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

Multi-Task Reinforcement Learning with linear approximator under group sparsity assumption

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

Advisor: Ing. Marcello Restelli
Co-Advisor: Ing. Alessandro Lazaric

Authors: Daniele Calandriello 783264

Academic Year 2012-2013

To whom it shall matter.

Abstract

This thesis introduces an original Multi-Task Sparse Fitted Q Iteration algorithm, and includes a theoretical analysis on its performance guarantees, as well as the introduction of a new optimization method to efficiently compute it and experiments to validate it. The powerful modeling tools of Markov Decision Processes and function approximation allows Reinforcement Learning algorithms to efficiently solve complex control problems defined on continuous domains. One main drawback of function approximation, and therefore also of Approximate Reinforcement Learning techniques, is the large number of samples needed for the learning process as the dimensionality of the problem increases. Using a restricted class of approximators, such as the linear models used in this thesis, can provide good results with a number of samples linear in the number of features. A restricted model introduces bias, so a common approach is to use a large number of features to describe the problem, in the hope that the space spanned by this features will contain a good representation. To avoid overfitting the large number of parameters introduced, many regularization techniques can be used. In particular, when the representation is sparse in its parameters, the approximator can exploit a number of parameters exponential in the number of samples, as long as most of the parameters will be equal to zero. In this thesis we will further increase this bound by solving together a set of tasks that share a common sparse representation. This property, called group sparsity, will allow us to keep increasing the number of features past the exponential limit as long as suitable new tasks are added linearly. This group sparse approach will allow us to efficiently express the problem as a sequence of optimization problems with matrix norm regularization. In addition we will also explore the possibility of learning a new group sparse matrix representation when the original problem is apparently not group sparse. This expands the algorithm potential applications to settings that are not group sparse, but admits a group sparse representation.

Estratto in Lingua Italiana

Questa tesi si centra sull'introduzione di un nuovo algoritmo Multi-Task basato su Sparse Fitted Value Iteration. Include un'analisi teorica delle performance del nuovo algoritmo, una metodo per risolvere l'associato problema di ottimizzazione e una valutazione sperimentale.

Il paradigma del Reinforcement Learning (RL), o apprendimento per rinforzo, è uno dei grandi campi di interesse del Machine Learning. Particolarmente adatto a descrivere l'interazione tra un agente e il suo ambiente, il RL trova nei Markov Decision Process (MDP) uno strumento di modellazione particolarmente potente per descrivere una ampio spettro di problemi di controllo. Nella prima parte del Capitolo 2 introdurremo una panoramica dei metodi più tradizionali per RL, a partire dai metodi di Dynamic Programming basati sulla esatta conoscenza delle dinamiche dell'MDP, per poi passare a risolvere tali problemi attraverso forme approssimate che usano campioni derivanti dall'interazione per raccogliere l'informazione necessaria all'apprendimento di una politica ottima. Una grande limitazione di questi approcci è la necessità di rappresentare esattamente delle value function per ogni possibile stato del sistema. Quando il numero di questi stati cresce, i costi computazionali e di memoria delle tecniche di RL e DP basate su rappresentazione esatta esplodono a proporzioni ingestibili.

L'aggiunta dell'uso di approssimazione di funzioni permette agli algoritmi basati sul Reinforcement Learning di risolvere efficientemente complessi problemi di controllo con un numero di stati elevato, o addirittura infinito. Questo è per esempio il caso di tutti quegli stati che per vari motivi possono essere rappresentati efficientemente solo con variabili di stato continue. Nella seconda parte del capitolo 2 introdurremo perciò le più importanti tecniche di Approximated Reinforcement Learning. Tra queste, gli algoritmi della categoria Fitted Value Iteration forniranno la base per lo sviluppo del nuovo metodo presentato in questa tesi.

Uno dei principali svantaggi dell'uso di approssimatori, e quindi delle tecniche di Approximated Reinforcement Learning, è il rapido aumento del numero di campioni necessari ad apprendere quando la domensionalità del problema cresce. Questo problema, riferito spesso come curse of dimensionality, restringe il campo di applicabilità di queste tecniche a problemi in cui ottenere una grande quantità di campioni non è un problema. Un metodo per cercare di evitare questa esplosione di complessità

è restringere la classe di funzioni che l'approssimatore di un algoritmo di Approximated RL (ARL) può rappresentare. Con questa restrizione si cerca di sacrificare la possibilità di rappresentare qualsiasi funzione senza distorsioni, in cambio di una riduzione nella difficoltà dello stimare i parametri associati con il modello introdotto. Per esempio, l'uso di un modello lineare come quelli scelti in questa tesi, permette di ottenere buoni risultati finchè il numero di campioni rimane lineare rispetto al numero di parametri usati nell'approsimatore. Il problema della regressione lineare sarà il centro del Capitolo 3.

Per evitare di introdurre un errore irriducibile troppo grande, dovuto alla scarsa capacità di approssimazione del modello, una scelta comune è introdurre una grande quantità di parametri, nella speranza che con una ricca descrizione si riesca ad approssimare bene le funzioni necessarie. Questa esplosione nel numero di parametri reintroduce la curse of dimensionality, e rende impossibile trattare problemi con molti paramtri senza correre il rischio di avere un overfitting dell'approssimatore ai dati. Per risolvere questo problema una grande varietà di tecniche di regolarizzazione delle soluzioni sono state proposte nel campo della regressione lineare. Tra queste sono di particolare interesse quelle che riescono ad indurre nella soluzione una sparsità dei parametri, che corrisponde ad ottenere una soluzione con molti dei parametri uguali a zero. Porre un parametro a zero equivale ad escluderlo dal problema, e questo aiuta a ridurre il numero di dimensioni effettivamente coinvolte nella soluzione del problema. Tra i risultati conosciuti in letteratura, presenteremo nel Capitolo 3 dei bound per questi approssimatori lineari sparsi, che riescono a superare il limite del numero lineare di parametri e possono utilizzare un numero di parametri esponenziale rispetto al numero di campioni, a patto di avere poi una soluzione sparsa che sia ancora lineare nel numero di campioni.

Il punto di partenza di questa tesi è l'idea di spingere ancora oltre questo limite sfruttando un concetto di sparsità diverso, la sparsità di gruppo. Nei problemi Multi-Task di RL, un insieme di MDP che condividono similarità viene risolto in contemporanea, in modo da sfruttare le similarità per estrarre informazione e ottenere una soluzione finale migliore di quella che si sarebbe ottenuta considerando ogni Task separatamente. In particolare, l'assunzione di questa tesi è che i vari Task condividano una rappresentazione sparsa comune, e che quindi sia possibile ottenere una soluzione sparsa a livello di gruppi di variabili simili tra Task, e non semplicemente una soluzione sparsa per ogni singolo Task. Il risultato principale del Capitolo 4 è quindi la derivazione a partire da risultati nel campo della regressione con soluzioni sparse a gruppi, di bound teorici sulla performance del nuovo algoritmo, Sparse Fitted Q Iteration. In particolare considerare multipli Task allo stesso tempo permetterà di superare il numero esponenziale di parametri, a patto di mantenere una rappresentazione sparsa comune e aggiungere Task in numero lineare.

Uno dei requisiti più stringenti di questa formulazione è la presenza di una rappresentazione sparsa a livello di gruppo. Nel Capitolo 5 riporteremo un utile

algoritmo chiamato Multi-Task Feature Learning, in grado non solo di calcolare una regressione sparsa per problemi dotati della necessaria struttura, ma anche potenzialmente di trovare una tale rappresentazione partendo dai dati, sotto una serie di condizione meno restrittive. Nel corso del capitolo presenteremo alcune modifiche a questo algoritmo per renderlo più simile alla formulazione usata nei risultati teorici, e un modo efficiente per risolverlo.

Infine nel Capitolo 6 svolgeremo alcuni esperimenti su problemi artificiali per verificare se le condizioni necessarie per il buon funzionamento degli algoritmi proposti si verificano in simulazioni concrete.

Acknowledgments

Senza il supporto di tutta la mia famiglia la scrittura di questa tesi non sarebbe neanche cominciata. Il loro aiuto nella mia vita di tutti i giorni mi ha spinto fin qui, e per questo li ringrazio. Grazie ad Andrea, Matteo, Piero, Simone e a tutti gli altri amici che mi hanno dato una spinta in più.

Ringrazio il mio relatore Prof. Restelli per avermi aiutato in tutti questi anni ad entrare nel mondo della ricerca. Senza il suo tempo e il suo supporto non avrei mai cominciato il mio percorso. Ringrazio anche il Prof. Lazaric per avermi dato l'opportunità di lavorare in un gruppo di eccellenza. Con il suo aiuto e i suoi consigli questa esperienza mi ha dato la possibilità di crescere molto come ricercatore.

Daniele Calandriello
Milano
April 29, 2014

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Among the various successful Machine Learning fields, the Reinforcement Learning (RL) framework is one of the most promising. The reason for this success is partly caused by the main application of RL, complex control problems where a complete description of the dynamics is too hard or impossible. In these cases, RL techniques will use approximations and sampling to build a model of the system accurate enough to allow the derivation of optimal policies for its control. RL is not limited to Machine Learning application, but has been successfully introduced in operational research, economy and general AI. An RL approach [55] is particularly suited to describe all kinds of interaction between a decision-making agent and its environment. Through the use of a suitable feedback function, a large class of problems can be modelled and solved using Dynamic Programming (DP) and RL techniques. Among these modeling techniques, one of the most powerful and studied is the Markov Decision Process (MDP) [43]. An MDP describes the interaction of the agent with the environment as a sequence of states and actions, and the key assumption of the Markov property for the problem allows to efficiently find an optimal solution, or a near-optimal solution when it is impossible to correctly represent an optimal solution. In the first part of the thesis we will introduce a small number of key DP and RL techniques useful to understand the different approaches used to solve MDPs, and therefore be able to give some context for the introduction of the original algorithm introduced in this thesis.

We will start with DP methods, that allow to solve MDPs when full knowledge about the model is available, with both the transition probabilities from one state to the others, and the feedback function. This class of basic techniques introduce the fundamental distinction between Value Iteration algorithms [61], that seek to find a good approximation of the value of each state in order to extract a good policy for the task, and Policy Iteration algorithms [48], that instead try to find the optimal policy that the agent should follow by incremental improvements.

Satisfying DP's requirements in a practical setting is usually hard, because these techniques assume access to information that in practice is difficult to model exactly,

and also need to be able to correctly represent this information for all states, a problem hardly feasible when the number of state is large.

To relax the first of the two requirements, tabular RL techniques choose to build an optimal solution incrementally, using samples from the transitions and rewards instead of requiring access to the true model. As the number of samples grows, the estimate used by the various tabular RL algorithms becomes more and more accurate, and an optimal solution can be found without explicitly solving the whole planning problem.

Relaxing the second assumption is a harder but necessary improvement. Limiting RL to problems where the value functions need to be exactly represented for each state greatly reduces the potential application of these techniques. When the number of states becomes too large, computational costs and memory occupation of tabular based DP and RL techniques quickly grow and explode to unfeasible proportions.

Through the introduction of function approximation, RL algorithms can exceed this limitation and efficiently solve complex control problems with a large number of states, or even an infinite number of states. This is for example the case for all those problems whose state can only be efficiently represented with the use of a continuous state variable. After tabular techniques we will then introduce some of the most important Approximated Reinforcement Learning (ARL) algorithms [30]. Among them, the Fitted Value Iteration framework [16] will receive particular attention, and will form the basis for the development of Sparse Fitted Q Iteration, the original algorithm proposed in this thesis.

One of the main disadvantages connected to the use of function approximation, and therefore to Approximated RL methods, is the rapid growth of the number of samples necessary to successfully carry out the learning process as the dimensionality of the problem increases. This drawback, often referred to as the curse of dimensionality, severely limits the potential application of these methods to problems where collecting a large amount of samples is not too difficult. When it is impossible to collect more data, such as historical data that are in the past, or it is dangerous to collect samples, such as when using an expensive and fragile piece of equipment, it is desirable to use more sample efficient techniques. For example considering the contribution of all the samples at the same time can improve performances [31].

One method to avoid a complexity explosion is to restrict the class of functions that the approximator is capable of representing. With this restriction a bias due to the model assumptions is introduced that potentially makes it impossible for the approximator to find a good set of parameters to fit the target function. On the other hand the number of samples necessary to estimate the parameters is usually smaller for more biased estimators [9]. This dilemma, called the bias/variance trade-off, is central to this thesis. For example, the introduction of a linear model, as the one chosen in the original contribution of this thesis, allows the approximator to find a solution even with a small number of samples, as long as this number remains

linear w.r.t. the number of parameters [22]. The problem of linear regression will be introduced in detail, and several common techniques used to solve it will be considered.

To avoid introducing too much bias, due to a model with low representation capabilities, a common choice is to introduce a large number of parameters, with the hope that with enough parameters it will be possible to obtain a richer description and correctly represent the target function. If the problem is easy to model, it is possible to correctly choose the features to include among the parameters, such as higher order polynomial for physics problems or frequency counts for natural language processing. But in the most common RL setting, where the RL framework is used to find a good solution in an unknown environment, this is not possible, and it is likely that the introduction of a large number of parameters will cause a large amount of redundancy. Most of the parameters introduced will not be meaningful to the solution of the problem, but need to be included because the best subset is not known in advance. The explosion in the number of parameters effectively triggers again the curse of dimensionality, and makes it hard to solve problems with a large number of parameters without risking obtaining an approximation that is heavily overfitted to the data. To solve this problem many techniques for the regularization of the solutions have been proposed in the linear regression field. Among these, a particular interest is dedicated to those techniques ([57]) that find a solution that has a sparse parametric representation, or in other words a solution that contains many parameters exactly equal to zero.

When a parameter of the model is set to zero its contribution to the construction of the solution is effectively removed from the problem, and this helps to reduce the number of dimensions involved in the approximation. Among the known results in literature, we will present bounds for this class of sparse linear approximators ([5]), and we will see that they can surpass the linear limit in the maximum number of features. In particular, they allow the problem to have an exponential number of parameters compared to the number of samples, as long as the solution can still be represented using a small subset linear in the number of samples. The starting point of this thesis is the idea of pushing even further this limit, exploiting a different concept of sparsity called group sparsity.

In Multi-Task RL problems ([32]), multiple MDP problems are solved at the same time. The underlying assumption is that the various MDPs share some form of similarity. Solving them together allows to extract as much information as possible from each task, and share it with other tasks. The goal is to obtain a better final solution compared to the one we would obtain by considering each task separately. In particular the main assumption of this thesis is that the various tasks share a sparse common representation, and therefore it is possible to obtain a sparse solution at the level of groups of variables similar across tasks, and not simply a sparse solution for each task ([65]). The main result of this thesis is then the derivation, starting

from useful results in the field of group sparse regression ([35]), of a theoretical bound on the performance of the new proposed algorithm, Sparse Fitted Q Iteration. In particular considering multiple tasks at the same time will allow this technique to surpass the exponential number of unnecessary features that limited single-task sparse regression, as long as we maintain a group sparse representation and new tasks are added linearly.

