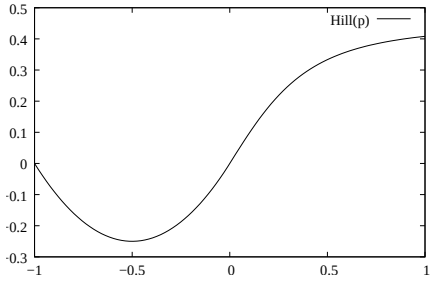


Figure 4.15: The profile of the hill.

where M is the mass of the car and g is the gravitational constant. The shape of the hill is (Figure 4.15):

$$Hill(p) = \begin{cases} p^2 + p & \text{if } p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & \text{if } p \geq 0 \end{cases} \tag{4.13}$$

The state variables are limited in the ranges $p \in [-1, 1]$ and $v \in [-3, 3]$. Whenever the car reaches the top of the hill ($p > 1$) the agent receives a positive reward of 1.0, while if the velocity $|v| > 3$ or the position $p < -1$ it receives a negative reward of -1.0 and the episode is ended. The default reward in any other situation is 0.

The parameters of the transfer algorithm are summarized in Table 4.4. The results are obtained by averaging 200 runs. In FQI, we use extra-randomized trees [41] with 50 trees, 2 random splits, and 2 minimum sample size for each node, trained on 30 iterations. Samples are collected through random sampling run on independent episodes of maximum 40 steps each. Each episode restarts with random position and velocity. Testing is performed on 50 episodes in which the car is placed at random with $p \in [-1, 0]$ and velocity $v = 0$.

4.6.2 Results: Region Transfer

In this section, we analyze the effectiveness of the region transfer compared to the random transfer. As described in Section 4.4.3, for each sample in the source task a measure of relevance is computed according to its compliance with the samples in the target task and its average distance from them. Whenever the distance is high, the relevance gets a

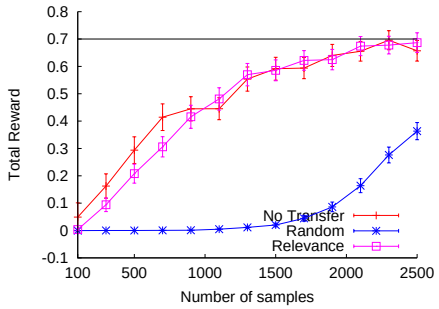


Figure 4.16: Total reward with or without transfer from S_1.

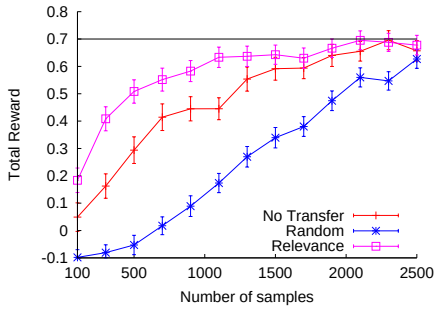


Figure 4.17: Total reward with or without transfer from S_1 and S_2.

default high value, under the assumption that in case of lack of evidence against the transfer of a sample, it is better to transfer it. On the other hand, when the distance is low, the relevance is directly related to the compliance of the sample.

The objective of the following experiment is to illustrate the effectiveness of the proposed transfer mechanism in identifying which samples are worth transferring in order to improve the learning performance. We consider a target task with "standard" dynamics and reward function and two source tasks in which the transition model undergoes a radical change in one region of the state-action space where the effect of the actions is inverted, that is for $a = -4$ the car accelerates, while for $a = 4$ the car decelerates. In particular, we consider a task S_1 in which the effect of the actions is inverted when the velocity is positive, while in task S_2 actions are inverted when velocity is negative. Thus, when agent selects an action a, the action executed in S_1 is

$$a_{S_1} = \begin{cases} -a & \text{if } a > 0 \\ a & \text{otherwise} \end{cases}, \tag{4.14}$$

while in S_2

$$a_{S_2} = \begin{cases} a & \text{if } a > 0 \\ -a & \text{otherwise} \end{cases}. \tag{4.15}$$

As a result, each task has roughly the same dynamics as the target task in half of the state-action space. As a result, while random sampling introduces samples that are significantly different from the correct dynamics (are exactly the opposite!), we expect the transfer mechanism in Algorithm 8 to identify the most relevant samples, thus avoiding negative transfer and improving the learning performance.

At first, we report the effect of relevance-based transfer (Algorithm 8) from one single source task, namely S_1. In this case, no significant improvement is expected with respect to learning without transfer. Samples from S_1 provide very limited information about the dynamics and the reward function. In fact, the optimal policy traverses many regions of the state space (i.e., both positive and negative velocity), thus the transfer of samples limited to one specific region is not likely to improve the performance of the learning algorithm in terms of learning speed. On the other hand, if samples from the region with different dynamics is transferred, negative transfer is expected.

Figure 4.16 compares the performance of the batch learning algorithm on the target task without transfer, with random transfer and with samples selected according to their relevance. As it can be noticed, the transfer of samples can lead to significant negative effects. In fact, half of the samples in the source are completely different from those of the target task and their transfer induces a policy that has a very poor performance. On the other hand, the transfer based on the relevance succeeds in avoiding negative samples, but the new samples are limited only to one region and are not informative enough to improve the learning speed of the learning algorithm. Considering the area ratio, we obtain that random transfer has a negative area ratio of $r_{random} = -198.46\% \pm 8.05\%$, while the relevance-based transfer achieves almost the same performance as learning without transfer ($r_{relevance} = -9.98\% \pm 10.26\%$).

In the second experiment, samples are transferred both from S_1 and S_2. In this case, we expect the transfer based on relevance to significantly improve the learning performance thanks to the transfer of samples from the regions of both the sources that are similar to the target task. On the other hand, the random transfer of samples is expected to obtain again a poor performance. In fact, it has a probability of 50% of transferring negative samples from any of the two source tasks to the target task. Figure 4.17 shows the average reward without transfer and with random and region transfer. As it can be noticed, region transfer greatly improves the performance of the learning algorithm. Unlike the experiments in the golf environment, in this case the improvement is not due only to a better initial performance. In fact, the region mechanism needs a sufficient number of samples before becoming effective. After few hundreds samples collected in the target task, the transfer algorithm succeeds in identifying which samples is more convenient to transfer and, as a consequence, the performance improves. In particular, while the random transfer has still a negative area ratio of $r = -115.12\% \pm 10.68\%$, the relevance-based

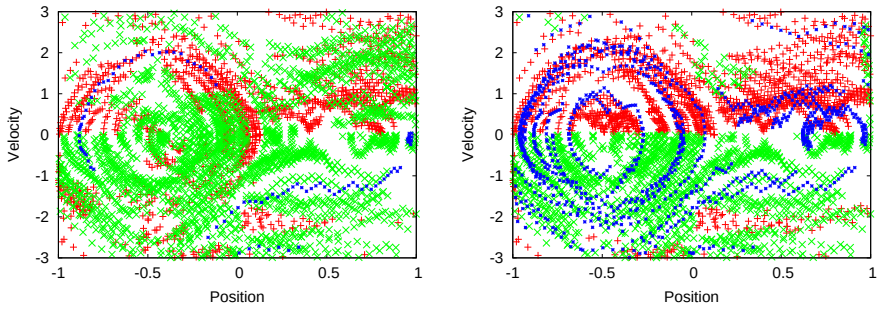


Figure 4.18: Transferred sample set $\widetilde{T}$ for $t = 100, 2000$. In *blue* the samples drawn from the target task T, in *red* the samples transferred from S_1 and in *green* the samples transferred from S_2.

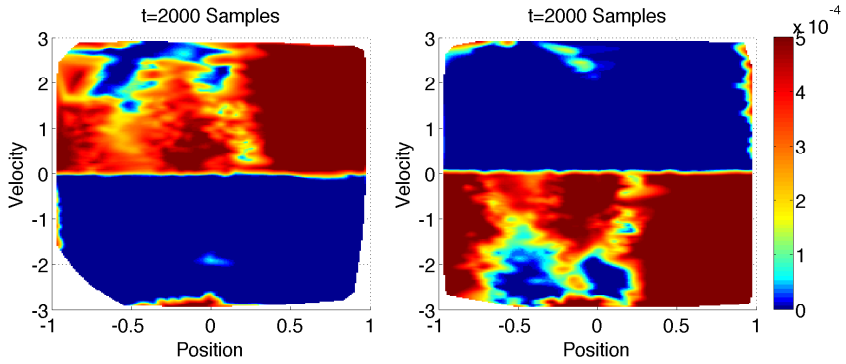


Figure 4.19: Relevance of samples in $\widehat{S}_1$ (*left*) and $\widehat{S}_2$ (*right*) for $t = 2000$.

transfer achieves an improvement of $r = 44.93\% \pm 10.07\%$. Finally, it is worth noting that the simple transfer of the policy of either S_1 or S_2 would obtain a very poor performance, since their optimal policies are significantly different from the optimal policy of the target task and they are not likely to improve the learning speed.

In order to get a better understanding of how the transfer algorithm works, in Figure 4.18, we report the transferred set of samples $\widetilde{T}$ for $t = 100, 2000$, while in Figure 4.19, we report the relevance for the two source tasks for $t = 2000$ (the plots are obtained by interpolating the relevance of the samples on the state space and averaging along the two actions). As described in Section 4.4.3, the relevance is based on the estimation of the target task model. Therefore, since in the initial stages of the learning

process, when few samples are available, the accuracy of the estimation is poor, the relevance of the samples of the source tasks depends on the distribution of the samples of the target task and a high value is given by default. As the learning progresses and more samples are available, the estimation of the model is more accurate and the relevance becomes more reliable. As it can be noticed in Figure 4.18-*left*, for $t = 100$, samples drawn from S_1 and S_2 are distributed over all the state space because the relevance is high in many regions by default. On the other hand, when 2000 target task samples are available, the relevance of the source tasks converges to a fixed configuration (Figure 4.19), in which only samples for positive velocity (source S_1) and negative velocity (source S_2) are transferred to the target task (Figure 4.18-*right*). As a result, the samples transferred from the source tasks are limited to their most relevant regions.

These experiments show that even when only a limited region of a source task is similar to the target task, the sample transfer algorithm can improve the learning speed of a batch RL algorithm, thus achieving nearly optimal performance even with a limited number of samples actually collected from the target task.

4.7 Experiments: the Boat Problem

4.7.1 Definition and Settings

The final experiment we consider is meant to sum-up the features of the proposed transfer method and to assess its effectiveness when source tasks are either chosen by hand or drawn from a continuous distribution of tasks. In particular, we consider the boat problem proposed in [72]. The problem is to learn a controller to drive a boat from the left bank to the right-bank quay of a river, in presence of a non-linear current. The boat's bow coordinates, x and y, are defined in the range $[0, 200]$ and the controller sets the desired direction $U \in [-90°, -45°, 0°, 45°, 90°]$. The action chosen by the agent is perturbed by a uniform noise in the range $[-5°; 5°]$. The control frequency is set to 1Hz and the dynamics of the boat's bow coordinates is described by the following equations:

$$\begin{aligned} x_{t+1} &= \min(200, \max(0, x_t + s_{t+1}\cos(\delta_{t+1}))) \\ y_{t+1} &= \min(200, \max(0, y_t - s_{t-1}\sin(\delta_{t+1}) - E(x_{t+1}))) \end{aligned}$$

where the effect of the current is defined by $E(x) = f_c\left(\frac{x}{50} - \left(\frac{x}{100}\right)^2\right)$, where f_c is the force of the current, and the boat angle δ_t and speed s_t

Parameter	Value
I/p	0.1 / 0.9
s_{MAX}/s_D	2.5 / 1.75
Z_s / Z_v width	10.0 / 10.0

Parameter	Value
m	2000
μ	0.8
δ_{sa}	0.1
δ_r	0.5
$\delta_{s'}$	0.1

Table 4.5: (*left*) Parameters of the dynamics of the boat. (*right*) Parameters for the sample transfer algorithm in the boat experiments.

are updated according to the desired direction U_{t+1} as:

$$\begin{aligned}
\delta_{t+1} &= \delta_t + I\Omega_{t+1} \\
\Omega_{t+1} &= \Omega_t + ((\omega_{t+1} - \Omega_t)(s_{t+1}/s_{MAX})) \\
s_{t+1} &= s_t + (s_D - s_t)I \\
\omega_{t+1} &= \min(\max(p(U_{t+1} - \delta_t), -45^\circ), 45^\circ)
\end{aligned}$$

where I is the system inertia, s_{MAX} is the maximum speed allowed for the boat, s_D is the speed goal, ω is the rudder angle, and p is a proportional coefficient used to computed the rudder angle in order to reach the desired direction U_t. Furthermore, we introduce regions of the river in which the speed is reduced by 20% because of sandbanks. The reward function is defined as:

$$\mathcal{R}(x,y) = \begin{cases} +10 & x = 200 \text{ and } y \in \mathcal{Z}_s \\ \text{D(x,y)} & x = 200 \text{ and } y \in \mathcal{Z}_v \\ \text{-10} & x = 200 \text{ and } y \in \mathcal{Z}_f \\ \text{-2} & (x,y) \in \text{sandbank} \\ \text{-2} & y \leq 0 \text{ or } y \geq 200 \\ 0 & \text{elsewhere} \end{cases} \tag{4.16}$$

where D is a function that gives a reward decreasing linearly from 10 to -10 relative to the distance from the success zone, $\mathcal{Z}_s$ is the quay, $\mathcal{Z}_v$ is the viability zone around the quay, and $\mathcal{Z}_f$ is the failure zone in all the other bank points.