The introduction of a group sparse assumption is a strong but necessary limit for the theoretical analysis. When the data do not present themselves as group sparse, this assumption does not hold, even if the underlying structure has some kind of intrinsic sparsity. To reduce this drawback, we considered the possibility of learning a sparse representation starting from the data when the original representation might be not suitable. To this end, we will use a popular feature learning algorithm, Multi Task Feature Learning [1], that can potentially improve performances if a group sparse representation is not present but can be derived. We propose a modification of this algorithm to make it more similar to the formulation used in the regression results, and provide an efficient way to solve the resulting optimization problem.

Finally, we will carry out some experiments on artificial settings, to confirm that the theoretical properties of the proposed algorithm are verified in actual simulations.

Mission The goal of this thesis is to investigate the consequences of introducing an assumption of group sparsity in an Approximated Reinforcement Learning problem. In particular we seek to improve the performances of ARL algorithms as the number of unnecessary features included in the model exceeds the exponential limit permitted by sparse, single-task techniques. Therefore we will formulate this problem in a multi-task setting and obtain improved theoretical guarantees on the performance of the algorithm.

Motivations In many realistic problems it is often unknown which underlying features are needed to build a good representation of the environment's state. If the right features are not included, the learning process will surely fail, and to avoid this a common choice is to introduce a large number of features, to increase the likelihood of correctly representing the necessary functions. Typical examples of these problems are localization, where selecting the coarseness of the localization grid is often a sensible problem, or physical problems with a large number of sensors, where only a small subset of measurements is really relevant to the problem. The introduction of a large number of parameters increases enormously the quantity of data that needs to be collected in order to successfully learn a good solution, due to a phenomenon called curse of dimensionality. Sparse approximations have proved themselves to be a good solution to high-dimensional problems. Several results exist for RL algorithms that exploit sparse solutions for the single-task setting. Multi-task approaches have proved themselves useful in the past, and we will investigate the introduction of a

multi-task, group sparse assumption, already popular in the regression setting, to RL.

Contributes The main contribution of this thesis is the introduction of a new Multi-Task RL algorithm called Sparse Fitted Q Iteration, its theoretical analysis, the derivation of an efficient implementation and an experimental validation of its performance. For the theoretical aspects, we provide a bound on the performance of the algorithm that, as long as suitable new tasks are introduced, is independent on the number of unnecessary features included in the model. This bound can be seen as a direct extension to the multi-task setting of similar results obtained for single-task RL algorithms. For the algorithmic contribution, we introduce a modification to a Multi-Task Feature Learning algorithm to closely match the assumptions of the bound. The introduction of feature learning to the SFQI algorithm can improve performances in scenarios that are not group sparse but have a suitable sparse representation. Finally, we provide experimental validation of the new technique.

1.1 Overview

We will now give a short overlook of the structure of this thesis. There are three main parts that compose the structure of this thesis. In the first part we introduce some initial results regarding RL techniques and regression. In particular we will introduce previous works that share similarities with this thesis, and the fundamental results that are preliminary to the derivation of the main results. In the second part we introduce SFQI and derive all the main contributions of this thesis. We will introduce other preliminary results recently proposed, and derive the main algorithmic definition of SFQI, its theoretical guarantees, and an extension with feature learning techniques. In the third part we will provide some experimental examples, and possible future directions for improvement.

In Chapter 2 we introduce MDP problems and preliminary results on their solution. This chapter will also define the notation used in the thesis regarding MDP problems and RL techniques. The important distinction between Policy Iteration techniques and Value Iteration techniques is introduced at the beginning of this chapter. In Section 2.4 we give a small introduction to function approximation, that will be later expanded in Chapter 3, and introduce several key algorithms for Approximated Reinforcement Learning. We conclude the section with a small overview of the Multi-Task approach to RL, and the improvements that it can provide.

In Chapter 3 we will introduce the linear regression setting and fully explain linear approximation. The beginning of the chapter focuses on the statistical justification for regression, and fully explain some details regarding function approximation that were implied in the previous chapter. After introducing regression as a statistical

problem we turn to the optimization approach to find the solutions to the various formulation of regression used in this thesis. The large family of linear Least Squares regression is considered, and some preliminary results on its potentials as an approximation tool are reported. As we mentioned, to obtain solutions where the Least Squares solution is not well-defined, we resort to the optimization tools and introduce several different regularization techniques, giving an intuitive interpretation of their underlying principles. This section introduces also the notation for the regression and optimization problems used in the main result. At the end of the section, several regularized RL algorithm will be introduced, both for PI and VI. For some of them we provide known theoretical results that we will later compared with SFQI.

In Chapter 4 the full description of SFQI is given with Algorithm 4.1. After having defined its basic formulation, we introduce additional recent results that will be used in the derivation of a bound for the performance of SFQI. We begin by introducing bounds on the performance of a generic Fitted Value Iteration, first proposed in [35]. These bounds depend on some mild assumption on the properties of the MDP. In particular a weak form of stability for the MDP transition is assumed, in order to guarantee that the error introduced by the approximations used at each step does not diverge. The other main assumption is a condition introduced in [35], that guarantees that the particular data provided to the function approximator is informative enough to reconstruct a good representation of the target function. The main result of the chapter is formulated and proven in Section 4.4. Exploiting the group sparseness of the solutions produced by the Group Lasso optimization problem we obtain a bound that scales favorably with the number of tasks. As we already mentioned this is a new, multi-task bound that has a more general application than previous results on single-task RL. A comparison with existing methods to show the main differences is included.

In Chapter 5 we consider an extension to the regression problem introduced in the previous chapters. In particular to relax the strict assumption of group sparsity we use the Multi-Task Feature Learning algorithm proposed in [1] to try to learn such a representation when the original features are not sparse. This approach should make the SFQI algorithm capable of dealing with a larger class of problems, not only those that are sparse but all problems that admit a sparse representation. We propose an original modification to the MTFL problem to formulate it as an exact extension of Group Lasso. With this modification the algorithm can follow the indications provided by theoretical results when used without the feature learning extension. Since introducing a new algorithm to exactly compute a Group Lasso solution is itself interesting, we compare the new MTFL-GL method to other optimization techniques.

In Chapter 6 the algorithms introduced are evaluated in an experimental setting to test whether the assumptions underlying the method hold, and in Chapter 7 we

make some final considerations and point at possible future developments.

Chapter 2

Markov Decision Processes: Introduction and Fundamental Results

This chapter introduces the formal definition of a Markov Decision Process (MDP), and the most important classes of algorithms that have been devised to solve such problems. As we will see, MDPs are suitable to represent a large class of control problems, and under certain assumptions they can be solved to an exact result, or to an approximately optimal solution when an optimal solution is impossible to obtain with the available data or would require computational costs not satisfiable in practice.

In particular, in Section 2.1 we will introduce the definition of the problem, as well as the first theoretical results on its solution. In Section 2.2 we will present some of the first Dynamic Programming algorithms proposed to find an exact, optimal solution, under the requirement of full access to all the information underlying the MDP, as well as a the possibility of precisely represent this model. In Section 2.3 we will instead introduce algorithms that do not need full information of the model, as long as an exact representation of it can be stored. Model-free tabular Reinforcement Learning techniques work with just sampled data from the MDP, and we will see that they still obtain performances asymptotically optimal. In Section 2.4 we will relax the exact representation requirement to allow MDPs to model problem with infinite states. Approximate Reinforcement Learning techniques find an approximately optimal solution, where the quality of the solution depends on the precision of our approximation. Under this class, we will introduce the Fitted Q Iteration algorithm in Section 2.4.4, that will be central to the rest of this thesis. Lastly, in Section 2.5, we will introduce the problem extension of solving not a single MDP, but a set of them. This multi-task approach can prove itself beneficial, for example when each single task has a scarce amount of available data for its solution.

2.1 Markov Decision Processes

A Markov Decision Process is a useful modeling tool for the description of the interaction that a decision-making agent has with its environment. The MDP describes this interaction in terms of the environment's state evolution in response to the actions taken by the agent. The agent will apply some decision rule, usually based on the state of the system, in order to reach its goal. This goal is represented under the form of a quantitative feedback that the environment gives to the agent in response to its actions. Therefore, solving the MDP means finding the optimal decision rule that will maximize this feedback.

This task is complicated by the fact that the actions that the agent takes influence the evolution of the state of the system, and therefore the optimal strategy will need to take into account the consequences of the present actions on the future of the system. Each of these actions represents the result of a decision that the agent made before executing the action. Decisions are made at time steps called decision epochs. Given a set $\mathcal{T}$ of decision epochs, we can consider two cases: $\mathcal{T}$ can be a continuous, infinite set of time instants, or it can be a, finite or infinite, set of distinct discrete instants. The continuous time setting will not be discussed in this thesis, since a discrete model of the time still allows to describe a large set of problems. For example the discrete nature of computer's operations makes all digitally controlled systems discrete. If our set $\mathcal{T}$ is finite, we can annotate the sequence of steps $\mathcal{T} = 0, \ldots, i, \ldots, N$, and we call this setting finite horizon. We can then extend the notation to the infinite horizon setting with $\mathcal{T} = 0, \ldots, i, \ldots, N, N \leq \infty$.

We will now formally define all the other properties, and the notation necessary to represent them, starting with discrete-space MDPs, to extend it later to the continuous-space case. For a more complete exposition of MDPs, and the techniques used to solve them, many good references are available such as [55, 12, 47, 4]

2.1.1 Discrete-Space Markov Decision Processes

We begin by introducing a distinction between continuous-state and discrete-state MDPs. In a discrete-state MDP, the number of possible states that the system can assume is finite. Therefore each state can be represented as a member s of a finite set S, with cardinality $|S|$. This is for example the case in simple problems such as small games like Tic-Tac-Toe or larger ones like chess, where the number of possible board states is finite. In a continuous-state MDP instead, at least one of the features that needs to be represented in order to identify the state can take a value in an infinite set. For example the distance to a goal on a line is a real number, and can take infinitely many values. In this case we use $x \in \mathcal{X}$ to represent the state. The same distinction can be done with the actions, but in this thesis we will concentrate on the case when the number of available actions to the agent is finite. Therefore we introduce an assumption on $\mathcal{A}$

Assumption 2.1.1. *$\mathcal{A}$ is a finite set.*

We will indicate with $A = |\mathcal{A}|$ the cardinality of the discrete action set.

A discrete-space, discrete-action MDP, also called countable MDP, is defined as a tuple $\mathcal{M} = (S, \mathcal{A}, P, R, \gamma)$. Over the course of the decision epochs, at each time step i, the system is represented by a state $s_i \in S$, and an action a_i is selected from the finite set of available actions $\mathcal{A}$. For the sake of simplicity we consider the case when all actions are available to the agent at each time step. A trajectory can now be defined as the collection of the states visited by the MDP at each time step, and the respective actions taken. Formally, a trajectory is defined as

$$h_i = (s_0, a_0, s_1, a_1, \ldots, s_{i-1}, a_{i-1}, s_i).$$

The space of all possible trajectories is defined as $\mathcal{H}_i : (S \times \mathcal{A})^{i-1} \times S$.

2.1.2 Transitions

When the agent at time i takes an action, the system transitions into a new state. We denote $M(\cdot)$ as a space of probability distributions over some space, and $\mu(\cdot)$ as a distribution from M. We also define $B(\cdot)$ as the space of bounded measurable functions over some space. Given a Borel set $b \subseteq S$, the transition probability kernel is a function $P : S \times \mathcal{A} \times \mathcal{B} \to [0, 1]$, with $\mathcal{B}$ the set of all possible b. For a given state s_i and an action a_i at time i,

$$\mathbb{P}(s_{i+1} \in b | s_i, a_i) = P(b | s_i, a_i).$$

Intuitively it represent the probability of transitioning into a new state s_{i+1} after taking action a_i in state s_i. In an equivalent formulation we can write $s_{i+1} \sim P(\cdot | s_i, a_i)$ The definition we have given of the transition probability kernel makes use of the main assumption of MDPs, which gives the name to the model. For any function associated with the MDP, that function is said to satisfy the Markov property if the influence the past can have on present results is limited, or in other words if the last state describes sufficiently the system to make previous records not useful. In the case of the transition kernel, the next state depends only on the current action and the current state, and not on the whole history h_i of the process up to that point. Formally,

$$\mathbb{P}(s_{i+1} \in b | h_i, a_i) = \mathbb{P}(s_{i+1} \in b | s_i, a_i).$$

2.1.3 Reward

The feedback that the environment gives to the agent at each time step can now be formally defined. The reward kernel function $R : S \times \mathcal{A} \to \mathbb{R}$ expresses quantitatively this feedback. Under normal conventions, a positive reward represent a positive

feedback, such as earning a sum or improving some performance index, while a negative reward represent a negative feedback, such as a loss or a failure. At each time step the agent collects a reward r_i after having executed an action in a state. The goal of an agent is to maximize some performance index based on the rewards that he collects along his trajectories. For example one might want to maximize the total sum of the rewards, or it's average over time. We will see in Section 2.1.5 several common choices. The reward in many problems is not a deterministic quantity, but instead can have random fluctuations. We take into account this when defining $R(s,a) = \mathbb{E}[r(s,a)]$. Moreover we make another assumption on the rewards,

Assumption 2.1.2. *Every reward kernel function R is bounded by some r_{max}.*

Without loss of generality we will assume that $r_{max} = 1$.

2.1.4 Policies

The main objective when solving an MDP is to derive a decision rule ζ that maximizes some performance index. A decision rule is intuitively defined as the method used to select the next action given our current trajectory. This can be written as $\zeta : \mathcal{H}_i \to M(\mathcal{A})$, where the action a_i is then selected according to this distribution. We can classify decision rules according to their properties: Markovian versus history-dependent and deterministic versus stochastic or randomized. A decision rule is Markovian if it follows the Markov property, that is if the action is chosen only depending on the current state and not on the whole trajectory. A deterministic decision rule is a degenerate stochastic decision rule, where the distribution from which to sample the action is all concentrated in a single action. A policy is a collection of decision rules that are followed at each time step

$$\pi = (\zeta_0, \zeta_1, \ldots, \zeta_i).$$

where each decision rule belongs to the same class. A policy itself can be classified as stationary or non-stationary, where the decision rule of a stationary policy does not change over time. Given this classification, we can see that the most general policy class is stochastic and history-dependent. All the possible sets are

$\Pi^{HR}, \Pi^{MR}, \Pi^{SR}$ history-dependent, Markovian and stationary randomized policies.

$\Pi^{HD}, \Pi^{MD}, \Pi^{SD}$ history-dependent, Markovian and stationary deterministic policies.

These classes inclusion graph is

$$\Pi^{SD} \subset \Pi^{SR} \subset \Pi^{MR} \subset \Pi^{HR}$$

$$\Pi^{SD} \subset \Pi^{MD} \subset \Pi^{MR} \subset \Pi^{HR}$$

$$\Pi^{SD} \subset \Pi^{MD} \subset \Pi^{HD} \subset \Pi^{HR}$$

In the rest of this thesis we will consider only stationary policies, referred as $\pi : S \times \mathcal{A} \to M(\mathcal{A})$. If the policy is stochastic we write $a_i \sim \pi(\cdot|x_i)$, while for a deterministic policy we can write $a_i = \pi(x_i)$.

2.1.5 Optimality

A definition can now be given for the performance indexes that guide the search for the optimal policy. As we said before, the goal of the agent is to maximize the accumulated rewards. A simple approach is to simply maximize the expected sum of the rewards collected along the trajectory obtained by starting from state s_0 and following policy π. Given a distribution over initial states μ, we want to find

$$\max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\substack{a_i \sim \pi \\ s_i \sim P}} \left[\sum_{i=0}^{N} r_i | s_0 \sim \mu \right],$$

where the expectation takes into consideration the randomness of the transitions, policy choices and the rewards themselves. This approach has a clear shortcoming, when the MDP has an infinite planning horizon, and the agent can collect reward forever, this quantity is not bounded. Another objective that is defined for the infinite-horizon case is the expected average reward objective

$$\max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\substack{a_i \sim \pi \\ s_i \sim P}} \left[\sum_{i=0}^{N} \frac{r_i}{N} | s_0 \sim \mu \right].$$

If the rewards are bounded, as $N \to \infty$ the whole quantity will remain bounded, under some mild assumptions on the properties of the MDP. This approach puts the same weight on rewards that are received in the short-term and on those that can be obtained only with a longer effort. If instead we want to value more short term rewards, such as in the case when prolonged trajectories risks entering a state from where it is impossible to recover, we can use the expected discounted reward

$$\max_{\pi \in \Pi} \mathop{\mathbb{E}}_{\substack{a_i \sim \pi \\ s_i \sim P}} \left[\sum_{i=0}^{N} \gamma^i r_i | s_0 \sim \mu \right].$$

Again this quantity is bounded even in the infinite-horizon setting, as long as the rewards are bounded, and the γ constant, called the discount factor, is strictly included between $(0, 1)$. The discount factor intuitively represent an indication of the probability that a trajectory will not terminate at each step. $\gamma = 0$ implies a single step trajectory, while $\gamma = 1$ an infinite one. For the rest of the thesis we will consider the expected discounted return under the assumption

Assumption 2.1.3. $0 < \gamma < 1$.

The performance of policy π w.r.t. some distribution μ is then defined as

$$J_\mu^\pi = \mathbb{E}\left[\sum_{i=0}^{N} \gamma^i r(s_i, a_i) : a_i \sim \pi(\cdot|s_i), s_{i+1} \sim P(\cdot|s_i, a_i), s_0 \sim \mu\right].$$

We add another assumption that will be assumed to hold for the rest of the thesis.

Assumption 2.1.4. *The reward kernel function $R(s, a)$ and transition kernel function $P(\cdot|s, a)$ do not vary over time.*

Under these assumptions, it is guaranteed that an optimal randomized stationary policy $\pi^* = \sup_{\pi \in \Pi^{SR}} J_\mu^\pi$ exists, and moreover there is also at least one optimal deterministic policy.

2.1.6 Value Functions and Bellman Equations

In the definition of the performance of a policy J_π, a starting distribution μ is used. Since the J_π captures the value of a policy, the value of a state is obtained by selecting $\mu(s) = 1$. This function is called Value function, and corresponds to the expected discounted reward the agent will obtain starting from state s and following policy π. It is defined as

$$V^\pi(s) = \mathbb{E}\left[\sum_{i=0}^{N} \gamma^i r(s_i, a_i) : a_i \sim \pi(\cdot|s_i), s_{i+1} \sim P(\cdot|s_i, a_i), s_0 = s\right].$$

Separating the first argument from the rest, we can obtain the recursive definition of the Value function

$$\begin{aligned} V^\pi(s) =& \mathbb{E}\left[r(s_0, a_0) + \sum_{i=1}^{N} \gamma^i r(s_i, a_i) : a_i \sim \pi(\cdot|s_i), s_{i+1} \sim P(\cdot|s_i, a_i), s_0 = s\right] \\ =& \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)}\left[R(s, a)\right] + \mathop{\mathbb{E}}_{\substack{s' \sim P(\cdot|s,a), \\ a \sim \pi(\cdot|s)}}\left[\gamma V^\pi(s')\right]. \end{aligned}$$

This recursive equation is called Bellman equation. It can be expressed as an affine linear operator T^π, called the Bellman operator,

$$(T^\pi V)(s) = \sum_{a \in \mathcal{A}} \pi(s, a)\left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V(s')\right]. \tag{2.1.1}$$

It can be shown that the Bellman operator is a contraction, that is,

$$\left\|T^\pi V - T^\pi V'\right\|_\infty \leq \gamma \left\|V - V'\right\|_\infty,$$

and therefore with the application of the Banach fixed point theorem, it is possible to prove that V^π is the unique fixed point of the Bellman operator [25]. The Value

function of the optimal policy π^* is called the optimal Value function, and is denoted as $V^* = V^{\pi^*}$. It can be shown that this Value function is the fixed point of another operator, called optimal Bellman operator,

$$(T^*V)(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]. \tag{2.1.2}$$

This operator is nonlinear, and corresponds to taking the greedy action w.r.t. to the current values of the Value function. A useful property of the optimal Value function, that follows directly from its definition is $V^*(s) \geq V^\pi(s)$ for all states s and policies π. Another useful representation of the value of a policy is the so called Q-function, or Action-Value function, which represents the expected discounted reward the agent will gain by selecting action a in state s, and then following policy π. It is defined as

$$Q^\pi(s,a) = \mathbb{E}\left[r(s,a)\right] + \mathop{\mathbb{E}}_{\substack{s' \sim P(\cdot|s,a), \\ a' \sim \pi(\cdot|s')}} \left[\gamma Q(s',a')\right].$$

Its corresponding Bellman operator is

$$(T^\pi Q)(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \left[\sum_{a'} \pi(s',a') Q(s',a') \right], \tag{2.1.3}$$

and the optimal Bellman operator

$$(T^* Q)(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \left[\max_{a'} Q(s',a') \right]. \tag{2.1.4}$$

The Q-function is connected to the Value function by the relationship $\sum_{a \in \mathcal{A}} \pi(s,a) Q^\pi(s,a) = V^\pi(s)$, and from this relationship we can easily see that any policy that satisfies $\sum_{a \in \mathcal{A}} \pi(s,a) Q^*(s,a) = V^*(s)$ is guaranteed to be optimal. This property allows us to extract an optimal policy from Q^* easily, simply taking the action that maximizes Q^* in each state. We call a policy $\pi^+(s) = \arg\max_{a \in \mathcal{A}} Q(s,a)$ a greedy policy w.r.t. to Q, and therefore the optimal policy is greedy w.r.t. to Q^*.

2.1.7 Continuous Markov Decision Processes

Equivalent results to the discrete Bellman operators can be derived for the continuous-space MDPs. Integrals take the place of summations over states, and the following

definitions hold

$$(T^{\pi}V)(x) = \sum_{a\in\mathcal{A}} \pi(x,a)\left[R(x,a) + \gamma \int_{x'\in\mathcal{X}} P(x'|x,a)V(x')\right], \tag{2.1.5}$$

$$(T^{*}V)(s) = \max_{a\in\mathcal{A}}\left[R(x,a) + \gamma \int_{x'\in\mathcal{X}} P(x'|x,a)V(x')\right], \tag{2.1.6}$$

$$(T^{\pi}Q)(x,a) = R(x,a) + \gamma \int_{x'\in S} P(x'|x,a)\left[\sum_{a'} \pi(x',a')Q(x',a')\right], \tag{2.1.7}$$

$$(T^{*}Q)(x,a) = R(x,a) + \gamma \int_{x'\in S} P(x'|x,a)\left[\max_{a'} Q(x',a')\right]. \tag{2.1.8}$$

Because the MDP can assume an infinite number of states, the exact representation at each state that could be stored for the Value function of countable MDPs cannot be physically contained in the finite memory of a computer. As we will see in Section 2.4, algorithms try to overcome this limitation by storing approximations of these function as close as possible to the true values.

2.2 Dynamic Programming

The first class of algorithms that we will introduce make use of the knowledge of the entire MDP model. This means that they assume access to the true definition of the R and P functions. Exploiting this knowledge with a countable MDP allows these techniques to find the guaranteed optimal policy. Depending on the method used to compute the policy, these methods can be classified into Policy Iteration and Value Iteration methods. We will now give an example of a method belonging to each category using the appropriate Bellman operators.

2.2.1 Model-Based Policy Iteration

The first class of Dynamic Programming techniques we introduce is called Policy Iteration, because it iterates through a sequence of policies in order to find the optimal policy π^*. Each step is composed of two phases, called policy evaluation and policy improvement.

The policy evaluation steps consist in computing an evaluation of the latest policy in the sequence, under the form of V^{π_k} or Q^{π_k}. After the Value function of the current policy is computed, it is used to select a new, improved policy. We will consider Policy Iteration with the use of Q-functions for the policy evaluation step, but with the use of the model, passing from a Q function to a V function is straightforward. Policy iterations then alternate between computing a new Q_k, then an improved π_{k+1}, until the policy cannot improve anymore and we converged to the optimal policy π^*.

$$\pi_0 \to Q^{\pi_0} \to \pi_1 \to Q^{\pi_1} \to \cdots \to \pi_K \to Q^{\pi_K} \to \pi^* \to Q^*$$

Algorithm 2.1 Policy Iteration with Q-function

input: P, R, γ
output: π^*, Q^*
  Initialize π_0
  do
    Policy Evaluation: Repeat $Q_k \leftarrow T^{\pi_k} Q_k$ until convergence.
    $k \leftarrow k+1$
    for all $s \in S$ **do**
      Policy Improvement: $\pi_k(s,a) = 1 : a \in \arg\max_{a \in \mathcal{A}} Q_{k-1}(s,a)$
    end for
  while $\pi_k \neq \pi_{k-1}$

Algorithm 2.1 details these steps of PI. Concretely, the policy improvement step consists in selecting the greedy policy w.r.t. the last Q-function computed. This greedy policy is not guaranteed to be unique, but ties can be broken arbitrarily. Improvement of the value of the policy is guaranteed by the policy improvement theorem

$$\text{if } \forall\, s \in S \underset{a \sim \pi^{k+1}(\cdot|s)}{\mathbb{E}}[Q(s,a)] \geq V^{\pi_k}(s) \text{ then } \forall\, s \in S \;\; V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s).$$

Since the greedy policy satisfies the hypothesis, the V function, and therefore the Q-function, will be monotonically increasing. When $\pi_k = \pi_{k+1}$ we can stop iterating, because the new policy will induce the same Q^{π_k} function, which in turn will induce the same policy. More steps are not needed anyway, because when we obtain the same policy twice, it means the policy cannot be improved anymore and we reached π^*.

Regarding the policy evaluation step, exploiting the knowledge of the MDP, we can simply repeatedly apply the appropriate Bellman operator until convergence. That is, given an initial Q_0 we repeat

$$Q_{i+1}(s,a) = (T^{\pi_k} Q_i)(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \left[\sum_{a'} \pi_k(s',a') Q_i(s',a') \right]$$

for all states and actions, until $Q_{i+1} = Q_i$. Due to the contraction properties we mentioned in Section 2.1.6, we will converge to Q^{π_k}.

2.2.2 Model-Based Value Iteration

As we saw in the previous section, Policy Iteration exploits the Bellman operator to find the Q^π of a policy, and then uses the actions that maximize this function to build a new policy.

A different approach to solve an MDP, when again full knowledge of the model is provided, is to try to find the optimal Q^* directly, without computing intermediate

Algorithm 2.2 Q-iteration

input: P, R, γ
output: Q^*
  Initialize Q_0
  do
    $k \leftarrow k + 1$
    for all $s \in S, a \in \mathcal{A}$ **do**
      $Q_k(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \left[\max_{a'} Q_{k-1}(s', a')\right]$
    end for
  while $Q_k \neq Q_{k-1}$

policies. That is, instead of applying the Bellman operator, we iteratively apply the optimal Bellman operator to some initial Q-function, until this sequence converges to Q^*. This simple method, called Q-iteration, is reported in Algorithm 2.2.

From the optimal Q^* a policy can then be easily extracted. An equivalent technique can be used for V function since the use of the model allows us to connect the two functions, and the only difference is that we will iterate T^*V from Equation 2.1.2 instead of T^*Q from Equation 2.1.8. The use of a Q-function simply makes extracting the final policy more intuitive.

The termination criteria given in Algorithm 2.2 is satisfiable only asymptotically, and difficult to use in practice. A more practical approach is to terminate when $|Q_k - Q_{k-1}|$ falls under a tolerance threshold or after a set number of iterations. Several ways to set the number of iterations can be considered, but a known result regarding VI ([16]) gives us a method to select a sensible number in terms of how much error we can commit in the estimation. This error is bounded by

$$\|V_k - V^*\|_\infty \leq 2\frac{\gamma^k r_{max}}{(1-\gamma)^2}, \tag{2.2.1}$$

and we can select an appropriate k based on it.

Policy Iteration and Value Iteration are both Dynamic Programming techniques that converge to the optimum. One clear difference is in their computational cost. At each step, VI need to iterate over $|S||\mathcal{A}|$ possible state actions, and for each of these to make a summation over $|S|$ states, computing a maximum over $|\mathcal{A}|$ actions at each step. The overall complexity of a single iteration is therefore $\mathcal{O}(|S|^2|\mathcal{A}|^2)$. Policy iteration instead has a fixed cost of $|S||\mathcal{A}|$ for the policy improvement step, which is shadowed by the cost of a policy evaluation step. The only difference between a policy evaluation step and a Q-iteration step is that we do not need to iterate over the actions to find the maximum, but simply select the a action of the deterministic policy. PI iteration therefore has a computational complexity of $\mathcal{O}(L|S|^2|\mathcal{A}|)$ where L is the number of times the Bellman operator needs to be applied to obtain convergence of the Q^π estimate. Because the Bellman operator is a contraction, a bound like

Equation (2.2.1) can be derived, and it is interesting to compare this computational cost with the $\mathcal{O}(|S|^3|\mathcal{A}|^3)$ cost of directly solving the Bellman linear equation system and compute Q^π single-shot. For a more detailed explanation refer to [9].

Since neither PI nor VI has a clear advantage computationally, both are viable solutions to solve an MDP. Their main drawback is the requirement of the full knowledge of the underlying problem. We will see how RL techniques try to solve this problem by finding an approximate solution, that will slowly converge to the true solution given enough data.

2.3 Reinforcement Learning With Tabular Representation

The techniques we will present in this section are model-free, tabular based RL algorithms. Model-free means that they do not depend on the knowledge of the transition and reward model to build the estimates needed to find optimal policies and value functions. Instead these quantities are slowly approximated making use of samples obtained directly from the MDP. Asymptotically in the number of samples and iterations, these methods will converge to an optimal solution. The optimal solution is obtained because we are still considering a finite state space, and we are using a tabular representation to exactly keep track of the values of each function and policy, for each state-action pair.