The dynamics and learning parameters are summarized in Table 4.5. In the following experiments, we use Gaussian kernels and Mahalanobis distance (see Section 4.4.2). The results are obtained by averaging 100 runs. In FQI, we use extra-randomized trees [41] with 50 trees, 2 random splits, and 2 minimum sample size for each node, trained on 25 iterations. Samples are obtained through random sampling run on independent episodes of maximum 50 steps each. Each episode restarts

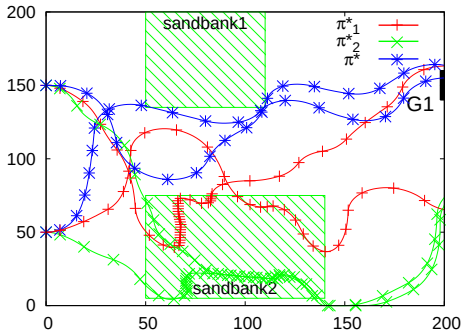


Figure 4.20: Position of sandbanks and of the goal in the target task and trajectories of the optimal policies of S_1, S_2, and T tested in T.

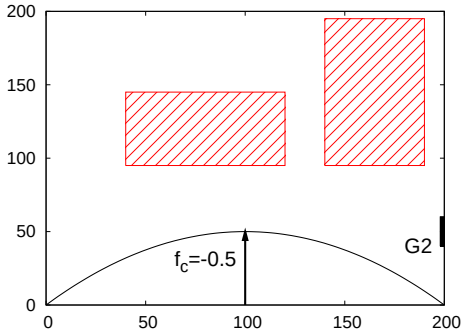


Figure 4.21: Position of sandbanks and of the goal in S_2.

the boat at the left bank in a random position. Testing is performed on 1,000 episodes with the initial position drawn at random from 20 evenly spaced positions at the left bank.

4.7.2 Results: Sample Transfer

The first experiment is meant to illustrate the effectiveness of compliance and relevance in identifying which samples are worth transferring. We consider a transfer problem with three tasks in which S_1 and S_2 are the source tasks and T is the target task. In T the quay is G_1 and there are two sandbanks as illustrated in Figure 4.20. In task S_1 there are two quays G_1 and G_2, and there is only one sandbank corresponding to the region labeled as *sandbank1* in Figure 4.20. Task S_2 has the quay G_2 and

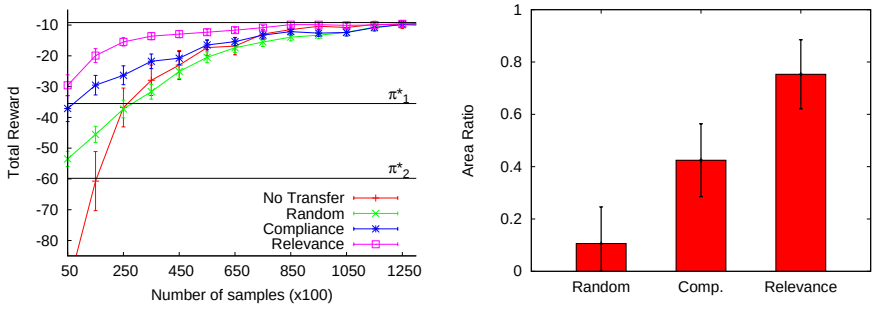


Figure 4.22: Total reward and area ratio for all the configurations.

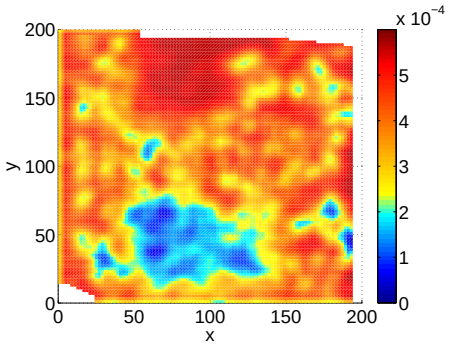


Figure 4.23: Relevance of samples in $\widehat{S}_1$ at convergence.

the sandbanks illustrated in Figure 4.21. While T and S_1 have the same current force ($f_c = 0.5$), the current in S_2 is in the opposite direction ($f_c = -0.5$). The source task S_2 has a completely different dynamics and reward function from those in T because of different sandbanks and current. Therefore, samples transferred from S_2 are likely to induce negative effects on the learning performance of T. Furthermore, as it can be noticed from the trajectories shown in Figure 4.20, the optimal policy π^*_2 of S_2 obtains very poor performance when tested on T. On the other hand, S_1 has the same dynamics as T in large regions of the state-action space and shares one goal with T. Although its optimal policy π^*_1 is still significantly different from π^*, it is possible to choose samples drawn from S_1 that can actually improve the performance of the learning algorithm in T.

In Figure 4.22-(*left*) we report the performance obtained by FQI with

four different configurations: *No Transfer*, *Random*, *Compliance*, and *Relevance Transfer*. The first configuration is FQI with an increasing number of samples directly collected from T. The other three configurations are run on the sample set $\widetilde{T}$ obtained by transferring samples chosen at random, according to the compliance, and according to the relevance (Algorithm 8) respectively. Furthermore, we also report the performance obtained by transferring policies π_1^* and π_2^* as baselines. The augmentation of $\widehat{T}$ with samples drawn from S_1 and S_2 at random does not lead to any significant improvement of the performance with respect to learning directly on samples in $\widehat{T}$. In fact, the only advantage achieved with the transferred samples is that the agent avoids to go outside of the boundaries, but she learns neither to avoid sandbanks nor to achieve the goal. The main reason for this poor performance is that samples drawn from S_2 do not provide any information about the actual dynamics and rewards of T and, thus, may lead to learning very bad policies. On the other hand, the compliance-based transfer successfully excludes samples of S_2 from the transfer process (the normalized compliance of S_1 for $t = 200$ is $\overline{\Lambda}_1 = 0.93 \pm 0.09$), thus augmenting $\widehat{T}$ with samples mainly coming from S_1. Since S_1 shares with T the dynamics and the rewards in all the state space but at *sandbank2* and in the quay G_2, the transfer is positive and leads to a significant improvement in the performance of the learning process. Nonetheless, there are still many trajectories leading to the quay G_2 and crossing the sandbank because of the negative effect of transferring samples from regions with dynamics and reward different from T. In Figure 4.23 we report the relevance of the samples in $\widehat{S}_1$ (averaged on all the actions). As it can noticed, the relevance succeeds in identifying regions where samples are actually similar in source and target tasks, excluding samples coming from the region *sandbank2* and the lower quay G_2. [3] As a result, the performance of the transfer based on the relevance is further improved.

In order to evaluate the relative improvement of transfer, we compute the area ratio [144] of the three transfer configurations, defined as the difference between the accumulated reward with and without transfer divided by the reward accumulated without transfer. Figure 4.22-(*right*) shows the area ratio for the three transfer configurations. As it can be noticed, the random transfer of samples does not lead to any significant improvement, while transfer based on relevance increases the cumulative

[3] It is worth noting that sandbanks have a slightly higher relevance than other regions because the boat moves slower and, thus, samples are more concentrated in those regions.

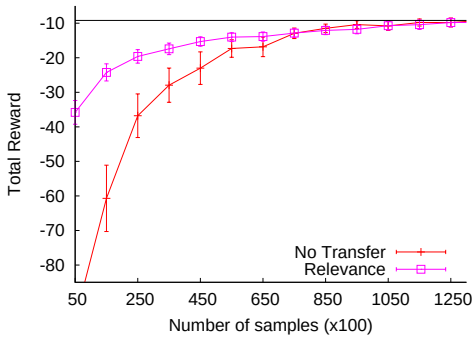


Figure 4.24: Performance of sample transfer with five random source tasks.

reward of about $75.3\% \pm 13.2$. The difference between the configurations is statistically significant ($p < 0.01$).

In the previous experiment, source and target tasks have been chosen by hand with the aim of illustrating how the algorithm works. Now, we consider the general case in which tasks are drawn from an infinite task space $\mathcal{T}$. For sake of simplicity, we consider the same target task of the previous experiment, while source tasks have current $f_c = 0.5$ and one sandbank. Source tasks are drawn from a distribution Ω such that the coordinates of the center, height, and width of the sandbank are uniformly drawn from the space $[20.0; 180.0] \times [20.0; 180.0] \times [40.0; 100.0] \times [40.0; 100.0]$, while the quay position is drawn uniformly from $[20.0; 180.0]$. In Figure 4.24, we report the results of relevance-based transfer obtained by averaging the result with 10 different sets of five source tasks. Although the source tasks are different from the target in large regions, the transfer algorithm is able to identify which samples are worth transferring from the source tasks and it successfully improves the performance of the learning, with an area ratio of $59.5\% \pm 15.4$.

4.8 Related Works

Since the algorithms of transfer in RL proposed so far rely on temporal-difference or model-based algorithms, an empirical comparison with the performance of sample transfer would not be fair. In this section, we discuss its similarities and differences with other transfer approaches.

The idea of transferring either samples or solutions from different sources in order to improve learning speed is not new in transfer learn-

ing literature [115]. At the same time, when multiple sources are available the problem of transfer comes with the problem of avoiding negative transfer and measuring task relatedness. In [22], Ben-David and Schuller introduce a data generating framework, for which a task relatedness measure is proposed. Given the task relatedness they derive strong generalization bounds proving the advantage of transfer learning over single task learning. In [79, 107], the problem of estimating a Bayesian network exploiting a set of samples generated from other networks is considered. It is shown that by approximating the models of the source networks, it is possible to achieve positive transfer by using the approximated models to estimate the parameters of the model of the target network. A similar idea is used in the context of MDPs in [129], where a Bayesian perspective is followed, in which the source task models are pre-posteriors distributions for the parameters distributions of the target model and a model based RL algorithm is used to compute the solution. Although we similarly adopt a Bayesian argument for the computation of the compliance of samples, we directly transfer samples and a model-free algorithm is used to learn the solution on the transferred sample set. Furthermore, instead of a parametric approximation of the model of the source tasks, we followed a non-parametric solution, that is more strictly related to the samples available from the tasks at hand. Finally, our transfer technique is related to [97], where the problem of model identification is stated as a POMDP in which the task at hand is considered as a partially observable variable. The proposed algorithm updates a belief state about the task that is used in the update formula of SMDP Q-learning [136] performed over a set of options transferred across tasks.

More theoretically founded metrics for MDP similarity are introduced in [47] in the context of bisimulation. They propose two distance metrics, defined on the state space, used to identify states with equivalent outcomes (i.e., dynamics and reward) that can be aggregated into equivalence classes, thus reducing the dimensionality of the MDP. Under a transfer perspective, these metrics can be used to measure the difference between states in two distinct tasks and to bound the performance loss of using the optimal policy of a source task in the target task. Unfortunately, this technique cannot be directly applied to our scenario for different reasons. The computation of the distance between different states is very expensive, because it requires the solution of a complex optimization problem. Furthermore, the proposed algorithm needs either the exact models of tasks or an accurate approximation. On the other hand, the proposed method relies on a lightweight solution with

low complexity depending only on the number of samples of the source tasks actually available. Finally, empirical analysis [98] showed that the theoretical bounds on the performance loss are often too loose in practical situations and thus they rarely provide useful directions about the actual performance of the transferred policy.