When the number of states becomes too large, or even infinite, this tabular approximation cannot be used anymore, and we will introduce in Section 2.4 algorithms that instead use function approximators to build estimates on continuous values.

In this chapter, the Policy Iteration and Value Iteration techniques that we introduced in the previous section are extended using the Temporal Difference approach. In this approach an initial estimate of a function is iteratively refined by updates that make use of the previous estimate and the new data. This quantity is called temporal difference, and can be interpreted as a gradient step, or as an estimate of a mean value. More specifically, if we want to estimate a Q^π function we will compute for each state and action

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[(T^\pi Q_i)(s,a) - Q_i(s,a) \right].$$

The term between brackets is the temporal difference. Written in this form it can be interpreted as a finite difference between Q_i and the desired function, the fixed point of $T^\pi Q$. If instead we want to see it as an average between the current estimate and the objective we write it as

$$Q_{i+1}(s,a) = (1-\alpha)Q_i(s,a) + \alpha(T^\pi Q_i)(s,a).$$

Since RL algorithms do not have access to the true model, they rely on the use of samples to estimate the Bellman operator, and substitute the $(T^\pi Q_i)(s,a)$ term

with $r_i + \gamma Q_i(s', a')$, where each algorithm will provide a method to choose s', a'. This approach is similar to Monte Carlo methods, in the sense that it uses rewards from the trajectories to estimate its values. Using an estimate of Q_i to build a new improved estimate is instead more similar to Dynamic Programming. In general we say that an algorithm bootstraps if it uses the current estimation to build better estimates.

TD methods can be extended to propagate rewards more into the past with a technique called eligibility traces. This makes TD more similar to Monte-Carlo, and is called TD(λ). When the Temporal Difference only updates the state-action that collected the immediate reward, and therefore propagates rewards only for one step, the method is called $TD(0)$.

We can also classify algorithms as on-policy or off-policy. An on-policy algorithm evaluates and improves the same policy that it will use for data collection. That is, the latest policy computed by the algorithm is used to collect additional data, and then the same policy is modified using the data collected. Another option is to use a separate policy to collect data, and to perform all the computation necessary to find the optimal policy and value function on a separate set of variables. This approach is called off-policy.

A similar classification, based on how much data is collected before processing, is on-line versus off-line. An on-line algorithm process data sequentially as they are collected. An off-line, or batch, algorithm collects all its data before performing its computation without further interaction with the system.

We will now introduce two basic TD algorithms that extend PI and VI, SARSA, an on-line, on-policy Policy Iteration algorithm, and Q-learning, an on-line, off-policy Value Iteration algorithm.

2.3.1 Model-Free Policy Iteration

The first algorithm we will introduce is called SARSA [48]. The name derives from the notation of the samples necessary for its execution: $(s_i, a_i, r_i, s_{i+1}, a_{i+1})$.

First we need to introduce the definition of an ϵ-greedy policy. An ϵ-greedy policy, w.r.t. some Q-function, is a policy that chooses the greedy action in state s with probability $(1 - \epsilon)$, and execute a random exploratory action otherwise. In the algorithm we indicate $a \sim Greedy(Q, s, \epsilon)$ to indicate an action chosen using this policy.

The method is presented in Algorithm 2.3. At each time step SARSA(0) executes a previously decided action, collect its reward and observes the next state. It then chooses an ϵ-greedy action, and uses it to compute a single temporal difference step, as well as storing it to execute it later. One strong difference between SARSA and PI is that instead of waiting for a full policy evaluation, in SARSA a single state-action value is updated with a single sample. Because the ϵ-greedy policy makes its choices based on the Q-function, this update also changes the policy that will be used to

Algorithm 2.3 SARSA(0)

input: $\gamma, \alpha_i, \epsilon_i, s_0$
output: Q^*
  Initialize Q_0
  $i \leftarrow 0$
  $a_0 \sim Greedy(Q_0, s_0, \epsilon_0)$
  do
    Execute a_i, obtain r_i, s_{i+1}
    $a_{i+1} \sim Greedy(Q_i, s_{i+1}, \epsilon_{i+1})$
    $Q_{i+1}(s_i, a_i) \leftarrow Q_i(s_i, a_i) + \alpha_i \left[r_i + \gamma Q_i(s_{i+1}, a_{i+1}) - Q_i(s_i, a_i)\right]$
    $i \leftarrow i + 1$
  while $i \leq N$

select future actions, and this corresponds to an implicit policy improvement step. This approach of only executing a few updates on the Q-function before doing a policy improvement, instead of executing the full policy evaluation step, is referred to as Generalized Policy Iteration. GPI has PI at one end of the spectrum, where the policy evaluation step is carried out completely, and algorithms that only execute a single iteration of policy evaluation like SARSA at the other.

Because in the end SARSA needs to obtain the greedy policy w.r.t. Q^*, letting the ϵ term decay over time makes the ϵ-greedy policy become more and more greedy. Conversely, a necessary assumption for convergence to the optimum is that each state-action pair will be visited asymptotically infinitely many times. This trade-off between making greedy choices to maximize short term reward, and make suboptimal choices in order to gain new information on unknown parts of the MDP is called the exploitation/exploration trade-off. The problem of efficiently exploring the agent's possible options, without suffering too much regret caused by the suboptimal choices is an open question. Exploration parameters, such as the value of ϵ need to be carefully selected, and are often chosen using external knowledge or heuristics. A simple choice is $\epsilon = \frac{1}{i}$, where in the beginning we will explore more, while later we will focus on exploiting. Another possibility is to avoid forced exploration altogether, and instead start with a Q-function that is a large upper bound of the true value. This principle, called "optimism in face of uncertainty"[3], motivates the agent to try often unknown state-actions, without an explicit need for forced exploration.

Another necessary constraint is on the α_i values

$$\sum_{i=0}^{\infty} \alpha_i^2 < \infty, \quad \sum_{i=0}^{\infty} \alpha_i \to \infty,$$

and is again satisfied by $\alpha_i = \frac{1}{i}$.

A drawback of SARSA is its sample inefficiency. Transition are used only to update a single state-action, and then thrown away. In order to use samples to

Algorithm 2.4 SARSA(λ)

input: $\gamma, \alpha_i, \epsilon_i, s_0, \lambda$
output: Q^*
  Initialize Q_0
  for all $s \in S, a \in \mathcal{A}$ **do**
    $e(s_i, a_i) \leftarrow 0$
  end for
  $i \leftarrow 0$
  $a_0 \sim Greedy(Q_0, s_0, \epsilon_0)$
  do
    Execute a_i, obtain r_i, s_{i+1}
    $e(s_i, a_i) \leftarrow e(s_i, a_i) + 1$
    $a_{i+1} \sim Greedy(Q_i, s_{i+1}, \epsilon_{i+1})$
    $\delta \leftarrow [r_i + \gamma Q_i(s_{i+1}, a_{i+1}) - Q_i(s_i, a_i)]$
    for all $s \in S, a \in \mathcal{A}$ **do**
      $Q_{i+1}(s, a) \leftarrow Q_i(s, a) + \alpha_i \delta e(s, a)$
      $e(s, a) \leftarrow \gamma \lambda e(s, a)$
    end for
    $i \leftarrow i + 1$
  while $i \leq N$

update more than one state-action, eligibility traces seek to distribute the reward to all the state-actions that participated in the trajectory that collected the reward. Concretely a decaying factor $e(s, a)$ is introduced, and each time the trajectory passes through a state-action, its factor is increased, or sometimes simply set to a maximum. At each time step, all the factors decay, therefore only states that were recently visited will be marked as contributors to the collection of the reward. This results in SARSA(λ), reported in Algorithm 2.4

2.3.2 Model-Free Value Iteration

We will introduce in this section a TD extension of VI, called Q-learning [61]. Unlike SARSA, this is an off-policy method, because the temporal difference will not use $T^\pi Q$ as its target, but rather $T^* Q$, as VI did in Dynamic Programming. Unlike DP, we will not use the model, but transitions from the MDP. The details are given in Algorithm 2.5. An interesting property of off-policy algorithms is that the exploration can be carried out efficiently with an exploratory policy, while learning the optimal policy on the samples collected. Therefore the exploratory value ϵ is not constrained to decrease over time as in SARSA. Instead, we now have problems introducing eligibility traces, because we cannot assign merit to a state-action pair that is the result of an exploratory policy that does not correspond to the value we are

Algorithm 2.5 Q-learning

input: $\gamma, \alpha_i, \epsilon_i, s_0$
output: Q^*
  Initialize Q_0
  $i \leftarrow 0$
  do
    $a_i \sim Greedy(Q_i, s_i, \epsilon_i)$
    Execute a_i, obtain r_i, s_{i+1}
    $Q_{i+1}(s_i, a_i) \leftarrow Q_i(s_i, a_i) + \alpha_i \left[r_i + \gamma \max_{a' \in \mathcal{A}} Q_i(s_{i+1}, a') - Q_i(s_i, a_i)\right]$
    $i \leftarrow i + 1$
  while $i \leq N$

estimating. One approach, although inefficiently, solves this problem by truncating all eligibility traces every time an exploratory move is performed. This results in Watkins' $Q(\lambda)$.

All the algorithms presented this far still rely on iterating over all possible states of the MDP, and are guaranteed to converge because the Q-function can be exactly represented in a table for a finite number of states. When the number of states is too high, these methods suffer an explosion in time execution and quickly become unfeasible. Therefore even if a continuous variable can be discretized, a discretization too fine can give rise to an enormous number of states that are impossible to manage. In order to solve this challenge, RL techniques that use an additional approximation to represent the state were devised. In the next section we will introduce some of these techniques and their properties.

2.4 Reinforcement Learning With Function Approximators

All the methods we introduced so far explicitly depends on the cardinality of the state space. For policy evaluation and improvement steps, their pseudocode includes an iteration over all states. When the number of states becomes too large, for example as Computer Go's 10^{170} states, such iterations become too long for real-world applications. Even when the states are not iterated over, the tabular representation we have used so far is a much more restrictive constraint in modern computer than the computation time, because storing information for all the states in memory is often impossible. It is important to keep in mind in all these discussions the consideration we made in the beginning, that many real world problems cannot be represented with a finite number of states at all. For example, all problems that are described by at least one continuous variable cannot be solved with DP or tabular RL. To solve this larger class of MDPs, a trade-off between the quality of the solution,

and the representation and computation complexity of the problem must be made.

We will now present extensions to Policy Iteration and Value Iteration that rely on approximating the Value and Action-Value functions, rather than exactly representing them in a tabular fashion. This poses a new problem: these methods needs to learn some value function from the data, but also to learn how to represent that function using a member of some restricted class of functions $\mathcal{F}$. We will introduce a classification of the possible $\mathcal{F}$ as parametric, non-parametric, linear and non-linear approximators. How well the methods will perform often depends on some measure of how well the chosen functional space can approximate the true value function, with results varying from failing to converge to any solution, to convergence to some suboptimal alternative, or even to a policy that correctly takes the optimal action.

In Section 2.4.1 we will introduce some preliminary classifications and results on function approximation. In Section 2.4.2 and Section 2.4.3 we will introduce the extension of PI and VI to continuous state space, as well as new techniques based on TD. A particular attention is given to Fitted Q Iteration in Section 2.4.4, due to the fact that this algorithm is the basis for Sparse Fitted Q Iteration, the main original contribution of this thesis.

2.4.1 Function Approximators

The problem of function approximation can be formulated, with a basic approach, as the search over some space of functions $\mathcal{F}$, of a function F capable of closely representing some target function that is of interest, but that cannot be fully represented. Another problem that arises in the stochastic setting is that we might have incomplete knowledge of the function's shape, and only have some noisy samples to test the quality of our approximation. This second case, usually referred to as regression, changes its goal to finding a function that closely represents the mean of the target function, to take into account the lost information due to noise and the limited number of samples.

In both cases we are interested in finding a compact representation of the function, using some kind of loss function to quantify the fidelity of the representation, where the loss function L is computed according to some distribution that represent our interest in representing the function closely over different parts of its domain. If we consider the more realistic case when we only have access to a dataset $\mathcal{D} = (y_i, x_i)$ of samples of the target function, regression or function approximation can be stated as

$$F = \min_{F' \in \mathcal{F}} L(F', \mathcal{D}).$$

The highest level distinction we can make when deciding which function space $\mathcal{F}$ consider for our problem, is parametric vs non-parametric. In parametric function spaces, the member functions of the set are described as a combination of a small

number of parameters. The number of parameters and how they combine is set in advance, and will not depend on the actual data from $\mathcal{D}$. A non-parametric function approximator will instead modify the space of functions it can represent using information from the dataset, and therefore it is not known in advance how it will represent the target function. Both of these methods have advantages and disadvantages, and there is not a clear dominance of one over the other.

2.4.1.1 Parametric Approximation

We will begin by describing parametric approximation, where the number and use of each parameter is decided in advance. Given a vector of such parameters $w \in \mathbb{R}^d$, and a target function we want to approximate, for example a Q-function $Q \in \mathcal{Q}$, then $F \in \mathcal{F}$ is a mapping $F : \mathbb{R}^d \to \mathcal{Q}$. The expressiveness of the possible functions belonging to $\mathcal{F}$ depends on the parameters, and those are set in advance, before any data from the dataset is ever seen. This does not mean that parametric approximation suffers some kind of limitation. In [11], it is proven that given enough parameters, one of the most traditional parametric approximators, a feed-forward neural network, can represent any continuous function on a bounded set up to any desirable precision. In general anyway the compact representation will not allow us to fully represent any function in $\mathcal{Q}$, because a form of approximation is always present and contributing to errors.

The mapping from the parameters to the function needs not to be linear. Neural networks [10, 40], that we already mentioned, are highly non-linear. But the class of approximator that use a linear combination of the parameters as a mapping has received a particular attention due to their simplicity, that helps derive theoretical results and is favorable to direct computation and optimization.

A linear parametric approximator defines its mapping as the sum of the parameters, also called weights, w_i multiplied by a set of functions of the input $\phi_i(x)$ where $x \in \mathcal{X}$ is the independent variable over which the represented function is defined.

$$F(x) = \sum_{i=1}^{d} \phi_i(x) w_i.$$

In a more compact notation we can define the vector

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \cdots \\ \phi_d(x) \end{bmatrix}.$$

Each function ϕ_i is called Basis Function, or feature, and evaluating $F(x)$ corresponds to preparing the features ϕ and computing the inner product $\phi(x)^\mathsf{T} w$. The criteria

for selecting the best function approximator given a dataset $\{(y_i, x_i)\}_{i=1}^n$ becomes

$$\widehat{w} = \arg\min_w \sum_{i=1}^{n} L(y_i, \phi(x_i)^\mathsf{T} w).$$

A commonly used and intuitive feature is the Radial Basis Function. An RBF $\phi_i(x)$ is defined as a function of the distance between the state x and a center c that identifies the RBF. A popular choice of RBF, that we will also use in the experiments, is the Gaussian RBF

$$\phi_i(x) = \exp\left(-(x-c)^\mathsf{T} D^{-1}(x-c)\right).$$

Here D is a semi-definite positive matrix that scales and rotates the norm of $(x-c)$. RBF can be seen as a continuous version of tile coding, another popular choice of BF ([62]). Tile coding is a complete partition of the space into tiles, where the whole set of tiles is called tiling. For each tiling, only one tile is active at a time, whereas more than one GRBF is usually active due to the continuous property of the exponential. In practice, multiple overlapping tiling are often used to improve precision. Other popular choices of BF are state aggregation ([52]) and Kuhn triangulation ([41]).

2.4.1.2 Non-Parametric Approximation

Non-parametric approximators, despite their name, still make use of parameters. The main difference with parametric approximators is that the number of parameters, as well as their role in the approximation, changes to adapt to the data available in the dataset.

One of the typical example of non-parametric approximation is Kernel-based approximation. Continuing with the example of approximating a Q-function, in a Kernel model we introduce a set of functions over two state-action samples $(s,a),(s',a')$

$$K\left((x,a),(x',a')\right) : (\mathcal{X}\times\mathcal{A})\times(\mathcal{X}\times\mathcal{A}) \to \mathbb{R},$$

called Kernels. The functions must satisfy several assumptions, refer for example to [53], and if these assumptions are met then Mercer's Theorem applies and the Kernel can be interpreted as an inner product in some high-dimensional or even infinite dimensional space [38]. Therefore applying a Kernel to a couple of samples is equivalent to applying a dot product to some mapped samples in a much richer feature space. The points are said to be embedded in this new feature space, usually a Reproducing Kernel Hilbert Space, and non-linear relationships among the points in the original space might be represented through simple linear dot product in the mapped space. One straightforward application of Kernels, for which the dataset is necessary, is the construction of BFs based on the samples $\{x_i\}_{i=1}^n$, and then use these BFs in a linear combination to build a function approximator

$$F(x',a') = Kw = \sum_{i=1}^{n} K(x',a',x_i,a_i)w_i.$$

As we can see, the number of parameters scales with the number of samples n, and the BF themselves adapt to the samples present in $\mathcal{D}$. Therefore the shape of the approximation space $\mathcal{F}$ will change and adapt to the samples. This allows for a greater flexibility, at the expense of an increased complexity of the analysis of these techniques. Kernel approximations have been successfully used in a number of applications, most notably Support Vector Machines [53, 27], and Gaussian Processes [45]. Other non-parametric approximators are regression trees [7], used for example in the original Fitted Value Iteration paper [16].

2.4.1.3 Remarks on Parametric vs. Non-Parametric Approximation

Both parametric and non-parametric approximators have strong points. One of the drawbacks of parametric approximators, and especially of linear parametric approximators, is their lack of flexibility. In the case of linear approximation, once a set of BF has been chosen, either the target function lies in the plane defined by the combination of the BF, or no amount of information on the target function will allow the approximator to accurately represent it. This problem often pushes researchers in the direction of highly increasing the number of BF, in order to increase the likelihood that the target function will be representable.

The increase in complexity introduces two challenges. Because it is not known a priori which BF are really needed to represent the function, the function approximator has a large number of parameters not needed to correctly represent the target function. This can lead to overparametrization of the function, which in turn poses computational and algorithmic problems. Another important problem is that the complexity of the F functions increases with the number of parameters, and several associated numbers, such as covering number and sample complexity, increase quickly with it. The necessity of managing this high number of superfluous parameters is the motivation of this thesis. The introduction of regularization can help linear approximator cope with a large number of useless BF, and allows for the simple analysis of linear approximation to be used even when a large feature space is needed.

On the other hand non-parametric approximation can represent complex functions using only a small number of parameters thanks to the adaptation capabilities provided by Kernels. But this increase in flexibility is counterbalanced by a much harder theoretical analysis, that leads to far rarer convergence results for non-parametric algorithms. Another problem is that the size and complexity of the problem increases with the number of samples. This leads to computational problems when the datasets are large, that can be partially solved by carefully selecting only a subset of the samples as members of Kernel functions [15].

2.4.2 Continuous-State Policy Iteration

Several algorithms for Policy Iteration with function approximation have been proposed. We will present one of the most representative, Least Square Policy Iteration [30].

Policy Iteration is composed of a policy evaluation step, followed by a policy improvement step. When the number of actions is finite, policy improvement with access to a Q-function is immediate, and reduces to a simple iteration over the actions looking for the best action. We will then initially describe LSPI with use of Q-functions in discrete action settings, and comment later on possible extensions.

The main difference with a standard PI problem is that the policy evaluation step is now based on approximations of functions. This means that if before we used to apply the Bellman operator to an exact representation Q_i to obtain a new exact representation Q_{i+1} we now have to take into consideration that the resulting function obtained with the application of the Bellman operator might not be representable with functions from $\mathcal{F}$. Therefore we need an additional step, where after computing the new target function with the Bellman operator, a suitable approximation $\widehat{Q}$ needs to be found to closely represent it. Formally, we define $\widehat{Q}_i \in \mathcal{F}$, and at each step need to find

$$\widehat{Q}_{i+1} = \arg\min_{\widehat{Q} \in \mathcal{F}} \|\widehat{Q} - T^\pi \widehat{Q}_i\|^2.$$

The $\|\widehat{Q} - T^\pi \widehat{Q}_i\|^2$ quantity is called the Bellman error. A main drawback of following the Bellman error as an indicator to choose the best representation is that it is a biased penalty of the error committed when estimating the Bellman operator through sampling. In particular if we introduce an approximated Bellman operator $\widehat{T}^\pi$

$$\mathbb{E}\left[\left(\widehat{Q}(s,a) - \widehat{T}^\pi \widehat{Q}_i(s,a)\right)^2\right] = \left(\widehat{Q}(s,a) - \widehat{T}^\pi \widehat{Q}_i(s,a)\right)^2 + Var\left(\widehat{T}^\pi \widehat{Q}_i(s,a)\right).$$

The Bellman error prefers functions with low variance. A simple way to correct this problem is to introduce a projection step before the estimation. Given a projection $\Pi_{\mathcal{F}}$, the new minimization becomes

$$\widehat{Q}_{i+1} = \arg\min_{\widehat{Q} \in \mathcal{F}} \|\widehat{Q} - (\Pi_{\mathcal{F}} \circ T^\pi)\widehat{Q}_i\|^2.$$

This projected Bellman error can be always reduced to zero, because the target function is now projected on $\mathcal{F}$.

A popular choice of projection is the weighted Least Squares projection. LSPI uses this projection with parametric linear function approximators. Because we consider LSPI with Q-functions, the Basis Functions will be functions over state-actions $\phi_i(x,a)$, and as we did in Section 2.4.1, we will form a $\phi \in \mathbb{R}^d$ vector. The

approximation is then defined as $\widehat{Q}_i(x,a) = \phi(x,a)^\mathsf{T} w$. A weighted Least Squares projection, with weights μ, onto this space is defined as

$$\begin{aligned}\Pi_{\mathcal{F}}\widehat{Q}(x,a) =& \phi(x,a)^\mathsf{T} w^*,\\ w^* =& \arg\min_w \sum_{i=1}^{n} \mu(x_i,a_i)\left(\phi(x_i,a_i)^\mathsf{T} w - \widehat{Q}(x_i,a_i)\right)^2.\end{aligned}$$

Our choice of using μ, a symbol we use for distribution, to denote the weights will be justified later in the exposition of LSPI. For a limited number of samples the projection has a closed form solution ([30]). Given samples $(x_i,a_i)_{i=1,\dots,n}$, we define the matrices

$$\begin{aligned}\Delta_\mu =& \operatorname{Diag}(\mu(x_i,a_i))_{i=i}^n : \mathbb{R}^{n\times n},\\ \Phi =& \begin{bmatrix} \phi(x_1,a_1)^\mathsf{T} \\ \phi(x_2,a_2)^\mathsf{T} \\ \vdots \\ \phi(x_n,a_n)^\mathsf{T} \end{bmatrix} : \mathbb{R}^{n\times d},\\ \Pi_{\mathcal{F}} =& \Phi(\Phi^\mathsf{T}\Delta_\mu\Phi)^{-1}\Phi^\mathsf{T}\Delta_\mu.\end{aligned}$$

The combination of linear approximator, projected Bellman error, and weighted least square projection is the basis of the two policy evaluation algorithms we will introduce for use with LSPI. LSTD-Q is a policy evaluation algorithm for Q-functions with a closed form solution, while LSPE-Q uses an almost identical derivation, but computes its solution in an iterative fashion.

To understand the quantities that are approximated by these algorithm we will first define them for countable MDPs with full model knowledge. We already introduced the Φ matrix, that can be constructed for all $S\times\mathcal{A}$ state-action couples. The same can be done for the Δ_μ matrix. We will further define $\boldsymbol{P}^\pi \in \mathbb{R}^{|S||\mathcal{A}|\times|S||\mathcal{A}|}$ as the matrix whose (i,j)-th component correspond to the $(s_i,a_i),(s_j,a_j)$ state action couples, and corresponds to

$$\boldsymbol{P}^\pi_{i,j} = P(s_j|s_i,a_i)\pi(s_j,a_j).$$

Lastly, we slightly abuse the notation and define $R\in\mathbb{R}^{|S||\mathcal{A}|}$ as the average reward for each state-action. The Bellman operator can be then defined as

$$(T^\pi Q) = R + \gamma \boldsymbol{P}^\pi Q.$$

The policy evaluation step is optimized by the fixed point of the projected Bellman error. Reminding the definition of the projection operator, and the linear approximation, this can now be expressed in terms of matrices.

$$\begin{aligned}&\Pi_{\mathcal{F}} T^\pi \widehat{Q} = \widehat{Q},\\ &\Phi(\Phi^\mathsf{T}\Delta_\mu\Phi)^{-1}\Phi^\mathsf{T}\Delta_\mu(R+\gamma\boldsymbol{P}^\pi\Phi w) = \Phi w.\end{aligned}$$

Algorithm 2.6 LSTD-Q

input: $\gamma, \phi_i, \pi,$, samples $(x_i, a_i, x'_i, r_i)_{i=1}^N$
output: $\widehat{w}$
  Initialize $\boldsymbol{A}_0 \leftarrow 0, \boldsymbol{B}_0 \leftarrow 0, \boldsymbol{c}_0 \leftarrow 0$
  for $i \leftarrow 1, \ldots, N$ **do**
    $\boldsymbol{A}_i = \boldsymbol{A}_{i-1} + \phi(x_i, a_i)\phi(x_i, a_i)^\mathsf{T}$
    $\boldsymbol{B}_i = \boldsymbol{B}_{i-1} + \phi(x_i, a_i)\phi(x'_i, \pi(x'_i))^\mathsf{T}$
    $\boldsymbol{c}_i = \boldsymbol{c}_{i-1} + \phi(x_i, a_i) r_i$
  end for
  Solve $\frac{1}{N}\boldsymbol{A}_N w = \frac{1}{N}\gamma \boldsymbol{B}_N w + \frac{1}{N}\boldsymbol{c}_N$

This is a linear equation in d variables and can be solved with any standard technique. In particular the system that needs to be solved can be rewritten as

$$\boldsymbol{A}w = \gamma \boldsymbol{B}w + \boldsymbol{c},$$
$$\boldsymbol{A} = \Phi^\mathsf{T}\Delta_\mu \Phi, \qquad \boldsymbol{B} = \Phi^\mathsf{T}\Delta_\mu \boldsymbol{P}^\pi \Phi, \qquad \boldsymbol{c} = \Phi^\mathsf{T}\Delta_\mu R,$$

where $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}, \boldsymbol{c} \in \mathbb{R}^d$. Therefore even when the number of states and actions increases, for example when they become infinite, the approximation will remain finite in terms of how many BFs we intend to use. Of course in RL problems we do not have access to the $\boldsymbol{P}^\pi$ matrix, and in continuous setting it is impossible to perform the matrix multiplications necessary to construct the linear system. In [31] Lagoudakis and Parr prove that these matrices can be built incrementally by iterating over all states and actions, and using a single state-action couple at a time to build a partial representation. This opens up the possibility to using state-action sampling instead than complete loops over all possible state-actions pairs to obtain an approximate solution that will converge to the real solution as the number of samples increases. This convergence was proved in [29] for LSTD and [64] for LSPE. One of the necessary condition for convergence is that the weights μ used for the projection are equal to the steady-state distribution of the MDP under policy π, and for this reason we used the μ notation. Another important assumption is that all the state-action pairs must be associated to non-null probability of being selected, otherwise some parts of the MDP might remain unexplored and their values could fail to converge. Therefore, a certain degree of exploration must be present in the policy, which for example eliminates deterministic policies.

The full description of the two methods are reported in Algorithm 2.6 and Algorithm 2.7.

Both methods use a normalization term on the various vector and matrices to compensate the slow increase in magnitude that the matrices follow as the number of samples grows up. This is just a modification to improve numerical stability, while the initialization of $\boldsymbol{A}_0$ in LSPE to a small d.p. constant is due to the fact that the

Algorithm 2.7 LSPE-Q

input: $\gamma, \phi_i, \pi, \alpha$, samples $(x_i, a_i, x'_i, r_i)_{i=1}^N$, a small positive constant δ
output: $\widehat{w}_N$
  Initialize $\boldsymbol{A}_0 \leftarrow \delta I, \boldsymbol{B}_0 \leftarrow 0, \boldsymbol{c}_0 \leftarrow 0$
  for $i \leftarrow 1, \ldots, N$ **do**
    $\boldsymbol{A}_i = \boldsymbol{A}_{i-1} + \phi(x_i, a_i)\phi(x_i, a_i)^\mathsf{T}$
    $\boldsymbol{B}_i = \boldsymbol{B}_{i-1} + \phi(x_i, a_i)\phi(x'_i, \pi(x'_i))^\mathsf{T}$
    $\boldsymbol{c}_i = \boldsymbol{c}_{i-1} + \phi(x_i, a_i) r_i$
    $\widehat{w}_i = \widehat{w}_i + \alpha(w - \widehat{w}_{i-1})$ subject to $\frac{1}{i}\boldsymbol{A}_i w = \frac{1}{i}\gamma \boldsymbol{B}_i \widehat{w}_i + \frac{1}{i}\boldsymbol{c}_i$
  end for

Algorithm 2.8 LSPI

input: γ, ϕ_i, samples $(x_i, a_i, x'_i, r_i)_{i=1}^N$
output: $\widehat{\pi}_K$
  Initialize $\widehat{\pi}_0$
  for $k \leftarrow 1, \ldots, K$ **do**
    $\widehat{w}_k \leftarrow$ LSTD-Q run with policy $\widehat{\pi}_{k-1}$
    $\widehat{\pi}_k(x) \leftarrow \arg\max_{a \in \mathcal{A}} \phi(x, a)^\mathsf{T} \widehat{w}_k$
  end for

matrix has to remain d.p. at each iteration.