The transfer of samples is strictly related to works about transfer of policies in the RL context (see Section 3.3.3 for the references). In the experimental section, we discussed that transferring samples from one source task is almost equivalent to the transfer of the optimal policy of the source task to the target task. Nonetheless, there are situations in which the transfer of samples can obtain significantly different results with respect to the transfer of policies. Let us consider the case in which the difference between the source and the target task is limited to a change in the transition model in few state-action pairs, while the rest of the models is identical. In this case, the two optimal policies can be significantly different and when the optimal source policy is applied to the target task, it may achieve very poor performance. On the other hand, the transfer of samples, in particular when performed using the region transfer criterion, can still be effective. In fact, since most of the samples in the two tasks are identical, the learning algorithm can benefit from samples coming from the source task independently from the actual difference of the two optimal policies. Furthermore, most of the works based on policy transfer are either limited to transfer the same set of policies in all the tasks (option transfer) or only to one source task (optimal policy transfer). In fact, it is still not clearly defined how to transfer policies from different source tasks, in particular if we want to transfer only parts of the policies as in the region sample transfer. Furthermore, the transfer of samples does not require to actually solve the source tasks, and it can be used even when the samples are not enough to solve source tasks. A solution more sophisticated than simple policy transfer is proposed in [82], in which the model-based hierarchical task decomposition allows for transfer at multiple levels of the hierarchy. Nonetheless, this approach relies on the assumption that rewards are a linear combination of *basis* reward functions and it can be applied only to problems of goal transfer, with a fixed transition model. On the other hand, sample transfer can be applied to any transfer scenario.

Finally, a recent work [140] proposes a techniques for the transfer of instance in model-based RL. Although similar in principle, this work deals with transfer problems with only one source task and focuses mainly on the issues arising from tasks with different state-action spaces.

4.9 Conclusions

In this chapter, we introduced a transfer mechanism for the improvement of learning speed on the basis of the most simple form of experience available to the agent, the trajectory samples. At the best of our knowledge this is the first attempt to use samples for transfer in batch RL. The main advantages of the proposed solution are: *(i)* the transfer of samples is completely independent from the similarity of the policies and action-value functions of the tasks at hand, *(ii)* the kernel-based computation of the approximated models has low complexity, *(iii)* it can be applied in conjunction with any batch RL algorithm, *(iv)* it can be applied to the general scenario of domain transfer, *(v)* there is no need for explicitly solving the source tasks. Experimental results show the effectiveness of the method in reducing the learning time and in avoiding negative transfer when the source tasks are significantly dissimilar from the target task. Furthermore, it is worth noting that sample transfer can be iteratively used as new tasks are extracted from Ω.

Future directions of investigation are:

1. *Transfer of trajectories.* In many problems, the performance of batch RL algorithms is affected from the way samples are collected from the environment. The transfer introduced here does not take into account the way samples are obtained. Nonetheless, in case of samples collected in few trajectories, it could be useful to extended the current approach to the transfer of segments of trajectories by extending the definitions of compliance.

2. *Separated transfer of model and reward.* In transfer problems, it may happen that some tasks either share exactly the same transition model or the same reward function (e.g., the golf problem discussed in Section 4.5). In this case, it is possible to transfer only the part of the samples in common between the source and the target task. For instance, if two tasks share the same transition model but has different goals, it is possible to transfer the $\langle s, a, s' \rangle$ part of the samples and to "complete" the sample using an approximation of the reward function of the target task (e.g., using the first iteration of fitted Q-iteration).

3. *Approximated models and compliance definitions.* More sophisticated algorithms of model approximation [129, 36] could be compared to the kernel-based solution adopted here. Furthermore, the

definition of compliance could be compared to the similarity metrics introduced in [47].

4. *Theoretical analysis.* The main drawback of the proposed solution of sample transfer is the lack of a theoretical relationship between the compliance of tasks and the actual speed-up that can be obtained from their transfer. Unfortunately, the problem of defining the "utility" of a sample in batch RL algorithm, that is, the advantage in terms of accuracy of the optimal action-value function approximation, is still an open question [40]. A possible research direction on this topic should take into account recent results on sample complexity of batch RL algorithm proved in [5] and performance bounds of policy transfer derived in [47].

5 Multi-Task Batch Reinforcement Learning

In this chapter, we investigate the possibility to adopt a multi-task perspective in the RL paradigm. In particular, we analyze how a batch RL algorithm, notably fitted Q-iteration (Section 2.7.3), can be integrated with a multi-task optimization algorithm in order to achieve a generalization improvement.

5.1 Introduction

As discussed in Section 3.2.2, multi-task learning is concerned with exploiting data collected from different tasks to improve the generalization performance in all of them. The main difference with respect to inductive transfer is that here the focus is on the tasks at hand and not on new tasks that will be eventually solved. Multi-task algorithms have as primary goal the reduction of the test error of a learning algorithm.

While many research activities in supervised learning focused on the multi-task perspective, in RL almost all the literature about learning on multiple tasks focuses on the perspective of inductive transfer, in which the goal is the improvement of learning speed. Nonetheless, it is widely recognized that one of the most relevant problems in RL algorithms is to design function approximators able to compute an accurate approximation of the optimal (action) value function. In this chapter, we analyze how it is possible to integrate multi-task learning algorithms into batch RL algorithms in order to change the structure of a function approximator and improve the generalization performance over a finite set of tasks.

5.2 Background

In traditional supervised problems the objective is to learn a function that accurately approximates a training set of data and that generalizes well in previously unseen inputs. Given a set of n tasks, the objective

of a multi-task learning algorithm is to learn a set of n functions so as to improve the generalization performance with respect to learning the functions for each task independently. The implicit assumption under multi-task learning is that all the tasks are somehow related and that learning algorithms can benefit from sharing the information stored in the training sets of each task. This relatedness can be of two types: *(i)* all the functions are close and, thus, learning one function can be a useful bias for learning the others [14, 42], *(ii)* there exists an underlying feature space shared across all the tasks, and, thus, learning the feature space can simplify the learning for all the functions [3, 9, 34, 106, 148]. In this chapter, we focus on the second perspective.

A large part in the success of a learning algorithm is determined by the representation of the tasks at hand and by the structure of the approximator. This evidence is at the basis of the multi-task learning approach on neural networks proposed in [34], where the functions are learned using one single neural network with n outputs, so that all the functions share at least one layer of the neural network, thus implicitly forcing a common representation shared across tasks. A more theoretically founded approach to multi-task learning is introduced in [18]. The main contribution of that work is in the definition of a model of multi-task learning (*inductive bias learning* in the paper) and a formal definition of the objective of a multi-task learning algorithm, that is the bias of the feature (hypothesis) space so as to minimize the empirical approximation error over all the tasks. Furthermore, bounds that demonstrate the effectiveness of the multi-task approach with respect to single-task learning are proved. A structural learning algorithm is proposed in [3] in the context of semi-supervised learning. The main intuition is that the feature space can be parameterized and the learning problem becomes a joint optimization problem on learning and feature parameters. The main problem is that the optimization problem is non-convex and only approximated solutions can be learned. Finally, in [9] a similar approach is introduced along with the definition of an equivalent optimization problem that is convex and that can be solved by iteratively minimizing on learning and feature parameters.

In RL, many works focused on the adaptation of function approximators in order to improve the performance of learning algorithms in single-task problems [161, 122, 84]. Recent works tried to define principled ways to extract basis functions (i.e., feature space) for batch RL algorithms (namely, LSPI) following different approaches, such as spectral structural learning [78, 45], dimension reduction [67], basis function

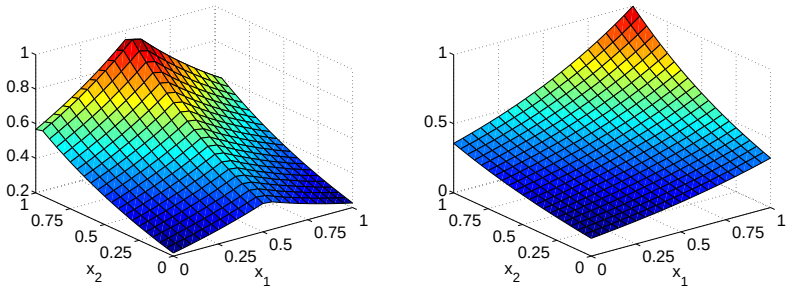


Figure 5.1: Optimal value functions of two tasks defined in a continuous maze environment.

expansion [95], discovery of irrelevant variables [59]. Nonetheless, none of these works addresses multi-task problems. On the other hand, a number of works highlighted the relevance of the representation in the performance of RL algorithms applied to multi-task problems. In [39] a mechanism for the identification and transfer of relevant features is proposed. Unsupervised, mixture model, learning methods are adopted in [48] to analyze the optimal value functions and extract fragments of value functions that are common to all the tasks. In [11] the subgoal decomposition of a MAXQ algorithm is shared across the tasks along with a state abstraction obtained through model-minimization techniques. Finally, [164] explicitly addresses the problem of multi-task in RL by following a hierarchical Bayesian approach.

5.3 Motivating Example

Before entering in details about the integration of multi-task learning in batch RL algorithms, we discuss a very simple example with the aim of showing how the representation affects the actual possibility to achieve effective transfer between different tasks.

We consider a simple continuous maze environment[1] in which the tasks share the same dynamics but have different goals. The reward function is zero everywhere but in the goal state where a positive reward is returned. Besides the position on the maze (x_1, x_2), the agent perceives two irrelevant variables (x_3, x_4) that take random values at each step. Furthermore, we introduce a bias variable x_0 fixed to a constant value

[1] Experiments in a more complex maze environment are reported in Section 5.5.

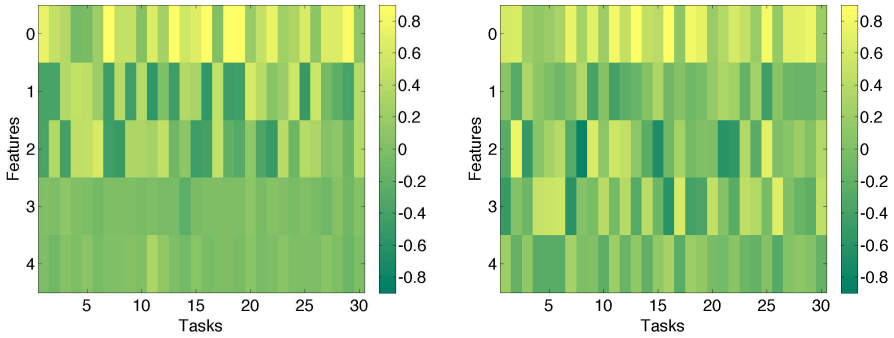


Figure 5.2: Learning parameters learned on 30 tasks independently drawn from Ω for two linear function approximators f and $\widetilde{f}$.

of 1, thus obtaining the state vector $x \in \mathbb{R}^5$. The task distribution Ω is a uniform distribution that gives high probability to goals near the borders of the maze. An example of the optimal value functions of two tasks is reported in Figure 5.1. It is worth noting that all the functions are in general very different (e.g., in 2-norm), but they are very similar in terms of the underlying representation (i.e., state variables x_1 and x_2). In order to approximate the optimal value function of t-th task task, we use two linear function approximators

$$f_t(x) = \sum_{i=0}^{4} \vartheta_{ti} x_i, \qquad \widetilde{f}_t(x) = \sum_{i=0}^{4} \widetilde{\vartheta}_{ti} \phi_i(x), \tag{5.1}$$

where ϑ_{ti} is the i-th learning parameters. While the former is defined directly on the state space x, the latter is defined in a feature space ϕ such that $\phi_0(x) = x_0$, $\phi_1(x) = x_1 + x_3$, $\phi_2(x) = x_2 x_3$, $\phi_3(x) = x_1 + x_2 x_4$, $\phi_4(x) = x_1 x_3 + x_4$. Using these approximators we implicitly force all the tasks to share the same underlying representation.

We draw 30 tasks with the same transition model but with different reward functions defined according to Ω and few samples for each task, then we approximate their optimal value functions using fitted Q-iteration. In Figure 5.2 we report the learning parameters of both the approximators for each task. On columns the parameters vectors ϑ_t,$\widetilde{\vartheta}_t$ for the t-th task are reported, while on rows there are the parameters corresponding to each feature across the tasks. Since only x_1 and x_2 are actually useful for the approximation of the value function, in f the only relevant parameters are ϑ_0, ϑ_1 and ϑ_2, while ϑ_3 and ϑ_4 are always

negligible (almost zero). On the other hand, in $\widetilde{f}$ the relevance of a feature ϕ_i depends on the specific task. Therefore, while the feature space ϕ cannot be generalized across tasks, the feature space of f is shared by all the tasks. As a result, we expect a multi-task learning algorithm to work better with the representation in f than with the feature space of $\widetilde{f}$. In fact, even when few samples are available for each task, if the chosen representation is actually shared across all the tasks, a learning algorithm can greatly benefit from learning on all the tasks at the same time. Conversely, the feature space of $\widetilde{f}$ does not allow any effective transfer of information across the tasks.

From this simple example, it appears clear that a critical aspect of multi-task learning is the representation used to define the multi-task problem. Therefore, in defining a multi-task perspective to transfer in RL, it is important not only to define a joint learning problem on all the tasks but also to adapt their representation in order to achieve effective transfer across tasks. In the following, we discuss the integration of fitted Q-iteration with a recent multi-task algorithm [9] able to learn both the learning parameters and the feature space at the same time.