With the policy evaluation step solved with any of these methods, LSPI is defined in Algorithm 2.8. The implementation is straightforward and follows closely a normal PI scheme, we also reported the simpler case where the same samples are reused at each iteration. It is interesting to note that the full policy is never explicitly constructed, but is implicitly encoded as the greedy policy w.r.t. $\widehat{Q}_i$. We still need to explicitly compute optimal actions for each sample in the dataset, because these actions are needed for the execution of LSTD.

2.4.3 Continuous-State Value Iteration

As we presented extensions of PI in the previous section, there are many examples of Approximate Value Iteration algorithms. We will introduce a version of Q-learning that makes use of linear parametric approximators in this section, and give some more details to Fitted Value Iteration in the next one.

Approximate Q-learning follows the interpretation of the Temporal Difference as a finite difference between a target and the current approximation. In the ideal case this target is the optimal Action-Value function Q^*, so that at each step we want to reduce $\left[Q^* - \widehat{Q}_i\right]$. Of course we do not have access to the real Q^*, so we are going to approximate it with the optimal Bellman operator. Because the goal is to obtain an approximation that is close to the target, we can take the derivative w.r.t.

Algorithm 2.9 Q-learning with linear approximation and gradient descent

input: $\gamma, \alpha_i, \epsilon_i, x_0, \phi_i$
output: w_N
  Initialize w_0
  $i \leftarrow 0$
  do
    $a_i \sim Greedy(\widehat{Q}_i, x_i, \epsilon_i)$
    Execute a_i, obtain r_i, x_{i+1}
    $w_{i+1} \leftarrow w_i + \alpha_i \left[r_i + \gamma \max_{a' \in \mathcal{A}} \phi(x_{i+1}, a')^\mathsf{T} w_i - \phi(x_i, a_i)^\mathsf{T} w \right] \phi(x_i, a_i)$
    $i \leftarrow i + 1$
  while $i \leq N$

the parameters, and perform a series of steps of gradient descent in the parameter space until we converge. Because Q-learning was an on-line algorithm, this extension will also process a single sample (x_i, a_i, x_{i+1}, r_i) at a time. The update rule uses a simple squared loss to measure the distance between the target and the current approximation,

$$\begin{aligned}\widehat{Q}_{i+1}(x_i, a_i) =& \widehat{Q}_i(x_i, a_i) - \frac{1}{2}\alpha_i \frac{\partial}{\partial w}\left[Q^*(x_i, a_i) - \widehat{Q}_i(x_i, a_i)\right]^2 \\ \widetilde{=}& \widehat{Q}_i(x_i, a_i) + \alpha_i \left[r_i + \max_{a'} \widehat{Q}_i(x_{i+1}, a') - \widehat{Q}_i(x_i, a_i)\right] \frac{\partial \widehat{Q}_i(x_i, a_i)}{\partial w}.\end{aligned}$$

The only requirement for the convergence to a local optima of this gradient descent version of Q-learning is that the function approximator is differentiable in the parameters. Various class of function approximators have been used together with this method, such as fuzzy rule-bases, neural networks and linear approximators.

This last class of approximators is used in Algorithm 2.9 to give an example of a possible implementation. The use of linear parameters simplifies the update. The algorithm still needs an exploratory policy to collect samples, because, as in Approximate Policy Iteration, information from the whole state space is needed to ensure that the approximations are meaningful.

A major drawback of Q-learning is its sample inefficiency. Many techniques have been developed to increase its performance, such as eligibility traces or experience replay. A completely different approach is to evaluate all the available data at the same time with a batch method. Fitted Value Iteration, that will be introduced in the next section, is one of the main examples of this approach.

2.4.4 Fitted Q Iteration

Fitted Q Iteration is a really successful off-line, off-policy, Approximated Value Iteration algorithm. Originally proposed in a variant that used regression trees [16], it has been successfully adapted with a variety of function approximators, such as

Algorithm 2.10 Fitted Q Iteration

input: $\mathcal{D}_0 = (x_i, a_i, x'_i, r_i)_{i=1}^N$, λ,γ,tol,K
output: $\widehat{Q}_K$
  Initialize $\widehat{Q}_0 \leftarrow 0$, $k = 1$
  do
    Build new dataset $\mathcal{D}_k \leftarrow (x_i, a_i, x_i, r_i + \gamma \max_{a'} \widehat{Q}_k(x'_i, a'))_{i=1}^N$
    Compute $\widehat{Q}_k$ by regression on $\mathcal{D}_k$
  while $\|\widehat{Q}_k - \widehat{Q}_{k-1}\|_2 \geq tol$ **and** $k < K$

neural networks [46], kernel approximation [17] and tile coding [59]. In the original method proposed in this thesis, Sparse Fitted Q Iteration, we use a regularized linear approximator with the same framework.

Approximate Value Iteration can be decomposed in a sequence of operators similarly to what we introduced for Approximate Policy Iteration. We have the approximator operator F, the optimal Bellman operator T^* and the projection operator $\Pi_{\mathcal{F}}$. For the parametric setting, and usually also for the non-parametric setting, we want to compute a fixed point of the combination of these operators

$$w_{k+1} = (\Pi_{\mathcal{F}} \circ T^* \circ F) w_k.$$

In the model-free, approximated setting, we assume that we do not have access to all of these operators, and that we cannot exactly represent intermediate steps. It is therefore advantageous to compute as many steps as possible implicitly, and to approximate the steps that we cannot compute exactly. Regarding the second problem, we can approximate the first two operators in a similar manner as we did in other approximated methods.

Given a set of samples $(x_i, a_i, x'_i, r_i)_{i=1}^N$ and an approximation of a Q-function $\widehat{Q}$, we can obtain samples $\widehat{Q}^\dagger$ of $T^* \circ F$ as

$$\widehat{Q}_i^\dagger = r_i + \gamma \max_{a'} \widehat{Q}(x'_i, a').$$

As the number of samples increases, the average of the samples $\widehat{Q}^\dagger$ will converge to its mean, the true value of $T^*\widehat{Q}$. This takes care of two of the three operators. The projection step is carried out with any desired regression method, as long as the regression estimates the mean of the regression target. For example Least Squares regression, that will be introduced in detail in Chapter 3 satisfies this condition.

Fitted Q Iteration is reported in detail in Algorithm 2.10. At the first step we build an approximation of the immediate reward function. At each subsequent step we use this approximation to build a new dataset where the reward is substituted with an approximated discounted sum of rewards. Intuitively at each step we are exploiting the definition of action-value function as a sum of discounted rewards to

propagate these rewards for one MDP transition, and build a new representation of this intermediate function.

When all the three operators are contractions, the algorithm is guaranteed to converge. The optimal Bellman operator is a contraction, so the convergence of the whole algorithm depends on the contraction of both the approximator and the regression algorithm. When these condition are met, additional results can be derived on the optimality of the solution. Given the set of all fixed points of $(\Pi_{\mathcal{F}} \circ T^* \circ F)$, we define an error ϵ that measures the minimum distance between the optimal policy Q^* and the closest of these points. That is, ϵ measures the minimum possible error the algorithm can obtain. The error when the algorithm converges to the optimal fixed point is then bounded by

$$\|Q^* - \widehat{Q}^*\|_\infty \leq \frac{2\epsilon}{1-\gamma}.$$

When we cannot guarantee contraction properties, the algorithm might not converge. Another approach to prove convergence without contraction assumptions is derived in [42]. This is the approach we will use to guarantee that with high-probability Fitted Q Iteration converges to a near-optimal solution as long as the MDP satisfies certain assumption reported in Section 4.2, and that the error committed when approximating the function is not too large.

2.5 Learning with Multiple Tasks and Transfer Learning

All the algorithms we have considered this far aim to solve an MDP problem using data collected from it. When the information on the single problem is rich enough, these techniques have excellent guarantees and quickly find optimal or near-optimal solutions.

In many real world scenarios, this abundance of information is hard to satisfy. Many practical applications are examples of this problem. Historical data is in the past, and it is often impossible to collect more. Even problems that take place in the present can have limitation on the number of samples that can be collected. One reason might be that executing all possible actions is strictly impossible, like testing too many treatments on a patient group. Another is when the collection itself might be costly or dangerous, for example when a complex and fragile piece of machinery is involved. One last example is in problem such as game playing, when we want to learn a good strategy as quickly as possible to avoid the risk of losing any chance of winning. In all this cases there is a need for efficient solution that can extract as much information from the samples as possible.

An orthogonal and interesting approach is instead the one used in Multi-Task Learning. Because we have a small quantity of samples, and obtaining more samples

is costly or impossible, a different solution is to look at alternative tasks that are similar enough to the current task to provide additional information useful to solve the initial problem. A simple and intuitive example is a set of MDPs that share their dynamics but not their reward functions. If it is possible to share information about the dynamics among the various tasks, the samples from each task can be pooled together to improve the average performance, or even the performance of each tasks.

A similar goal is central to Transfer Learning. As the name suggests it aims to bring information from one domain, the source task, to another domain, the target task. The two tasks can be extremely different as long as the transfer procedure can extract information for the target task from the samples available to the source task.

The fields of Multi-Task learning and Transfer Learning are mature and many results from all kind of Machine Learning methods, Supervised, Unsupervised and Reinforcement Learning, are available. For what regards this thesis we are interested in Transfer and Multi-Task techniques for Reinforcement Learning. The reason for this interest is that the main result of this thesis, a Multi-Task Fitted Value Iteration algorithm, can be better compared to other RL methods. We will present a few methods of interest, the reader can refer to [56] or [32] for a more detailed survey.

We first formalize the notation necessary for our setting. In a Multi-Task Reinforcement Learning problem we are faced with T different tasks, each an MDP defined by $\mathcal{M}_t = (S_t, \mathcal{A}_t, P_t, R_t, \gamma_t)$. Our goal is to improve the average performance across all tasks, or in a more restrictive setting, to strictly improve the performance of each task taking advantage of the shared information. The negative case when the addition of more tasks is detrimental to the overall performance is referred to as negative transfer.

A first interesting result regarding countable MDPs is given in [8], within a framework called PAC-MDP sample complexity. Their staring point is an algorithm called E^3, the Explicit Explore or Exploit algorithm. Unlike SARSA or Q-learning, E^3 tries to build a model representation of the MDP, and uses confidence bounds on this model to control the exploration of the MDP and the exploitation of the constructed model. In other words E^3 tries to address the exploration/exploitation dilemma. Because it is defined on a countable MDP, the sample complexity of E^3 is in the order of $\mathcal{O}(\sum_{t=1}^{T} b|S_t||actionspace_t|)$, where the b factor represent an upper bound on the number of possible future state every transition can reach. This bound shows a direct dependence on the dimensionality of the MDPs, which is expected for countable MDPs, and also scales linearly with the number of tasks, because E^3 was introduced as a single task algorithm. By carefully exploiting the Multi-Task setting [8] can guarantee performance improvement with no negative transfer. In particular their algorithm expects to receive as tasks a number of MDP. Information sharing comes from the assumption that the MDPs are sampled from a small number of possible MDP classes C. The algorithm then initially treats each MDP as an unknown, single task problem, and runs E^3 to estimate its model. Once the model

is defined well enough, the MDPs can be clustered into classes, and the information among each class can be shared. The sample complexity after the initial estimation phase reduced to a factor independent of the dimensionality of the tasks and the number of tasks, driven instead by the number of classes. This shows that at the single class level, the sharing of samples provide an improvement.

An approach based more on the idea that the value functions of the various tasks share a common representation structure is explored in [33]. While Sparse Fitted Q Iteration assumes that the value function can be represented as a linear combination of a small subset of features that is shared across tasks, Bayesian Multi-Task RL models the value function with a Gaussian Process, and the approximation used is a linear combination of features where the weights w are sampled from a Gaussian distribution $w \sim \mathcal{N}(\tau, \Sigma)$. The model of the reinforcement is then $R = \Phi w - \gamma \Phi' w + \epsilon$ where ϵ is a zero mean noise. Using standard Bayesian inference it is possible to estimate the most likely parameters for the distribution of w according to the data. But if access to several similar task is possible, trying to estimate some hyper-parameters that control the distribution of the τ_t and Σ_t parameters of each task might be more efficient. This is the principle of the Single-Class Multi-Task Bayesian RL algorithm, that establish hyper-priors $\psi = (\tau, k, \nu, \Sigma, \alpha, \beta)$ and then uses an Expectation Maximization algorithm to find a good set of hyper-parameters to fit the data. This approach allows all the samples from all tasks to influence the hyper-parameters, and therefore the learning will be more efficient than learning each class separately. Using of the right conjugate priors allows the E-step and M-step of Expectation Maximization to be carried out efficiently. A further generalization can be obtained if we consider more than one class of similar tasks, and establish an additional hyper-prior distribution over the possible classes. This results in the Multi-Class Multi-Task Bayesian RL. The main advantage of the multi-class approach is that new MDPs can be quickly identified as members of one of the classes using the class prior, and performance can improve. A similar approach is taken in [63].

Chapter 3

Linear Regression and Regularization

In Chapter 2 we introduced Markov Decision Processes, and some of the techniques available to solve them. Among them Fitted Q Iteration, introduced in Section 2.4.4, forms the basis of the original contribution of this thesis, Sparse Fitted Q Iteration. Fitted Q Iteration is a flexible framework, where the flexibility derives from the possibility of freely choosing the class of approximating functions used with the algorithm. In SFQI's case, the chosen function approximator is a linear function approximator with non-linear Basis Functions. This allows the method to use a rich description of the problem through complex features, while maintaining the simplicity of a linear approach. The projection step required by FQI needs to estimate the mean of a set of Q_i samples as a random variable dependent on the linear approximator parameters. The use of a squared loss to penalize errors in the estimation completes the definition of a Least Squares linear regression problem. Because the motivating problem of this thesis is to solve a set of MDP problems where each of the separate problems does not provide sufficient information to be solved alone, we will see that in these cases Ordinary Least Squares regression cannot perform the projection step required by FQI. For this reason in this chapter we will also introduce regularized alternatives to the OLS problem, that can help cope with the scarcity of samples.

In detail, we will introduce the regression and linear regression problems in Section 3.1. Particularly, in Section 3.1.3 we introduce the Ordinary Least Squares regression problem, and some methods to solve it. In Section 3.2 we will detail several regularized variations of the Least Squares regression problems. Regularized techniques can help solve underconstrained problems where the number of observations vastly exceeds the number of regression variables. At first we introduce methods that operates similarly on all variables, and later more advanced methods that can operate on groups of variables instead. This group level techniques help us introduce a multi-task approach in regularized regression. Finally, in Section 3.3 we introduce

several existing methods that make use of regularization techniques in Approximate Reinforcement Learning.

3.1 Regression

In mathematics the problem of finding the function f in a function space $\mathcal{F}$ that best represents another function is called function approximation. In the exact setting this is the field of approximation theory, where we know the exact shape of the function we need to approximate, and are interested in the approximation for reasons such as obtaining a compact representation or a form that allows for easy computation. Examples of this field are interpolation techniques, or famous approximation equivalences such as the Fourier transform.

When the function itself is unknown, we can still hope to recover an approximation through the analysis of some samples. Under a common assumption, the samples themselves are now just the image of randomly sampled points, and are therefore themselves random variables. In addition to this, the application of the function might introduce some error, for example due to the fact that we cannot exactly measure its results. In statistical decision theory this problem is called regression, in particular we want to find the regression function f that minimizes the prediction error on future samples.

We will now formally define this problem and some initial results regarding its solution. For a more detailed introduction refer to [22].

3.1.1 Problem Definition

We assume that given a random input vector X and some random target variable Y, for each realization $\{y_i, x_i\}_{i=1}^n$ the relationship

$$y_i = f^*(x_i) + \epsilon$$

holds, where ϵ is a zero mean error.

A regression function f is a function that minimizes the expected prediction error w.r.t. some loss function

$$\mathrm{EPE}(f) = \mathbb{E}\left[L(Y, f(X))\right] = \int_{x,y} L(y, f(x)) Pr(dx, dy).$$

Using a squared loss, and Bayes rule we can derive

$$\mathrm{EPE}(f) = \mathop{\mathbb{E}}_{X} \mathop{\mathbb{E}}_{Y|X} \left[(Y - f(X))^2 | X\right]. \tag{3.1.1}$$

To minimize this quantity we can simply obtain the function that is pointwise equal to the conditioned expected value of Y given X

$$f(x) = \mathbb{E}\left[Y | X = x\right].$$

This is what we introduced in Fitted Q iteration when we anticipated that the regression problem seeks to find the mean of the target variable.

A simple way to estimate this conditional mean is to use a k-nearest neighbor approximation. In k-nearest neighbor approximation $f(x)$ is simply the average of the y_i associated with the k closest x_i samples to x. As the number of samples n grows to infinity, and k grows as well, this quantity converges to the true expectation. On the other hand, in real problems we cannot obtain infinite amounts of data to build the average. In addition to this, as the number of dimension grows, the neighborhood of a point x grows to include larger and larger parts of the whole space. It is easy to picture this curse of dimensionality when we think of an evenly spaced line against an evenly spaced cube.

In this finite setting, a dilemma called bias-variance trade-off arises.

$$\begin{aligned}&\mathbb{E}\left[(Y - f(X))^2\right] = \mathbb{E}\left[(Y - f^*(X) + f^*(X) - f(X))^2\right] \\ &= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[(f^*(X) - f(X))^2\right] + 2\,\mathbb{E}\left[(Y - f^*(X))(f^*(X) - f(X))\right].\end{aligned}$$

The third term is equal to 0, and the first term is the irreducible error our regression will commit. Applying the same decomposition to the second term we obtain

$$\mathbb{E}\left[(Y - f(X))^2\right] = \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[(f^*(X) - \mathbb{E}[f(X)])^2\right] + \mathbb{E}\left[(\mathbb{E}[f(X)] - f(X))^2\right].$$

The second term, called bias, measures the minimum distance between the best function approximation possible in $\mathcal{F}$ and the true function. In the limit for example k-nearest neighbor is an unbiased approximator. The third term is the variance of the approximator. When we have infinite samples this term disappears, but the main drawback of k-nearest neighbor is that when the number of samples is limited this term can be large. Depending on the situation it might be better to introduce some bias, or in other words choose an approximator that makes erroneous assumptions and cannot represent exactly the function, in exchange for a large reduction in variance when the number of samples is limited.

Reduction of variance is the reason why we introduce model assumptions in the form of limiting the approximator to be a linear combination of BFs. As we will see this reduces the variance as the number of samples grows, but the variance still depends on the number of BFs that we use to approximate. Therefore another trade-off between sample size and number of features must be considered. Regularization will help in achieving a better bound on the number of features.

3.1.2 Linear Regression

In linear regression we make the assumption that the target variable is a linear combination of the inputs. For simplicity we will consider now only deterministic BFs, and will avoid writing $\phi(x)$, assuming that the samples have been already

mapped. The model reduces to

$$y_i = x_i^\mathsf{T} w^* + \epsilon.$$

We also assume Gaussian i.i.d noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$. While the choice of Least Squares and Gaussian noise can seem arbitrary, it can be cast in the larger framework of Maximum Likelihood Estimation. Instead of the EPE, the MLE goal for a model $y_i = f_w(x_i) + \epsilon$ is to maximize

$$\text{MLE}(w) = \sum_{i=1}^{n} \log Pr_w(y_i),$$

where Pr_w is the probability distribution of the target samples according to current approximator parameters w. If we choose this conditional distribution to be $Pr(Y|X, w) = \mathcal{N}(f_w(X), \sigma^2)$, we obtain again the least square criterion to maximize the problem.

Using the definition of EPE from Equation 3.1.1, we can take the derivative w.r.t. w, and obtain a closed form solution

$$w = \mathbb{E}\left[X^\mathsf{T} X\right]^{-1} X^\mathsf{T} Y.$$

Already the simple linear structure allows us to compute a solution efficiently. Because we assume that the model is valid, the bias term is 0. The EPE error will then be equal to the irreducible noise plus the variance of the predictor. Without too stringent assumptions, it can be shown that for a random sample x_0, this quantity converges to

$$\mathop{\mathbb{E}}_{x_0} \text{EPE} \sim \frac{\sigma^2 d}{n} + \sigma^2.$$

The variance is proportional to the ratio of the dimension of the feature vector and the number of samples. Compared to the curse of dimensionality explosion that k-nearest neighbors had, we traded a reduction in variance with the possible introduction of a bias.

We will see now how to estimate the w vector using all the available data, in order to reduce the variance, with a procedure called Ordinary Least Squares.

3.1.3 Ordinary Least Squares

The simple structure of the linear estimator allows for a closed form solution to the n sample problem of linear regression with Least Squares penalty. Formally, we want to minimize

$$\widehat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^\mathsf{T} w)^2.$$

Because we assume that the dataset is now a fixed, single realization of samples extracted from the distribution, we do not need the random variables X and Y anymore. Instead we will indicate with $X \in \mathbb{R}^{n \times d}$ the matrix whose row X^i corresponds to the i-th sample, and with $Y \in \mathbb{R}^n$ the vector whose i-th component is the y_i sample. The problem can be rewritten as

$$\widehat{w} = \arg\min_w (Y - Xw)^\mathsf{T}(Y - Xw). \tag{3.1.2}$$

This is a convex optimization problem, and the solution can be found by looking for the value that makes the derivative of the objective null. By taking the derivative w.r.t. to w, and equating it to 0 we obtain

$$\begin{aligned} \frac{\partial}{\partial w} = X^\mathsf{T}(Y - Xw) = 0, \\ X^\mathsf{T}Xw = X^\mathsf{T}Y. \end{aligned}$$

The first equation clearly characterizes the solution. The solution is the vector that makes the residuals $(Y - Xw)$ perpendicular to the subspace of $\mathbb{R}^d$ that composes the column range of X. In other words, the solution will be the orthogonal projection of the target Y onto the space of linear functions $\mathcal{F}$ spanned by X. When the $X^\mathsf{T}X$ matrix is non-singular, the unique solution is

$$\begin{aligned} \widehat{w} &= (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y, \\ \widehat{y} &= X^\mathsf{T}(X^\mathsf{T}X)^{-1}X^\mathsf{T}Y. \end{aligned}$$

The projection matrix $X^\mathsf{T}(X^\mathsf{T}X)^{-1}X^\mathsf{T}$ is similar to the matrix used in Section 2.4.2.

This illustrates the main drawback of Ordinary Least Squares. When the matrix $X^\mathsf{T}X$ is singular, the projection is not unique and there is not a definite solution to the problem. In particular, whenever the number of samples n is smaller than the number of features d, this problem will arise. Even with more samples than features, the matrix might still be singular if the samples are not well distributed, or alternatively the matrix might be close to singular and the inverse operator might be numerically unstable. This relegates OLS to problems where the number of samples is larger than the number of features, and the samples are of good quality. We will see how the introduction of regularization mitigates this problem and allows different formulations that can find a Least Squares solution with the additional introduction of constraints on the solution found.

3.2 Regularization

Regularization is a term introduced in mathematics, optimization and machine learning to indicate all the techniques that use additional model assumptions to solve more robustly ill-posed problems, or select a single solution out of a large or infinite

set of candidates. In this thesis ill-posed usually refers to matrix inversion problems where the matrix is singular or close to singular.

More formally, a regularized problem introduces a regularizer or penalty function $\Omega(\cdot) : \mathcal{F} \to \mathbb{R}$ that expresses an additional goal in the optimization problem of finding the squared loss minimizer. The new optimization problem will then have as a minimum a new solution that will minimize a balanced objective composed by both the loss function and the penalty function. The balance between the two objectives is controlled by a regularization parameter λ, with $\lambda = 0$ corresponding to the unregularized problem and $\lambda = \infty$ corresponding to optimizing only the penalty function. The regression problem becomes

$$f : \arg\min L\left(f, Y\right) + \lambda\Omega\left(f\right).$$

Different choices for the Ω penalty produce vastly different optimization problems. We will now introduce some of the most representative penalty functions and the corresponding optimization problems. In Section 3.2.1, simple ℓ_2 regularization is introduced, to obtain minimum norm solutions. Then in Section 3.2.2 a non-linear ℓ_1 penalty results in the LASSO problem, where for the first time we seek to obtain sparse solutions. Finally, in Section 3.2.3 we introduce a combination of these two techniques, to obtain solutions that are sparse at the level of groups of variables. This will allow us to model problems where more than one task is present and solve them efficiently using samples from all the tasks.

3.2.1 Minimum Norm Solutions with ℓ_2 Regularization

When the $X^\mathsf{T}X$ matrix is singular, its null-space, or kernel, is not limited to a single vector, but is a subspace. If we define $A = X^\mathsf{T}X, b = X^\mathsf{T}Y$, then the vector b either lies in the range of A, or it does not. If $b \notin \mathrm{Ran}(A)$, then the system has no solution. If instead $b \in \mathrm{Ran}(A)$ then any vector $w = v + z$ such that $Av = b$ and $z \in \mathrm{Ker}(A)$ is a solution. Therefore OLS with a singular matrix will either have no solution or infinite solutions. As we said, every time the number of samples is smaller than the number of features the A matrix will be singular.

We will consider first the case when $b \in \mathrm{Ran}(A)$. Among the infinite Least Squares solutions we want to pick one. Minimum norm Least Squares decides to pick the solution w with minimal norm $\|w\|_2$. A straightforward way to express this is through a penalty $\Omega(w) = \|w\|_2^2$. This formulation allows for a concise representation taking advantage of the fact that $\|w\|_2^2 = w^\mathsf{T}w$ using the standard definition of ℓ_2 norm for vectors on a Euclidean space. The final formulation using a squared loss is usually called Ridge regression [23]

$$\begin{aligned}\widehat{w} &= \arg\min_w \|Y - Xw\|_2^2 + \lambda \|w\|_2^2 \\ &= \arg\min_w (Y - Xw)^\mathsf{T}(Y - Xw) + \lambda w^\mathsf{T}w.\end{aligned}$$

Ridge regression is a shrinkage method. That is, the values of the single regression variables w_i are shrinked toward zero by the penalty. This will select among the infinite solutions the one with minimum norm. It can be shown that as λ increases, the norm of the Ridge solution shrinks. At the two extremes we have $\lambda = 0$, where the infinite OLS can assume an infinite norm, and $\lambda = \infty$ where the regularization selects $w \cong 0$. One of the main advantages of this formulation, is that with a similar derivation based on the convexity of the function and the null derivative in the minimum, we obtain a closed form solution.

$$\frac{\partial}{\partial w}(Y - Xw)^{\mathsf{T}}(Y - Xw) + \lambda w^{\mathsf{T}} = 2X^{\mathsf{T}}Xw + 2\lambda w - 2X^{\mathsf{T}}Y = 0,$$
$$\widehat{w} = (X^{\mathsf{T}}X + \lambda I)^{-1}X^{\mathsf{T}}Y.$$

As we can see the inverse is now based on a perturbation of the original A matrix. The addition of a small identity matrix guarantees that the matrix that is going to be inverted will be non-singular. Therefore the solution of a Ridge problem exists regardless of the X samples.

It is interesting to rewrite the formula of the Ridge solution in terms of the singular values of the X matrix. Given a singular value decomposition $X = U\Sigma V^{\mathsf{T}}$, the values σ_i are called singular values of the matrix X. The ridge solution can then be expressed as

$$X\widehat{w} = U\Sigma(\Sigma^2 + \lambda I)^{-1}\Sigma U^{\mathsf{T}}Y = \sum_{i=1}^{d} v_i \left(\frac{\sigma_i^2}{\sigma_i^2 + \lambda}\right) v_i^{\mathsf{T}} X^{\mathsf{T}}Y.$$

Ridge regression shrinks the singular values proportionally to the value of λ. Because the singular values σ_i^2 are equal to the eigenvalues of $X^{\mathsf{T}}X$, and $X^{\mathsf{T}}X/n$ converges to the covariance matrix, we can give an interpretation of Ridge regression in terms of Principal Component Analysis. The eigenvectors v_i correspond to the Principal Components, and Ridge regression shrinks the magnitude of the associated eigenvalues by a factor smaller than 1. The values that will be shrinked proportionally less are the ones associated with the largest eigenvalues, that in turn correspond to the directions with the largest sample variance. Ridge regression then implicitly reduces the regression coefficients in the direction of the least variance. This reduction in variability is often quantified as a number called effective degrees of freedom, that is strictly decreasing as λ increases.

3.2.1.1 Tikhonov Regularization

Introduced in [58], Tikhonov regularization is a more general formulation, that includes Ridge regression as a special case. In particular, when the various features that compose the X matrix are not on the same scale, simply penalizing their magnitude might pose problems. A superfluous variable that is large might greatly

increase the norm penalty, forcing the regularization to penalize also important variables. For this reason, many techniques assume, without loss of generality, that each feature vector is normalized so that they all have the same order of magnitude.

An alternative solution is to modify the problem to use a different penalty function. Tikhonov regularization makes use of a Tikhonov matrix Γ to build a weighted norm of the objective. The objective becomes

$$\begin{aligned}\widehat{w} &= \arg\min_w \|Y - Xw\|_2^2 + \lambda \|\Gamma w\|_2^2 \\ &= \arg\min_w (Y - Xw)^\mathsf{T}(Y - Xw) + \lambda w^\mathsf{T}\Gamma^\mathsf{T}\Gamma w, \\ \widehat{w} &= (X^\mathsf{T}X + \lambda\Gamma^\mathsf{T}\Gamma)^{-1}X^\mathsf{T}Y.\end{aligned}$$

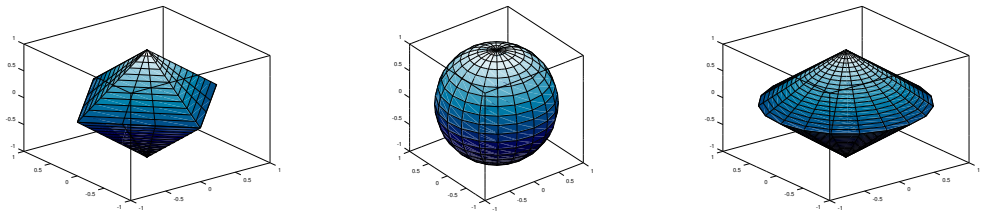
When $\Gamma = I$, we recover the Ridge regression. Other choices of penalty can for example scale the features penalties to take into account difference in magnitude, or even more complicated approaches.