5.4 Multi-Task Batch Reinforcement Learning

In this section we introduce a formulation of the multi-task problem in batch RL and integrate one multi-task learning algorithm in fitted Q-iteration (FQI).

5.4.1 Problem Formulation

In the following, we consider tasks in which the use of function approximation is mandatory (e.g., tasks with continuous state and action spaces). The structure of the appoximator (e.g., basis functions in a linear function approximator, layers and nodes in a neural network) determines the space $\mathcal{Q}$ of the functions that can be represented, while the learning algorithm computes the function $\widehat{Q} \in \mathcal{Q}$ that better approximates the optimal value function Q^* of the task at hand. It is clear that the choice of $\mathcal{Q}$ is critical in order to have a good approximation of Q^*, and, thus, a policy with a nearly-optimal performance. When multiple tasks must be solved, the problem becomes the problem of choosing a suitable $\mathcal{Q}$ in which a good approximation for the optimal value function of each task is available.

Thus, in this chapter, we focus on the multi-task learning perspective as the most suitable approach to achieve the improvement of the learning

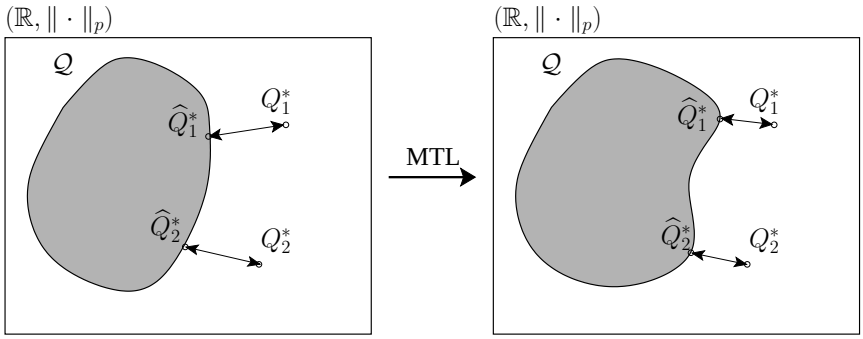


Figure 5.3: Qualitative representation of the desired effect of a multi-task learning (MTL) algorithm. The space of action-value functions $\mathcal{Q}$ is biased so as to reduce the approximation error (computed in a given norm $\|\cdot\|_p$).

performance. Unlike the scenario in Chapter 4, we do not distinguish between source and target tasks, but we have a finite set of n tasks $\{T_t\}_{t\in\mathbb{N}_n}$ extracted from Ω. We can qualitatively state the problem of defining an algorithm that biases the space of action-value functions $\mathcal{Q}$ as

$$A_{Transfer} : \widehat{\mathcal{T}}^n \rightarrow \mathbb{Q}. \tag{5.2}$$

Given the samples available for each task, the objective is to identify an action-value function space $\mathcal{Q} \in \mathbb{Q}$ that leads to small approximation errors of each optimal action-value function (Figure 5.3).

As discussed in Section 3.4, the advantage of batch RL algorithms is the possibility to distinguish between the sampling algorithm, that mainly affects the learning time, and the learning algorithm, that directly affects the quality of the approximation. Although the multi-task perspective could be implemented in different batch RL algorithms, fitted Q-iteration (FQI) is the most suitable for the integration with multi-task algorithms. In fact, since at each iteration of FQI a regression problem is solved, it is possible to state the problem of regression on multiple tasks as a multi-task learning problem and different solutions defined in supervised literature can be easily adapted to the FQI process.

The definition of the fitted Q-iteration algorithm under the multi-task perspective is summarized in Algorithm 9. At each iteration k, we build a dataset for each task, with input samples $x_{ti}^k = (s_{ti}, a_{ti})$ and output samples $y_{ti}^k = r_{ti} + \max_{a'} Q^{k-1}(s'_{ti}, a')$. The set of input (output) samples for task T_t is denoted by x_t^k (y_t^k), while the set of all the input

Algorithm 9 The Multi-Task Fitted Q-iteration algorithm

Input: sets of samples $\{\widehat{T}_t\}_{t\in\mathbb{N}_n}$
Parameters: γ, N, *multi-task-regressor*
Output: $\{\widehat{Q}_t^N\}_{t\in\mathbb{N}_n}$

Initialize $\{\widehat{Q}_0^t\}_{t\in\mathbb{N}_n}$
for $k = 1$ to N **do**
  for $t = 1$ to n **do**
    Generate input set x_t^k, with $x_{ti}^k = (s_{ti}, a_{ti})$, $i = 1 \dots |\widehat{T}_t|$
    Generate output set y_t^k, with $y_{ti}^k = r_{ti} + \max_{a'} \widehat{Q}_t^k(s'_{ti}, a')$, $i = 1 \dots |\widehat{T}_t|$
  end for
  Let $\mathbf{x}^k$ be the vector of input sets x_t^k and $\mathbf{y}^k$ be the vector of output sets y_t^k
  $\{\widehat{Q}_t^{k+1}\} \leftarrow$ multi-task-regressor$(\mathbf{x}^k, \mathbf{y}^k)$, $t \in \mathbb{N}_n$
end for

samples is $\mathbf{x}^k$ ($\mathbf{y}^k$). Given the joint dataset $(\mathbf{x}^k, \mathbf{y}^k)$, we solve a multi-task problem and compute functions Q_t^k. The basic assumption underlying this algorithm is that the action-value functions approximated at each iteration k share a common underlying representation and, thus, the accuracy of the approximation can be improved by solving a multi-task regression problem. Furthermore, it can be easily proved [41] that the more accurate the approximation at each iteration k, the more accurate the final approximation of the optimal action-value function and, as a result, of the optimal policy.

In the following, we focus on the case of n linear function approximators that share the same basis functions. Let $\widehat{Q}_t^k(x)$ be the action-value function used to approximate the dataset (x_t^k, y_t^k)

$$\widehat{Q}_t^k(x) = \sum_{i=1}^{M} \vartheta_{ti}\phi_i(x) = \langle \vartheta_t, \phi(x) \rangle, \tag{5.3}$$

where M is the dimensionality of the feature space, $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^M$ is the vector of basis functions, and $\vartheta_t \in \mathbb{R}^M$ is the vector of learning parameters for task t. While the basis functions determine the space of functions $\mathcal{Q}$, an assignment of parameters vectors ϑ_t determines one specific function $\widehat{Q}_t^k \in \mathcal{Q}$. As it can be noticed, at each iteration k, we force all the tasks to share exactly the same function space, while the parameters can be optimized for the specific task at hand. Therefore, the multi-task problem is how to learn the parameters ϑ_t and to change the feature space ϕ in order to achieve the best approximation of the

action-value function Q_t^k.

The first approach that can be adopted is to design the feature space ϕ in advance and to define a joint optimization problem on the learning parameters ϑ_t. Therefore, unlike running n independent regression problems, we define a global loss function that combines the losses on each task and a learning algorithm that learns the set of parameters vectors that minimizes this loss function. This perspective is implicitly adopted in some algorithms of multi-task learning [34, 37]. This algorithm forces all the action-value functions to have a common representation, but it does not directly change the action-value function space $\mathcal{Q}$. The main drawback is that it is often difficult to identify the most suitable space by hand and, although the learning algorithm can benefit from samples of different tasks, the final performance can be negatively affected by a wrong choice of $\mathcal{Q}$.

Another approach is to define a feature extraction algorithm that defines a set of basis functions according to the information available for each task. After the feature extraction, the learning parameters are learned by solving n independent regression problems. Examples of this approach in SL literature are structure learning algorithms, whose goal is to identify the structure shared across the tasks [7, 88]. An example of this approach in the RL paradigm is the proto-value function framework [78]. Although it does not explicitly address a multi-task problem, proto value functions can be seen as a multi-task feature extraction algorithm based on spectral analysis of the MDP graph, in which all the tasks share the same transition model. While this approach actually change the space $\mathcal{Q}$, the learning algorithm independently optimizes the parameters ϑ_t for each task and does not benefit from the possible similarities among the value functions to be learned.

Finally, some recent works [9, 8, 3, 18] focused on the definition of a joint learning problem where both the parameters and the feature space are optimized by minimizing a global loss function. This perspective is the most complete multi-task approach, since all the elements involved in the approximation problem (i.e., learning parameters and feature space) are optimized according to the tasks at hand. The main difficulty in solving such problems is in the computation of the solution of the optimization problem. In fact, in general they are non-linear non-convex problems whose solution has a high complexity. In the following, we will focus on Multi-Task Feature Learning (MTFL) [9, 8], a recent algorithm in which the optimization problem is defined in a kernel space for which there exists an equivalent convex optimization problem that can

be exactly solved through an iterative algorithm.

5.4.2 Fitted Q-iteration with Kernel Regression

Since MTFL defines the multi-task problem in kernel spaces, before entering in details about its definition when applied to a RL problem, we briefly introduce a version of fitted Q-iteration with kernel regression (or kernel ridge regression) [108]. We consider a generic iteration of Algorithm 7 and we focus on the action-value function of one task. Let $(x, y) = \{(x_i, y_i)\}_{i \in \mathbb{N}_m}$ be the dataset used to train a linear function approximator

$$\widehat{Q}(x) = \langle \vartheta, \varphi(x) \rangle, \tag{5.4}$$

where $\widehat{Q} : \mathbb{R}^d \rightarrow \mathbb{R}$ is defined in a *reproducing kernel Hilbert space* (RKHS) and $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^M$ is the feature space for which the kernel function $K(x, x') = \langle \varphi(x), \varphi(x') \rangle$ can be computed. The kernel regression problem, is usually stated as the problem of finding the function $\widehat{Q}$, i.e., parameters ϑ, that minimizes the following regularized error function (Tikhonov minimization problem)

$$\mathcal{E}(\vartheta) = \sum_{i=1}^{m} L\left(y_i, \langle \vartheta, \varphi(x_i) \rangle\right) + \lambda \|\vartheta\|_2, \tag{5.5}$$

where $L(\cdot, \cdot)$ is a loss function, λ is a regularization parameter, and $\|\cdot\|_2$ is the 2-norm. The regularization parameter λ determines the sparsity of the solution, that is the number of zero elements in the optimal vector of parameters ϑ. The intuition underlying regularized approximators is that many dimensions of the feature map φ are often useless and forcing the solution to be sparse increases the generalization capability of the approximator. The choice of λ is critical for the actual performance of the regressor and it is usually determined through leave-one-out validation. An advantage of kernel regression with respect to other regularized algorithms is that there exist formulas to compute the leave-one-out mean-squared error using the results of the training, without actually performing the leave-one-out. In the following, we consider a square loss function $L(y, f(x)) = (y - f(x))^2$, thus reducing to the traditional regularized least squares problem. Let G be the Gram matrix, whose generic element G_{ij} is $K(x_i, x_j)$ with $i, j \in \mathbb{N}_m$. Following the result of the representer theorem [85, 109], the optimal vector of parameters can be computed in the space of the training data x as

$$\vartheta = \sum_{i=1}^{m} c_i \varphi(x_i). \tag{5.6}$$

Therefore, the problem of computing the vector ϑ minimizing the approximation error, becomes the problem of computing the vector of coefficients $c \in \mathbb{R}^m$. By minimizing the error function on c, we obtain

$$c = (G + \lambda I_m)^{-1} y, \tag{5.7}$$

where $I_m \in \mathbb{R}^{m \times m}$ is the identity matrix. Finally, the approximated action-value function becomes

$$\widehat{Q}(x) = \sum_{i=1}^{M} \vartheta_i \varphi_i(x) = \sum_{i=1}^{m} c_i K(x_i, x). \tag{5.8}$$

As it can be noticed, in the computation of the coefficients c and in the evaluation of the approximated action-value function in a state-action x, there is no need to compute the value of the basis function $\varphi(x)$ but only the value of $K(x_i, x)$ must be computed. As a result, it is possible to use high-dimensional kernel functions $\varphi(x)$ (in the limit, M could also be infinite) and, at the same time, to have a very efficient regressor. In fact, the kernel regression algorithm can be easily optimized in the case of fitted Q-iteration. Since the input samples are always the same in each iteration k, the inversion of the matrix $(G + \lambda I)$ can be performed only once and the computation of the optimal coefficients c is obtained from a simple matrix multiplication. The main drawback of kernel regression is that it becomes more and more inefficient as the number of samples increases. In fact, the minimization problem has a complexity of $O(m^3)$, that is the complexity of the inversion of the matrix $(G+\lambda I)$. Furthermore, all the samples must be kept also after the optimization in order to evaluate the learned function $\widehat{Q}(x)$. A possible optimization is to use the eigendecomposition of the Gram matrix G, that is a semi-definite positive matrix, thus reducing the complexity of the inversion operation. Furthermore, several mechanisms for the selection of subsets of samples are available.

The performance of kernel regression is strictly related to the choice of the function $\varphi(x)$ or, more precisely, to the choice of the kernel function $K(x, x')$. Examples of kernels are [110]:

- *polynomial*: $K(x, x') = (\langle x, x' \rangle + c)^d$
- *sigmoid*: $K(x, x') = \tanh(c\langle x, x' \rangle + d)$
- *Gaussian*: $K(x, x') = \exp\left(\frac{-\|x-x'\|^2}{2\sigma^2}\right)$

The Gaussian kernel is one of the most common kernel functions because its corresponding feature space has an infinite dimensionality (the Gaussian kernel is obtained from the polynomial kernel when the degree tends to infinity) and the bandwidth σ directly affects the generalization of the approximator.

5.4.3 Multi-Task Feature Learning

In this section, we review the Multi-Task Feature Learning algorithm [9] and we discuss its application to the approximation of the action-value functions Q_t^k generated at each iteration k of fitted Q-iteration. In the following, we omit the index k in the formulas and we describe the approach for generic action-value functions Q_t. The objective of multi-task feature learning is to adapt the feature space shared across a set of n tasks, together with the optimization of the learning parameters for each task. In MTFL we consider a feature space ϕ defined as a linear combination of a kernel function φ

$$\phi(\cdot) = U^\mathsf{T}\varphi(\cdot),$$

where U is an orthonormal matrix. The resulting linear function approximator is

$$\widehat{Q}_t(x) = \langle \vartheta_t, U^\mathsf{T}\varphi(x)\rangle. \tag{5.9}$$

The multi-task problem is then defined both on the learning parameters ϑ_t of each task and on the transformation matrix $U \in \mathbb{R}^{M\times M}$ that modifies the feature space ϕ.

For the sake of simplicity we first consider the case of linear kernel $\varphi(x) = x$, thus reducing the function approximator to

$$\widehat{Q}_t(x) = \sum_{i=1}^{d} \vartheta_{ti}\langle u_i, x\rangle = \langle \vartheta_t, U^\mathsf{T}x\rangle, \tag{5.10}$$

where $x \in \mathbb{R}^d$ and $u_i \in \mathbb{R}^d$ is the i-th column vector of the matrix $U \in \mathbb{R}^{d\times d}$. Notice that in this case we set the number of features M equal to the dimensions of the task d, that is the number of state variables plus the number of action variables.

The MTFL definition of the multi-task problem is based on two main assumptions: *(i)* the solution of each task is a sparse vector of learning parameters ϑ_t, *(ii)* the features whose corresponding learning parameter is non-zero are common across all the tasks. While the first assumption is common to all the regularized algorithms (e.g., kernel regression, SVM),

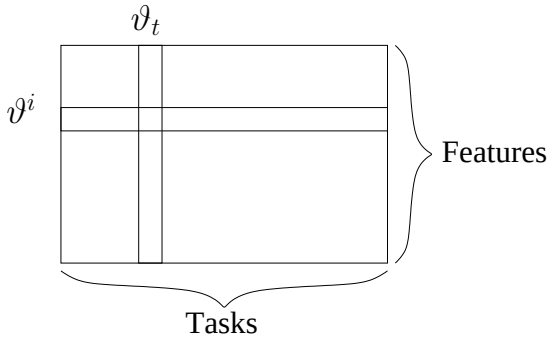


Figure 5.4: Matrix Θ of learning parameters. Columns ϑ_t contain the learning parameters for task t, while rows ϑ^i contain the weight for feature i across the tasks.

the second assumption is specific of the multi-task problem and forces the relevant features to be exactly the same in all the tasks.

These two assumptions are formalized in MTFL as the problem of computing the learning parameters ϑ_t and the matrix U that minimizes the error function

$$\mathcal{E}(\Theta, U) = \sum_{t=1}^{n} \sum_{i=1}^{m} L\left(y_{ti}, \langle \vartheta_t, U^{\mathsf{T}} x_{ti} \rangle\right) + \lambda \|\Theta\|_{2,1}^2, \tag{5.11}$$

where $\Theta \in \mathbb{R}^{d \times n}$ is the parameter matrix with vectors ϑ_t on columns (Figure 5.4) and the matrix norm $\|\Theta\|_{2,1}^2$ is defined as

$$\|\Theta\|_{2,1}^2 = \left(\sum_{i=1}^{d} \|\vartheta^i\|_2\right), \tag{5.12}$$

where ϑ^i is the i-th row of Θ that contains the parameters of ϑ_t corresponding to feature ϕ_i across the tasks.

While the loss function L is an average of the approximator errors in all the tasks, the regularization term is computed over the whole parameter matrix and penalizes configurations in which the relevant features are different in each task. Since the basic assumption is that all the tasks share a common sparse representation (i.e., the set of relevant features is relatively small and it is the same in each task), we expect all the vectors ϑ_t to have only few non-zero elements corresponding to the same features. Thus, we force matrix Θ to have only few non-zero rows ϑ^i.

Algorithm 10 The Linear Multi-Task Feature Learning algorithm

Input: dataset $(\mathbf{x}^k, \mathbf{y}^k)$
Parameters: λ, ϵ, tol
Output: W^*, D^*

Initialize $D = \frac{I_d}{d}$
while $\|W - W_{prev} > tol\|$ **do**
  for $t = 1$ to n **do**
    Compute $w_t = arg\min\left(\sum_{i=1}^{m} L(y_{ti}, \langle w, x_{ti}\rangle) + \lambda\langle w, D^{-1}w\rangle\right)$
  end for
  Set $D = \frac{(WW^\intercal + \epsilon I_d)^{\frac{1}{2}}}{\mathrm{trace}(WW^\intercal + \epsilon I_d)^{\frac{1}{2}}}$
end while

At the same time, the optimization on U forces to identify the feature space in which the assumption of common features actually holds.

Although the problem of minimizing (5.11) is effective in modeling the initial assumptions, it is difficult to be solved for two main reasons: *(i)* the joint problem on Θ and U is non-convex, *(ii)* the regularization term is non-smooth. Therefore, the optimization problem cannot be directly solved with exact methods and, similarly to other multi-task problems [89], approximated solutions should be applied. Nonetheless, in [9], it is shown that there exists an equivalent convex optimization problem whose solution is strictly related with the global optimal solution of 5.11.

The equivalent problem is

$$\mathcal{J}(W, D) = \sum_{t=1}^{n}\sum_{i=1}^{m} L\left(y_{ti}, \langle w_t, x_{ti}\rangle\right) + \lambda \sum_{t=1}^{n}\langle w_t, D^{+} w_t\rangle, \tag{5.13}$$

where $W \in \mathbb{R}^{d\times n}$ has vectors w_t on columns, $D \in \mathbb{R}^{d\times d}$ is a diagonal matrix and D^+ is its pseudoinverse. It can be proved that the problem of minimizing $\mathcal{J}(W, D)$ is a convex problem and that the optimal solution (W^*, D^*) has a direct relationship with the optimal solution of the original problem

$$(W^*, D^*) = \left(U^*\Theta^*, U\mathrm{Diag}\left(\frac{\|\vartheta^{*,i}\|_2}{\|\Theta^*\|_{2,1}}\right)_{i=1}^{d} U^{*\intercal}\right). \tag{5.14}$$

Furthermore, it can be proved that the solution of the equivalent optimization problem can be obtained by iteratively minimizing on W and D separately. The algorithm works as follows. We initialize D to D_{init} and

we define the problem of computing W that minimizes $\mathcal{J}(W, D_{init})$. As it can be noticed from (5.13), if D is kept fixed, the optimization problem can be divided into n 2-norm regularization problems on w_t, that can be solved through kernel regression [2]. Given the optimal solution W_{opt}, we keep W constant and we optimize on D. By iterating on separated optimizations of W and D, this process converges to the optimal solution (W^*, D^*). More formally, in order to guarantee the convergence to the optimal solution, the following minimization problem is defined

$$\mathcal{J}_\epsilon(W, D) = \sum_{t=1}^{n} \sum_{i=1}^{m} L\left(y_{ti}, \langle w_t, x_{ti} \rangle\right) + \lambda \text{trace}(D^{-1}(WW^\mathsf{T} + \epsilon I_d)). \quad (5.15)$$

It can be proved that the alternating algorithm summarized in Algorithm 10 converges to the optimal solution by letting $\epsilon \to 0$.

Although the definition of the problem and the structure of the solution remain mainly unaltered in case of kernel functions $\varphi(x)$, there are some critical passages in the solution of the optimization step on W of the alternating algorithm. In case of one single task, the problem reduces to the case illustrated in Section 5.4.2, while in the multi-task scenario the representer theorem must be extended to take into account the different scenario. The result is that the optimal matrix W can be computed as

$$W = \Phi C, \quad (5.16)$$

where $\Phi \in \mathbb{R}^{M \times mn}$ contains in each column the value of the features corresponding to sample x_{ti}, that is $\varphi(x_{ti})$, and $C \in \mathbb{R}^{mn \times n}$ is the matrix of coefficients c_t. The main problem of this formulation is the dimensionality M of the kernel function φ that in general is high or even infinite, thus leading to the impossibility to solve the optimization problem in C. Therefore, we need a transformation that reverts to a problem with finite dimensionality. Let $\mathcal{L} = \{\varphi(x_{ti}), t \in \mathbb{N}_n, i \in \mathbb{N}_m\}$ be the space of the features evaluated in the input samples and let δ be its dimensionality. Given a matrix H whose columns form an orthogonal basis for $\mathcal{L}$, there exist a matrix Ψ such that

$$W = \Phi C = H\Psi, \quad (5.17)$$

where $H \in \mathbb{R}^{M \times \delta}$ and $\Psi \in \mathbb{R}^{\delta \times n}$. As it can be noticed, with this transformation we obtained an optimization problem defined on Ψ that is

[2] Other algorithms, such as SVM, can be used to solve the 2-norm regularization problems.

Algorithm 11 The Kernel Multi-Task Feature Learning algorithm

Input: dataset $(\mathbf{x}^k, \mathbf{y}^k)$
Parameters: λ, ϵ, tol
Output: $\Psi^*, \{\widetilde{x}\}_{i=1}^{\delta}$

Find samples $\widetilde{x}$ so that $\varphi(\widetilde{x})$ is a basis for $\mathcal{L}$
Build R through Gram-Schmidt orthogonalization
Initialize $\Delta = \frac{I_\delta}{\delta}$
while $\|\Psi - \Psi_{prev} > tol\|$ **do**
  for $t = 1$ to n **do**
    Compute $\psi_t = arg\min\left(\sum_{i=1}^{m} L(y_{ti}, \langle \psi, z_{ti}\rangle) + \lambda\langle \psi, \Delta^{-1}\psi\rangle\right)$
  end for
  Set $\Delta = \frac{(\Psi\Psi^\mathsf{T} + \epsilon I_\delta)^{\frac{1}{2}}}{\mathrm{trace}(\Psi\Psi^\mathsf{T} + \epsilon I_\delta)^{\frac{1}{2}}}$
end while

independent from the dimensionality of the kernel function. The resulting optimization problem is

$$\min\left(\sum_{t=1}^{n}\sum_{i=1}^{m} L\left(y_{ti}, \langle \psi_t, z_{ti}\rangle\right) + \lambda\|\Psi\|_{\mathrm{tr}}^2\right), \tag{5.18}$$

where $z_{ti} = H^\mathsf{T}\varphi(x_{ti})$. Unfortunately, the problem of dimensionality is still present in the transformation from samples x_{ti} to samples z_{ti} since it requires the evaluation of the kernel function φ and the use of matrix H. Let us consider the set of δ samples $\widetilde{x}$ that defines a basis for the feature space $\mathcal{L}$ and let $\widetilde{\Phi} \in \mathbb{R}^{M\times\delta}$ be a matrix with $\varphi(\widetilde{x})$ on columns. Matrix H can be written as

$$H = \widetilde{\Phi}R, \tag{5.19}$$

where $R \in \mathbb{R}^{\delta\times\delta}$ can be computed through Gram-Schmidt orthogonalization [51] using only kernel values $K(x, x')$. Therefore, the transformed input samples become

$$z_{ti} = H^\mathsf{T}\varphi(x_{ti}) = R^\mathsf{T}\widetilde{\Phi}^\mathsf{T}\varphi(x_{ti}) = R^\mathsf{T}\widetilde{K}(x_{ti}), \tag{5.20}$$

where $\widetilde{K}(x_{ti}) \in \mathbb{R}^\delta$ is a vector whose components are $K(\widetilde{x}, x_{ti})$ where $\widetilde{x}$ is one of the samples used to compute $\widetilde{\Phi}$.

Finally, given the output of the algorithm (Ψ) and the samples $\widetilde{x}$ used to build an orthonormal basis for $\mathcal{L}$, we define a matrix $B \in \mathbb{R}^{\delta\times n}$ that plays a role similar to coefficients C but in the space $\mathcal{L}$

$$B = R\Psi. \tag{5.21}$$

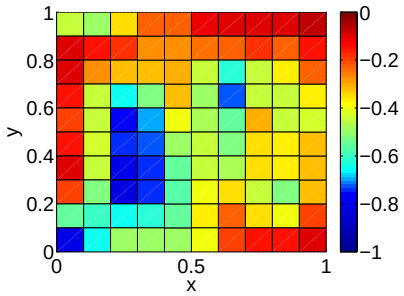


Figure 5.5: Map of colors used in the experiments in the colored maze environment.

Therefore, the resulting optimal matrix W becomes

$$W = \widetilde{\Phi} B, \tag{5.22}$$

and, thus, the approximation of the action-value function becomes

$$\widehat{Q}_t(x) = \langle \vartheta_t, U^\mathsf{T} \varphi(x) \rangle = \langle w_t, \varphi(x) \rangle = \langle b_t, \widetilde{\Phi}^\mathsf{T} x \rangle = \langle b_t, \widetilde{K}(x) \rangle. \tag{5.23}$$

As it can be noticed, the definition of the optimization problem in Ψ does not require the evaluation of functions φ and, at the same time, the computation of B and $\widetilde{K}$ guarantees that the approximate action-value function can be evaluated in each state-action x with no need to compute $\varphi(x)$.

The overall MTFL with kernel function is summarized in Algorithm 11. As it can be noticed, it is exactly the same as in Algorithm 10 except for the different space δ in which the optimization is carried out.

5.5 Experiments: the Colored Maze

5.5.1 Definition and Settings

The first problem we consider is a variant of the colored maze experiment introduced in [164]. We consider an empty maze with state variables x and y limited in the interval $[0, 1]$, one action variable a with four discrete values $\{0, 1, 2, 3\}$ that moves the agent of a step 0.1 perturbed by a Gaussian noise with zero mean and standard deviation 0.05 along the chosen direction. The maze is divided into 100 areas with different colors c_i that correspond to a negative number in $[-1, 0]$. The color settings used in this experiment is illustrated in Figure 5.5. The reward

Parameter	Value
λ_{MTFL}	0.01
λ_{Ind}	0.1
ϵ	0.0
tol	0.0001

Table 5.1: Parameters for multi-task fitted Q-iteration in the colored maze environment.

in each state is obtained as $R(s) = wc$, where c is a vector that contains the color of the area corresponding to the current state and the colors of the surrounding areas and w is a normal vector of weights. The goal of the agent is to achieve the upper left corner of the maze by following the highest reward path. The task distribution Ω is defined as a uniform distribution over $w \in \mathbb{R}^5$. Therefore, all the tasks share the same domain but have different goals.

From Ω we draw $n = 20$ independent tasks and we consider six different configurations:

- *Independent*: fitted Q-iteration with kernel regression running on each task independently
- *MTFL-i*: fitted Q-iteration with MTFL running on i tasks

where $i = 1, 2, 5, 10, 15, 20$. It is worth noting that the version *MTFL-1* is not necessarily equal to the independent version. In fact, while kernel regression use a 2-norm regularizer, MTFL applied to only one task at a time, reduces to a regularized algorithm in 1-norm. For all the algorithms we adopted a Gaussian kernel with standard deviation 0.1.

The parameters used in the experiment are summarized in Table 5.1. In all the following experiments, the discount factor is $\gamma = 0.9$, the number of iterations of fitted Q-iteration is $N = 10$.

5.5.2 Results: Generalization Improvement

The objective of the following experiment is to assess the effectiveness of multi-task batch RL in reducing the error of the approximated action-value functions. In particular, we analyze the approximation as the number of tasks increases. In fact, we expect that the optimal value functions share a common representation that can be identified and exploited by the MTFL algorithm. All the tasks are drawn according to Ω, the learned action-value functions are tested on a set of testing samples

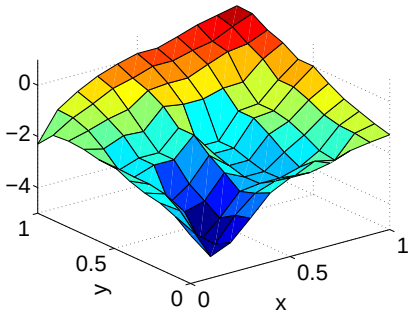


Figure 5.6: Example of optimal value function for the colored maze for a given w.

randomly extracted from the same tasks used in the training phase. The results are averaged over 30 runs.

In Figure 5.7, we report the mean square errors for the six configurations of the algorithm and for different number of samples extracted from the tasks ($m = 50, 100, 150, 200, 250, 300$). Although *MTFL-1* obtains results slightly worse than learning independently in some cases, as the number of tasks considered in MTFL increases, the testing error is significantly reduced. As it can be noticed, the improvement is almost monotonic with the number of tasks and *MTFL-20* reduces the MSE to half of the error of the algorithm that learns each task separately. In fact, according to the environment the tasks are strictly related and FQI in conjunction with MTFL succeeds in exploiting this relatedness by adapting the feature space so as to maximize the advantage of learning all the tasks at the same time.

5.5.3 Results: Irrelevant Variables

The second experiment in the colored maze environment has the objective of analyzing the approximation error when irrelevant variables are introduced in the definition of the problem. In fact, since MTFL automatically adapt the feature space according to a joint optimization problem defined on all the tasks, we expect it to benefit from the information stored in all the samples in order to reduce the negative effect due to introduction of irrelevant variables. In the following experiment we add up to 2 irrelevant variables (z_1 and z_2) to x and y. The value of z_1 and z_2 is randomly drawn in the interval $[0, 1]$ at each step. We consider the

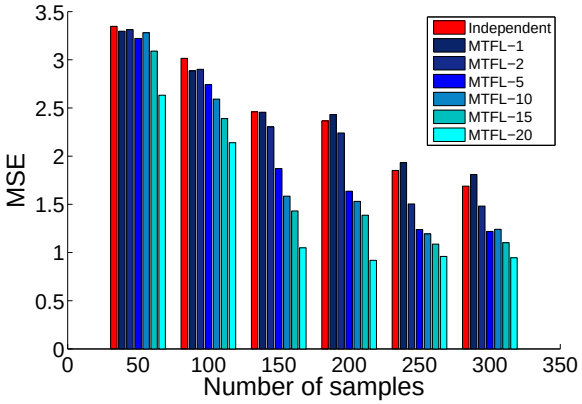


Figure 5.7: Approximation error averaged over all the tasks for different algorithms.

case in which $m = 500$ samples are available for each task. In Figure 5.8 we report the mean squares error for the case of two, three and four variables (x, y, z_1, z_2). As it can be noticed, the introduction of irrelevant variables significantly worsens the performance of all the configurations, such that when both z_1 and z_2 are added to the state space the MSE is three times higher than the MSE of the default setting. Nonetheless, even when two irrelevant variables are added, *MTFL-20* still keeps a significant advantage with respect to independent learning.

5.6 Experiments: the Boat Problem

5.6.1 Definition and Settings

The second environment we consider is the boat problem introduced in Section 4.7. The chosen action a_{t+1} is perturbed by a uniform noise $U[-10; 10]$. Unlike the problem in Section 4.7, there are no sandbanks and the reward function is defined as

$$R(x,y) = \begin{cases} +10 & (x,y) \in \mathcal{Z}_s \\ D(x,y) & (x,y) \in \mathcal{Z}_v \\ \text{-10} & (x,y) \in \mathcal{Z}_f \\ \text{-0.1} & \text{otherwise} \end{cases} \tag{5.24}$$

where D is a function that gives a reward decreasing linearly from 10 to -10 relative to the distance from the success zone. The discount factor is $\gamma = 0.99$.

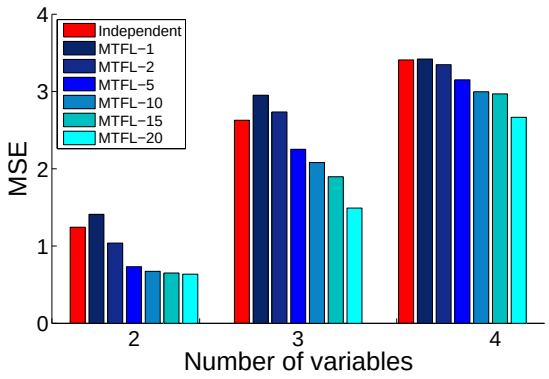


Figure 5.8: Approximation error averaged over all the tasks for different algorithms with the introduction of irrelevant variables.

Parameter	Value
λ_{MTFL}	0.1
λ_{Ind}	0.005
ϵ	0.0
tol	0.0001

Table 5.2: Parameters for multi-task fitted Q-iteration in the boat environment.

The parameters used in the following experiments are summarized in Table 5.2. Both the independent version of FQI and the version with MTFL are run on five iterations and use a Gaussian kernel with parameter $\sigma = 15.0$.

5.6.2 Results: Policy Performance

The objective of the following experiment is to evaluate the actual performance achieved by the multi-task batch RL algorithm. While in the colored maze environment we focused only on the approximation of the action-value function, here we analyze the performance on the learned policy with and without multi-task learning. Since MTFL should improve the accuracy of the approximation, we also expect the corresponding policy to achieve a better performance than learning each task from scratch.

Since the problem is more complex than the previous maze problem

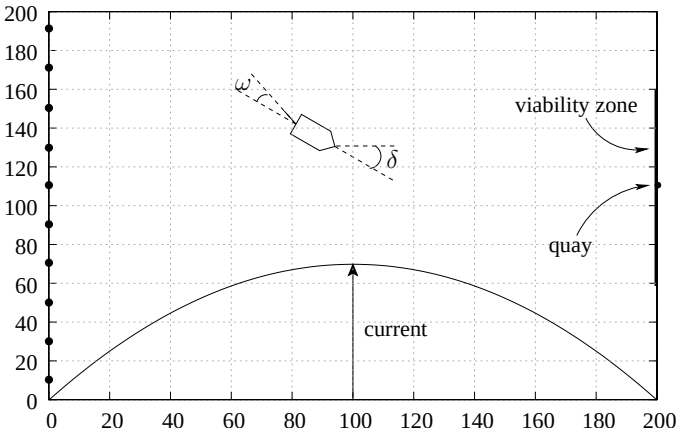


Figure 5.9: The boat environment.

a greater number of samples is needed and this negatively impacts on the complexity of the pre-processing of Algorithm 11. In particular, the computation of R through Gram-Schmidt orthogonalization becomes particularly expensive. In order to avoid this problem, in the following experiments we used a generative model of the boat environment to draw the samples of different task. At first we collected m samples of state-action pair $\langle s, a \rangle$ at random from the environment, independently from the specific task considered. Then, we used the generative model to build the outcomes s' and r for each task. As a result, the input set $x = \{\langle s, a \rangle_i\}_{i \in \mathbb{N}_m}$ is shared across all the tasks and the computation of the basis for $\mathcal{L}$ can be reduced to this input set, instead of using nm samples.

We consider a task space $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$ and a uniform distribution Ω on $\mathcal{T}$. Each task shares the same domain but differs for the position of the quay: T_1: $y_{quay} = 70$, T_2: $y_{quay} = 80$, T_3: $y_{quay} = 90$, T_4: $y_{quay} = 100$. It is worth noting that in order to reach even relatively near positions of the bank the sequence of actions can be different because of the highly non-linear dynamics of the environment.

In Figure 5.10 we report the total reward achieved by the policy learned with and without MTFL with $m = 100, \ldots, 2000$. The performance is an average of the performance on all the tasks and is obtained by averaging 30 runs. The initial performance is almost the same. In fact, with a limited number of samples FQI does not succeed in learning a good policy and even the joint use of samples coming from different tasks does

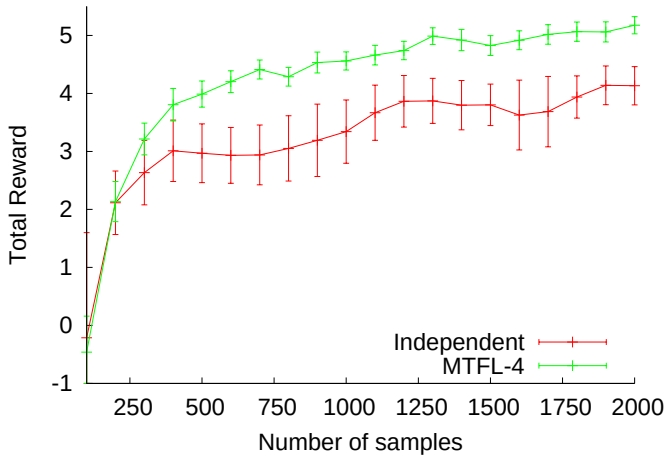


Figure 5.10: Total reward in the boat environment for different algorithms of FQI.

not provide enough information about the underlying representation of the optimal action-value functions. On the other hand, as the number of samples increases the advantage of learning through MTFL becomes more and more relevant. It is interesting to notice that the difference in performance is still present at convergence ($m = 2000$) (the difference is statistically significant with $p < 0.05$). In fact, although both the algorithms shares the same regressor (kernel regression with Gaussian kernel) and, in principle, they have the same capability to approximate functions, MTFL succeeds to adapt the feature space so as to bias the learning towards a set of functions that are more similar to the action-value functions to be approximated.

5.7 Related Works

This work is obviously strictly related to the main works in multi-task learning [34, 18, 3, 9]. Nonetheless, our focus is not on developing a new method for feature learning but to show how existing solutions can be integrated into the RL paradigm. The main differences between the approach of MTFL with respect to other multi-task learning algorithms are: *(i)* a joint optimization problem on learning and feature parameters, *(ii)* an exact solution through an alternating algorithm, *(iii)* use of high-dimensional kernel spaces.

As far as reinforcement learning is concerned, to the best of our knowledge this is the first attempt to adopt recent results in multi-task learning literature into a reinforcement learning algorithm. Many works faced the problem of extracting basis functions for linear function approximators in batch reinforcement learning algorithms. The most similar approach to our algorithm is introduced in [84], where a set of basis functions is parameterized and optimized together with the learning parameters by gradient descent and cross-entropy methods in single-task problems. In [59] an algorithm for the identification of irrelevant state variables is proposed. Although relevant in many domains, MTFL faces the more general problem of feature learning (instead of feature selection) in the context of multi-task learning. In [95], a given set of basis functions is expanded by adding basis functions obtained from the Bellman error in order to improve the accuracy of the approximation of the optimal value function of a given task. In [78] and [67], the focus is on the problem of feature extraction, while each task is solved independently. It is worth noting that, besides the scenario (single- vs multi-task), we also differ in the batch RL algorithm adopted and in the use of the function approximator. All the approaches described before rely on LSPI as learning algorithm, while we focused on fitted Q-iteration. All the regularized multi-task learning algorithms define a joint approximation error with a loss function defined on the current approximation and the target function. In RL no target function is actually available. The only way to define a similar loss function is to use the Bellman residual error ($T\widehat{Q} - \widehat{Q}$). The fixed-point LSPI does not rely on the definition of an approximation error and thus it cannot be directly integrated with multi-task learning algorithms. On the other hand, since at each iteration of FQI a regression problem based on the Bellman error is solved, the extension to a multi-task perspective is more straightforward. Furthermore, the solutions proposed in [78, 95] require to store basis functions by enumerating state-action pairs. Although this allows to consider feature spaces crafted for tasks at hand, their representation can have a complexity comparable to learning the action-value function itself. On the other hand, the use of kernel spaces, in which it is not necessary to explicitly define the basis functions, avoids this problem and, at the same time, makes it possible to use high-dimensional feature spaces.

Finally, the proposed method is related to the recent work on multi-task RL based on a hierarchical Bayesian approach [164]. Both of them merge results from the SL literature on multi-task learning. Nonetheless, the objective of hierarchical Bayesian approaches is to identify a distri-

bution over models that allows to rapidly infer the characteristics of new environments based on previous environments.

5.8 Conclusions

In this chapter we proposed an integration of FQI with a multi-task learning algorithm, with the main objective of improving the generalization performance of the learning algorithm on a finite set of tasks. At the best of our knowledge this is the first attempt to pursue the objective of multi-task learning by merging RL and SL solutions. The experimental results show that the multi-task version of FQI actually reduces the approximation error and improves the learning performance with respect to FQI when applied on each task independently.

Future directions of investigation are:

1. *Extension to transfer.* The main drawback of the proposed solution is that the learned feature space cannot be easily transferred to new tasks extracted from the same distribution. In fact, matrix U cannot be explicitly computed in case of kernel functions because of the dimensionality of the feature space. It would be interesting to define a suitable method to transfer the output of Algorithm 11 to bias the kernel regression on new tasks.

2. *Theoretical analysis.* The implicit assumption underlying the integration between MTFL and FQI is that at each iteration k action-value functions Q_t^k share the same representation. It is possible to define problems in which only the optimal action-value functions ($k \to \infty$) are actually related but the intermediate functions are significantly different. In these situations, MTFL is likely not to provide any significant generalization improvement. A theoretical analysis about the characteristics of the action-value functions for which the benefit from MTFL is maximized should be carried out.

3. *Multi-task feature extraction.* While MTFL focuses on the problem of adapting a given set of features, another relevant topic of multi-task learning is the extraction of features starting from some characteristics of the tasks at hand. This perspective is implicitly adopted in the proto-value functions (PVF) framework [78] but a very generic class of tasks is taken into consideration. A possible extension is to integrate the PVF framework with recent works of multi-task structure learning with spectral regularization [7].

6 Conclusions

6.1 Contributions

Transfer of knowledge in Reinforcement Learning (RL) is a challenging problem and the results obtained so far in literature showed that it can be an effective approach to improve the learning performance in complex tasks. Nonetheless, a deep understanding of how transfer should be accomplished in RL and how it affects the final performance is mostly an open problem. Furthermore, the connections between transfer in supervised learning and in RL are still unexplored.

The main contributions of this thesis to the research in transfer RL can be summarized as follows:

- **Classification of the state of the art**. In Chapter 3, we introduced a novel classification of works of transfer in RL. We focused on three different dimensions: the objectives, the scenarios, the knowledge retained and transferred across tasks. The analysis of the state of the art under this perspective showed that most of works of transfer in RL focused on the objective of improving the learning speed and the initial offset. The main techniques adopted to achieve these objectives are either the direct transfer of solutions or the transfer of subpolicies (i.e., options). On the other hand, little attention has been devoted to other topics, such as, transfer of trajectory samples and multi-task perspective for generalization improvement.

- **Transfer in batch RL**. RL algorithms performance depends on many aspects, such as, the exploration policy, the function approximator, the update rule, and so on. These elements contribute all together to determine the performance. As a result, it is often difficult to analyze how transfer solutions affect the final learning performance and which aspects should be considered to achieve the different objectives of transfer. In Chapter 3, we proposed the batch RL framework as the most suitable perspective to analyze the problem of transfer in RL. In fact, in batch RL algorithms it is possible to distinguish different aspects of the learning process

(i.e., sampling and learning) so that the effect of transfer solutions can be isolated to specific elements of the algorithm.

- **Transfer of samples**. In Chapter 4, we introduced a novel sample-based transfer algorithm pursuing the objective of improving the learning speed. We considered the typical inductive transfer scenario in which a set of source tasks are used to improve the performance in one target task. Under the assumption that tasks are somehow related, the idea is to transfer part of the samples collected in the source tasks to augment the set of samples collected in the target task used to feed the batch RL algorithm. The main problem is to avoid transfer of samples from tasks that are dissimilar from the target task. Therefore, we introduced a method for the identification of which source tasks are more similar to the target task (*compliance*) and, among the samples of one source task, which samples are more convenient to transfer (*relevance*). The proposed method has been tested on a number of domains aiming at investigating properties and performance of the algorithm. The results show that the relevance-based transfer of samples can significantly improve the learning performance by avoiding negative transfer effects.

- **Multi-task fitted Q-iteration**. In Chapter 5, we proposed the integration of batch RL with the multi-task learning perspective pursuing the objective of improving the generalization performance. This objective received little attention in works of transfer in RL so far and this is the first attempt of integration of a RL algorithm with a multi-task learning algorithm. The basic assumption is that the optimal action-value functions of the tasks at hand share an underlying common representation. If this representation can be learned starting from the information collected from each task, the accuracy of approximation can be significantly improved. In particular, we focused on fitted Q-iteration (FQI) and we defined a multi-task learning problem at each iteration of FQI. Then, we adopted Multi-Task Feature Learning (MTFL) to solve the joint optimization problem on the learning parameters of each task and on the feature space shared across tasks. Preliminary results show that the integration of MTFL with FQI leads to a significant improvement both in terms of reduction of the approximation error and improvement of policy performance.

6.2 Future Works

Although this thesis represents a contribution to the advancement of research in transfer RL, many aspects of the problem remain unexplored. Following the perspective introduced in this thesis, future directions of investigation are:

- **Formalization of transfer objectives in batch RL**. Although in this thesis we proposed a first definition of the objectives in batch RL, much work should be devoted in providing a more rigorous formal definition. In fact, the algorithms introduced in Chapter 4 and Chapter 5 are not guaranteed to achieve their corresponding objectives and only experimental evidence of their effectiveness is provided. Possible directions of research could rely on the theoretical results and formalizations proposed in [64, 5, 18].
- **Transfer among tasks with different state and action spaces.** The main limit of the algorithms proposed in this thesis is that they require all the tasks to share the same state and action spaces. On the other hand, recent works [144] specifically focused on the definition of suitable mappings between tasks defined on different state and action spaces. The integration of these works with the transfer algorithm proposed in this thesis could lead to more sophisticated transfer algorithms able to face complex real-world problems.
- **Integration of sample-based transfer and multi-task fitted Q-iteration**. The transfer solutions introduced in Chapter 4 and 5 are orthogonal. In fact, while the transfer of samples changes the output space of the sampling algorithm, the multi-task algorithm adapts the output space of the learning algorithm. Therefore, the two solutions could be easily integrated into one single transfer algorithm.
- **Improvement of initial offset**. In this thesis we considered the objectives of improvements of learning speed and generalization performance, no transfer algorithm for the improvement of the initial offset is proposed. Nonetheless, the sample transfer algorithm could be extended along the direction introduced in [164], where a prior is computed over the tasks distribution through a hierarchical Bayesian model. The samples could be obtained from a generative model defined by merging all the source tasks, obtaining the most likely task. This way, the performance of the batch RL algorithm could be initialized by learning on this set of samples.

Bibliography

[1] James S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, 1975.

[2] Charles W. Anderson and Stewart G. Crawford-Hines. Multigrid q-learning. Technical Report CS-94-121, Colorado State University, Fort Collins, 1994.

[3] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[4] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning Journal*, 71(1):89–129, 2008.

[5] Munos R. Antos A., Szepesvari C. Value-iteration based fitted policy iteration: Learning with a single trajectory. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 330–337, April 2007.

[6] Andreas Argyriou. Personal communication, 2007.

[7] Andreas Argyriou, Charles A. Micchelli, Massi Pontil, and Yiming Ying. A spectral regularization framework for multi-task structure learning. In *Advances in Neural Information Processing Systems*, Vancouver, British Columbia, Canada, December 2007.

[8] Andreas Argyrious, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems*, pages 41–48, Vancouver, British Columbia, Canada, December 2006. MIT Press.

[9] Andreas Argyrious, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning, to appear*, 2007.

[10] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *In Proc. of IAPR/IEEE Workshop on Visual Behaviors*, pages 112–118, 1994.

[11] Mehran Asadi and Manfred Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2054–2059, Hyderabad, India, January 2007. Morgan Kaufmann.

[12] Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In Bernhard Scholkopf, John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems*, pages 49–56, Vancouver, British Columbia, Canada, December 2006. MIT Press.

[13] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 30–37, 1995.

[14] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4:83–99, 2003.

[15] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 672–677, Hyderabad, India, January 2007. Morgan Kaufmann.

[16] A.G. Barto, R. Sutton, and C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst., Man. Cybern.*, SMC-13:834–846, September - October 1983.

[17] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[18] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[19] Jonathan Baxter, Rich Caruana, Tom Mitchell, Lorien Y. Pratt, Daniel L. Silver, and Sebastian Thrun, editors. *Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems*, 1995.

[20] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA, 1957.

[21] Shai Ben-David. Inductive transfer via embeddings into a common feature space. Presentation at "OPEN HOUSE on Multi-Task and Complex Outputs Learning", 2006.

[22] Shai Ben-David and Reba Schuller. Exploiting task relatedness for multiple task learning. In *Proceedings of the sixteenth Annual Conference on Computational Learning Theory*, pages 567–580, 2003.

[23] D. S. Bernstein. Reusing old policies to accelerate learning on new mdps. Technical report, University of Massachusetts, Amherst, MA, USA, 1999.

[24] D.P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[25] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[26] A. Bonarini, A. Lazaric, and M. Restelli. Incremental skill acquisition for self-motivated learning animats. In *From Animals to Animats 9. 9th International Conference on Simulation of Adaptive Behavior (SAB'06)*, volume 4095 of *Lecture Notes in Artificial Intelligence*, pages 357–368, Berlin, 2006. Springer Verlag.

[27] Michael Bowling and Manuela Veloso. Bounding the suboptimality of reusing subproblems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1340–1347, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[28] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems*, pages 369–376, Cambridge, MA, 1995. The MIT Press.

[29] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

[30] R.I. Brafman and M. Tennenholtz. a general polynomial time algorithm for near-optimal reinforcement learning, r-max:. *Journal of Machine Learning Research*, 3:213–231, 2002.

[31] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial: second edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[32] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[33] J. L. Carroll and K. Seppi. Task similarity measures for transfer in reinforcement learning task libraries. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 803–808, 2005.

[34] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[35] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, pages 271–278, Denver, Colorado, USA, 1993.

[36] Richard Dearden, Nir Friedman, and David Andre. Model-based bayesian exploration. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 150–159, San Francisco, CA, 1999. Morgan Kaufmann.

[37] O. Dekel, P. M. Long, and Y. Singer. Online multitask learning. In *Proceedings of the 19th Annual Conference on Learning Theory*, pages 453–467, Pittsburgh, PA, USA, June 2006. Springer.

[38] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[39] Chris Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, 16:59–104, 2002.

[40] D. Ernst. Selecting concise sets of samples for a reinforcement learning agent. In *Proceedings of CIRAS 2005*, Singapore, December 2005.

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

[42] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637, 2005.

[43] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, New York, NY, USA, 2004. ACM.

[44] Tom Fawcett, James Callan, Chris Matheus, Ryszard Michalski, Michael Pazzani, Larry Rendell, and Rich Sutton, editors. *Constructive Induction Workshop at the International Conference on Machine Learning*, 1994.

[45] Kimberly Ferguson and Sridhar Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[46] Fernando Fernandez and Manuela Veloso. Exploration and policy reuse. Technical Report CMU-CS-05-172, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 2005.

[47] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 162–169, Arlington, Virginia, United States, 2004. AUAI Press.

[48] David J. Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49(2-3):325–346, 2002.

[49] D. Gentner, J. Loewenstein, and L. Thompson. Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95(2):393–408, 2003.

[50] M. L. Gick and K. J. Holyoak. Schema induction and analogical transfer. *Cognitive Psychology*, 15:1–38, 1983.

[51] G. H. Golub and C. F. van Loan. *Matrix Computation*. The Johns Hopkins University Press, 1996.

[52] Geoffrey Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1999.

[53] Geoffrey J. Gordon. Chattering in sarsa(lambda). Technical report, Carnagie Mellon University, 1996.

[54] Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems*, pages 1040–1046, 2000.

[55] Milos Hauskrecht. Planning with macro-actions: Effect of initial value function estimate on convergence rate of value iteration. Technical Report working paper, Department of Computer Science, University of Pittsburgh, 1998.

[56] Geremy Heitz, Gal Elidan, and Daphne Koller. Transfer learning of object classes: From cartoons to photographs. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[57] Bernhard Hengst. *Discovering hierarchy in reinforcement learning.* PhD thesis, University of New South Wales, 2003.

[58] Nicholas Jong and Peter Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *ICML Workshop on Kernel Machines and Reinforcement Learning*, June 2006.

[59] Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 752–757, August 2005.

[60] Nicholas K. Jong and Peter Stone. Model-based function approximation for reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1–8, Honolulu, Hawaii, May 2007.

[61] Tobias Jung and Daniel Polani. Least squares svm for least squares td learning. In *Proceedings 17th European Conference on Artificial Intelligence*, pages 499–503, 2006.

[62] Sham Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[63] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[64] Zs. Kalmar and Cs. Szepesvari. An evaluation criterion for macro-learning and some results. Technical Report TR-99-01, Mindmaker Ltd., 1999.

[65] Shivaram Kalyanakrishnan and Peter Stone. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

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

[67] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 449–456, New York, NY, USA, 2006. ACM.

[68] George Konidaris and Andrew Barto. Autonomous shaping: knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496, New York, NY, USA, 2006. ACM.

[69] George Konidaris and Andrew G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, Hyderabad, India, January 2007. Morgan Kaufmann.

[70] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[71] Pat Langley. Transfer of knowledge in cognitive systems. Talk, June 2006. Presented at the ICML-2006 Workshop on Structural Knowledge Transfer for Machine Learning.

[72] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential

monte carlo methods. In *Advances in Neural Information Processing Systems*, 2007.

[73] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[74] Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 415–20, July 2006.

[75] Michael G. Madden and Tom Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artif. Intell. Rev.*, 21(3-4):375–398, 2004.

[76] Pattie Maes and Rodney A. Brooks. Learning to coordinate behaviors. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 796–802, Boston, Massachusetts, USA, 1990.

[77] Sridhar Mahadevan. Enhancing transfer in reinforcement learning by building stochastic models of robot actions. In *ML92: Proceedings of the ninth international workshop on Machine learning*, pages 290–299, San Francisco, CA, USA, 1992. Morgan Kaufmann.

[78] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 38:2169–2231, 2007.

[79] Z. Marx, M.T. Rosenstein, L.P. Kaelbling, and T.G. Dietterich. Transfer learning with an ensemble of background tasks. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[80] Maja J. Mataric. Reinforcement learning in multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

[81] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368. Morgan Kaufmann Publishers Inc., 2001.

[82] Neville Mehta, Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. Transfer in variable-reward hierarchical reinforcement learning. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[83] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, pages 295–306. Springer-Verlag, 2002.

[84] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, February 2005.

[85] Charles A. Micchelli and Massimiliano A. Pontil. On learning vector-valued functions. *Neural Comput.*, 17(1):177–204, 2005.

[86] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[87] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[88] Alexandru Niculescu-Mizil and Rich Caruana. Inductive transfer for bayesian network structure learning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS 2007)*, 2007.

[89] G. Obozinski, B. Taskar, and M. I. Jordan. Multi-task feature selection. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, PA, 2006.

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

[91] Özgür Şimşek and Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pages 751–758, New York, NY, USA, 2004. ACM.

[92] Özgür Şimşek and Andrew G. Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pages 833–840, New York, NY, USA, 2006. ACM.

[93] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pages 816–823, New York, NY, USA, 2005. ACM.

[94] Stephan Pareigis. Multi-grid methods for reinforcement learning in controlled diffusion processes. In *Advances in Neural Information Processing Systems 9*, pages 1033–1039, Denver, Colorado, USA, 1997.

[95] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, pages 737–744, New York, NY, USA, 2007. ACM.

[96] T. J. Perkins and M. D. Pendrith. On the existence of fixed points for q-learning and sarsa in partially observable domains. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 490–497, 2002.

[97] T. J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical report, University of Massachusetts, Amherst, MA, USA, 1999.

[98] Caitlin Phillips. Knowledge transfer in markov decision processes. http://www.cs.mcgill.ca/~martin/usrs/phillips.pdf, McGill School of Computer Science, 2006.

[99] M.J.D. Powell. Radial basis functions for multivariate interpolation: A review. *Algorithms for Approximation*, pages 143–167, 1988.

[100] M.L. Puterman. *Markov Decision Problems*. Wiley, New York, 1994.

[101] M.L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, (24):1127–1137, 1994.

[102] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of the 18th European Conference on Machine Learning*, pages 699–707, 2007.

[103] Bohdana Ratitch and Doina Precup. Using mdp characteristics to guide exploration in reinforcement learning. In *Proceedings of the 14th European Conference on Machine Learning*, pages 313–324, 2003.

[104] Balaraman Ravindran and Andrew G. Barto. Relativized options: Choosing the right transformation. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 608–615, 2003.

[105] Martin Riedmiller. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, pages 317–328, 2005.

[106] Mark Ring. Toward a formal framework for continual learning. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[107] M .T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[108] Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 515–521, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[109] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*, pages 416–426, London, UK, 2001. Springer-Verlag.

[110] B. Schölkopf. *A short tutorial on kernels.* NIPS'00 Kernel Workshop, 2000.

[111] Oliver G. Selfridge, Richard S. Sutton, and Andrew G. Barto. Training and tracking in robotics. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 670–672, Los Angeles, CA, USA, 1985. Morgan Kaufmann.

[112] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In *Proceedings of the Symposium on Abstraction, Reformulation and Approximation*, pages 194–205, Airth Castle, Scotland, UK, 2005.

[113] Alexander A. Sherstov and Peter Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 1024–1029, July 2005.

[114] Daniel Silver. *Selective Transfer of Neural Network Task Knowledge*. PhD thesis, University of Western Ontario, 2000.

[115] Daniel Silver and Ryan Poirier. Requirements for machine lifelong learning. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, Whistler, British Columbia, Canada, December 2005.

[116] Daniel L. Silver and Ryan Poirier. Requirements for machine lifelong learning. In *IWINAC*, pages 313–319, 2007.

[117] Danny Silver, Goekhan Bakir, Kristin Bennett, Rich Caruana, Massimiliano Pontil, Stuart Russell, and Prasad Tadepalli, editors. *Inductive Transfer: 10 Years Later Workshop*, 2005.

[118] S. Singh, A.G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, 2004.

[119] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7*, pages 361–368, Denver, Colorado, USA, 1995.

[120] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4):323–339, 1992.

[121] S.P. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[122] William D. Smart. Explicit manifold representations for value-function approximation in reinforcement learning. In *AMAI*, 2004.

[123] William D. Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 903–910, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[124] Vishal Soni and Satinder P. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*, 2006.

[125] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, New York, NY, USA, 2006. ACM.

[126] Alexander L. Strehl and Michael L. Littman. An empirical evaluation of interval estimation for markov decision processes. In *The 16th IEEE International on Tools with Artificial Intelligence Conference*, pages 531–539, 2004.

[127] Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pages 856–863, New York, NY, USA, 2005. ACM.

[128] Malcolm J. A. Strens. A bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 943–950, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[129] Funlade T. Sunmola and Jeremy L. Wyatt. Model transfer for markov decision tasks via parameter matching. In *25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, 2006.

[130] R S Sutton and A G Barto. An adaptive network that constructs and uses an internal model of its world. *Cognition and Brain Theory*, 4:217–246, 1981.

[131] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[132] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning*, pages 216–224, 1990.

[133] Richard S. Sutton. Generalization in reinfrocement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044, 1995.

[134] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[135] Richard S. Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, pages 871–878, New York, NY, USA, 2007. ACM.

[136] Richard S. Sutton, D. Precup, and S.P. Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[137] Csaba Szepesvári and William D. Smart. Interpolation-based q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 100, New York, NY, USA, 2004. ACM.

[138] Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1065–1070, Hyderabad, India, January 2007. Morgan Kaufmann.

[139] F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of mdps. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 3, pages 1108–1113, July 2003.

[140] Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *AAMAS 2008 Workshop on Adaptive Learning Agents and Multi-Agent Systems (ALAMAS)*, May 2008.

[141] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[142] Matthew E. Taylor and Peter Stone. Representation transfer for reinforcement learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, November 2007.

[143] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[144] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

[145] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.

[146] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38, 1995.

[147] Sebastian Thrun. Efficient exploration in reinforcement learning. Technical report, Carnagie-Mellon University, 1992.

[148] Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 489–497, 1996.

[149] Sebastian Thrun and L. Pratt. *Learning to Learn: Introduction and Overview*. Kluwer Academic Publishers, 1998.

[150] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 385–392, Vancouver, British Columbia, Canada, December 1994. MIT Press.

[151] S. Timmer and M. Riedmiller. Fitted q iteration with cmacs. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–8, Honolulu, HI, April 2007.

[152] Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the 17th European Conference on Machine Learning*, pages 425–436, 2006.

[153] Lisa Torrey, Trevor Walker, Jude W. Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the 15th European Conference on Machine Learning*, pages 412–424, 2005.

[154] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:670–690, Msy 1997.

[155] Paul Utgoff. Shift of bias for inductive concept learning. *Machine Learning*, 2:163–190, 1986.

[156] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998.

[157] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, 2002.

[158] Thomas J. Walsh, Lihong Li, and Michael L. Littman. Transferring state abstractions between mdps. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[159] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[160] Chris Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[161] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[162] M. Wiering. Convergence and divergence in standard and averaging reinforcement learning. In *Proceedings of the 15th European Conference on Machine Learning*, pages 477–488, 2004.

[163] Marco Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam, 1999.

[164] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, New York, NY, USA, 2007. ACM.

[165] Kai Yu and Volker Tresp. Learning to learn and collaborative filtering. In *NIPS Workshop on Inductive Transfer: 10 Years Later*, 2005.

[166] Kai Yu, Volker Tresp, and Anton Schwaighofer. Learning gaussian processes from multiple tasks. In *Proceedings of the 22nd International Conference on Machine learning*, pages 1012–1019, New York, NY, USA, 2005. ACM.

[167] Jian Zhang, Zoubin Ghahramani, and Yiming Yang. Learning multiple related tasks using latent independent component analysis. In *Advances in Neural Information Processing Systems*, 2005.

[168] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, pages 1114–1120, 1995.

[169] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report TR 1530, University of Wisconsin Madison, 2007.

[170] Omer Ziv and Nahum Shimkin. Multigrid algorithms for temporal difference reinforcement learning. In *Proceedings of the Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.