The main drawback of Tikhonov regularization, and therefore Ridge regression, is that the derivative of the regression variables w_i is linear. This means that when the variables are close to 0, the derivative quickly decreases, until shrinking further the variable has little effect on the objective function. Therefore, while Ridge regression succeed in shrinking the variables toward zero, it will not make the last step and completely zero out parts of the w vector. In other words, the solution will be small but not sparse. This means that compared to OLS, we can obtain solutions even when $n < d$, and we will obtain more stable results numerically, but the number of dimensions we can manage still needs to be linear in the number of sample. In the next section we will see how the use of a non-linear ℓ_1 norm will allow us to manage exponentially many features in the number of samples, as long as the underlying true solution is sparse.

3.2.2 Sparse Solutions with ℓ_1 Regularization

The simplicity of linear approximators allows to obtain closed form solutions such as in the Ridge problem, and convergence results for algorithms based on it. More importantly, linear models have an intuitive interpretation and the values that the weights assume can give insight on the relationship of the input features with the target variable that the regression seeks to estimate.

Among these relationships, one of the most indicative is the association of a 0 weight with a feature. Zeroing out a parameter implies that the corresponding feature has no influence on the result of the prediction. A solution where only a small subset of important variables are not zeroed is called sparse, and it is extremely useful when one of the goal of the regression is to find which are the influential variables in a problem, as well as their influence. The problem of selecting only a subset of useful features in a machine learning problem is referred to as feature selection. Another



ℓ_1 ℓ_2 $\ell_{2,1}$

Figure 3.1: Visualization of various norms. The surface corresponds to the $\|w\| = 1$ ball, where $w \in \mathbb{R}^3$.

possible goal is to instead learn a useful feature representation. As we will see in Chapter 5, Multi Task Feature Learning is a Least Squares approximator that tries to learn features during the minimization problem. Feature learning and feature selection are two techniques commonly used in dimensionality reduction.

The use of ℓ_1 regularization can act as a soft version of feature selection, where the features are shrinked all the way until the 0. Among all existing methods, one of the most popular ℓ_1 regularized Least Squares problems is the variant called Lasso. It has received extensive study and wide practical use. We will now introduce it and some interesting results on its ability to recover a sparse representation underlying the function.

3.2.2.1 Lasso

The original Lasso problem, proposed in [57], is defined as

$$\widehat{w} = \arg\min_{w} \|Y - Xw\|_2^2 + \lambda \|w\|_1 .$$

The difference between the ℓ_2 norm penalty and the ℓ_1 norm penalty can be visualized by looking at the possible values of two variables w_1, w_2 given their unit norm. In Figure 3.1, we can see that the ℓ_2 norm corresponds to a circle, or in general as we will see to an ellipsoid. The ℓ_1 norm instead corresponds to the smallest convex envelope around the set of $0, 1$ values. In particular, it is the smallest envelope that remains convex. In practice this characteristic will make it more likely for the projection of the regression vector on this convex set to hit the set on one of the extremities, where some of the values are exactly zero.

This is not only important because we obtain a sparse solution. When the number of samples is much smaller than the number of features, OLS cannot find a solution, and even the solution that Ridge regression finds can be too biased. But if we assume that the underlying true representation is sparse, or in other words the true generating vector w^* has d components but only s non-zero components, we can reduce the complex d dimensional problem, for which we do not have enough samples,

to a smaller s dimensional problem that can be solved. Under several complex assumptions, but not for a too restrictive class of problems, Lasso can effectively operate in the smaller space and recover the true vector w^* with high probability. In [5], a bound on the error of the reconstruction is given with the form

$$\|X(w^* - \hat{w})\|_2 \leq C\frac{s\log(d)}{n} \tag{3.2.1}$$

for some large constant C. In other words, Lasso is not constrained to have the same order of magnitude for the total feature d and samples n, but it can introduce exponentially many useless features. This is a big improvement over OLS and Ridge, because the limitations of linear approximation can be reduced with the use of a large dictionary of Basis Functions.

Computing the Lasso is not as simple as computing an OLS or Ridge solution. Over the years several proposed techniques improved performance. Among them it is important to mention Least Angle Regression, or LARS, introduced in [13]. Following a different theoretical principle and interpretation than the Lasso, LARS is an active set technique that starts with a single active variable, the one most correlated with the target, and slowly increases the magnitude of the variables in the active set. The variables are increased in a way that keeps their correlation with the residual $r = y - \widehat{y} = y - X\widehat{w}$ equal. As the values in the active set converge to their OLS value, their correlation with the residual reduces, until a variable outside the active set is a better predictor than the ones in the active set. At this point the new variable is introduced and the process continues. The only modification necessary to obtain a Lasso solution is that in LARS variables can change sign freely, while to obtain a Lasso path variables that hit 0 are excluded from the active set. LARS is an extremely efficient method that brings solving a Lasso problem in the same cost range as an OLS solving cost. In particular the update direction is given by $\delta = (X^\mathsf{T}X)^{-1}X^\mathsf{T}r$. Moreover it can compute the whole solution path of Lasso efficiently, that is it finds the Lasso solution for all possible λ.

Another interesting solution is proposed in [6]. This article generalizes an iterative algorithm called Shooting Lasso to execute updates in parallel. Under mild assumptions, parallel updates, if carried out not too many at a time, do not conflict with each other, and the whole algorithm converges to a Lasso solution. Parallel techniques are extremely important when considering the direction that modern computing is taking, with computer system that are enormous distributed aggregation of heterogeneous units.

All the techniques presented this far treat the penalization of each regression variable indiscriminately. More general techniques can treat different variables differently, in order to better tune the direction of the regularization. In the case of this thesis, we have an underlying assumption of similarity within a group of tasks. We can then group these variables according to the task, and obtain better results than if we simply tried to obtain a sparse solution. This different level of sparsity,

called group sparsity, can be induced with the use of mixed $\ell_{2,1}$ norms. In the next section we will introduce a problem that makes use of them, and methods for its solution.

3.2.3 Group Sparse Solutions with Mixed $\ell_{2,1}$ Regularization

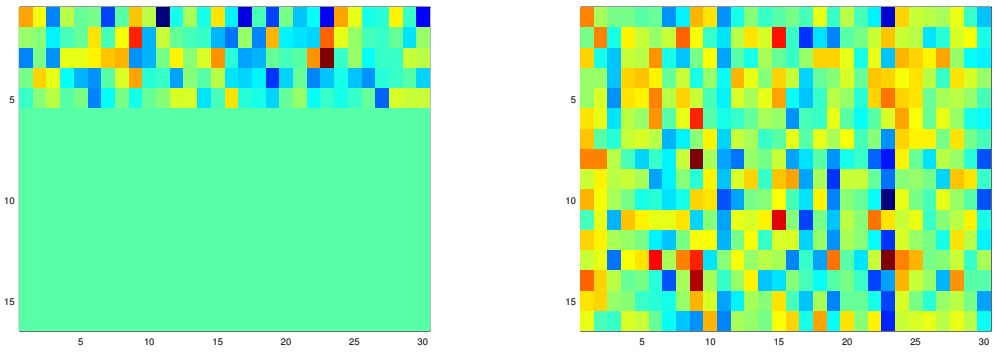
When the number of variables exceeds the number of samples, the choice of ℓ_1 regularization helps to find an underlying sparse structure that allows for a good solution to emerge. On the other hand, when the number of variables exponentially exceeds the number of samples, then even Lasso regression cannot recover the true solution.

An interesting case is when we have previous knowledge on the relationship between the variables. For example we might know that a group of variables are useful only together, so we are not interested in setting them to zero separately, but to set them to zero either all together or not.

In particular, for the main results of this thesis, we expect the w regression variables to be the concatenation of $\{w_t\}_{t=1}^{T}$ vectors, each corresponding to a different problem or task. In this multi-task approach each regression problem is defined on the same space, that is, each vector $w_t \in \mathbb{R}^d$ is combined with the same Basis Function. In other words, the various tasks share the same features, but the samples that generate these features will be different for each task. In particular, each weight w_{it} corresponds to the same feature ϕ_i applied to sample x_{it}. If we believe that a feature is important across all tasks, we can express this with a group $\{w_{i,1}, \dots w_{i,T}\}_{i=1}^{d}$ that includes all the weights corresponding to the same feature across tasks. With this setup, if all the tasks share a low-dimensional representation of dimension s, then only s of the groups will be non-zero. This idea of group sparsity produces a different result compared to simply applying Lasso to each task separately, and as we will see in Section 4.3, this can allow us to break through the limits that Lasso has in terms of useless dimensions. We will now introduce the Group Lasso problem, and leave more of the theoretical analysis associated with it for the next chapter.

3.2.3.1 Group Lasso

We will now introduce the formal description of a Multi-Task Group Lasso problem. It is a particular case of a Group Lasso problem where each group has the same number of variables, and can be expressed more concisely using a matrix $\ell_{2,1}$ norm. In particular we consider $t = 1, \dots, T$ tasks, and for each task a vector of regression variables w_t, input samples X_t and target samples Y_t, as well as a regularization parameter λ. The $W \in \mathbb{R}^{d \times T}$ regression variable matrix is constructed such that its columns are the w_t vectors. The groups are identified as the rows of the W matrix w^i, such that each groups corresponds to the same feature across tasks. The penalty is formalized as the ℓ_1 norm of the ℓ_2 norm of the groups, or in other words as the



Matrix with low $\|W\|_{2,1}$ Matrix with high $\|W\|_{2,1}$

Figure 3.2: Visualization of $\|W\|_{2,1}$ penalties.

sum of the ℓ_2 norm of the rows of W. As specified in Appendix A, this is denoted as $\|W\|_{2,1}$. We gave a visualization of this norm in Figure 3.1, and we provide an additional intuition of the $\|W\|_{2,1}$ in Figure 3.2, where the first matrix, which is group sparse, has only a few non-zero lines, while the dense second matrix has values evenly distributed across all groups.

In detail, the Group Lasso problem is defined as

$$\begin{aligned}\widehat{W} &= \arg\min_{W} \sum_{t=1}^{T} \|Y_t - X_t w_t\|_2^2 + \lambda \sum_{i=1}^{d} \left\|w^i\right\|_2 \\ &= \arg\min_{W} \sum_{t=1}^{T} \|Y_t - X_t w_t\|_2^2 + \lambda \|W\|_{2,1} .\end{aligned}$$

Initially introduced in [65], the Group Lasso minimization problem is centered on a particular property of the $\|w_t\|_2$ function. Unlike the $\|\cdot\|_2^2$ of a Ridge regression, the simple 2-norm is not differentiable in 0. Therefore for this particular case we have to consider not its derivative but its subgradient. The subgradient, a generalization of the derivative for non-differentiable functions, is often used in convex optimization. The subgradient $\frac{\partial f(w)}{\partial w_0}$ at a point w_0 is defined as any vector v such that for all other vectors w

$$f(w) - f(w_0) \geq v^{\mathsf{T}}(w - w_0).$$

Intuitively, the subgradient is a lower bound on the growth rate of a convex function in any direction. It can be represented as a supporting hyperplane in w_0 such that the image $f(w)$ of every other point in the domain is above the plane. The subgradient is not always unique, and the set of all possible subgradients is called subdifferential. In convex optimization, a sufficient and necessary condition for the optimality of a solution, is that the subdifferential of the objective function at the solution point

Algorithm 3.1 Block Gradient Descent Group Lasso

input: X, Y, λ, tol, stepsize α
output: $\widehat{W}$
  Initialize $\widehat{W}$
  do
    $\widehat{W}_{old} = \widehat{W}$
    for $i \leftarrow 1, \ldots, d$ **do**
      if $\left\| {X^i}^\mathsf{T} r_{-i} \right\|_2 \leq \lambda$ **then**
        $\widehat{w}^i \leftarrow 0$
      else
        do
          $\widehat{w}^i_{old} = \widehat{w}^i$
          $\widehat{w}^i \leftarrow \max\left(1 - \frac{\alpha\lambda}{\left\|\widehat{w}^i + \alpha {X^i}^\mathsf{T} r_{-i}/n\right\|_2}, 0\right)\left(\widehat{w}^i + \alpha r_{-i}^\mathsf{T} X^i/n\right)$
        while $\widehat{w}^i_{old} = \widehat{w}^i$
      end if
    end for
  while $\left\|\widehat{W}_{old} - \widehat{W}\right\|_{2,1} \geq tol$

contains the null vector. When the convex function is differentiable in a point, then its subdifferential contains only the derivative.

The subdifferential of $\|\cdot\|_2$ is

$$\frac{\partial \|w\|_2}{\partial w} = \begin{cases} \frac{w}{\|w\|_2} & \text{if } w \neq 0 \\ z : \|z\|_2 \leq 1 & \text{if } w = 0 \end{cases}.$$

As we will see, this property allows the case $w^i = 0$ to be treated differently during the optimization, and allows entire blocks to be set equal to zero.

As an example, we introduce a Block Gradient Descent algorithm, presented in [51], that can be used to compute a Group Lasso solution. The details are included in Algorithm 3.1. The name Block Gradient derives from the fact that the algorithm iterates among groups, each called block, updating each block iteratively. In particular, given a block-diagonal matrix X composed by the X_t matrices, X^i is the submatrix composed by the columns associated with the i-th groups, which corresponds to the i-th feature across tasks, and to the w^i row of the W matrix. The r_{-i} vector represents the behaviour of the whole predictor without considering changes to w^i, and is defined as $r_{-i} = Y - \sum_{j \neq i} X^j {w^j}^\mathsf{T}$. We have an outer loop that iterates over all groups until convergence, and an inner loop that updates iteratively a single block, again until convergence. The sparsity property is introduced by the condition $\|{X^i}^\mathsf{T} r_{-i}\|_2 \leq \lambda$, that forces the whole group w^i to 0 if satisfied. With some simple optimizations, the computational cost of the algorithm is dominated by the

computation of the residuals and the gradient direction, and is in the order of $\mathcal{O}(nd)$ for each iteration of the single block descent.

This algorithm is intrinsically iterative. Even if matrix multiplications can be easily parallelized, and run in parallel on most modern linear algebra implementations, the loops need to update a single group at a time, because the direction of the upgrade depends on the values of the other blocks. When the updates are serial, convergence is guaranteed. As we will see, in Chapter 5 we introduce a modified Multi-Task Feature Learning algorithm that uses Group Lasso penalties and can compute solutions to the Group Lasso problem. Although the computational cost of a single iteration might be higher in our method, we will see that the most costly operation can be carried out separately for each task, and is therefore straightforward to parallelize.

3.2.4 Extensions of Group Sparse Solutions

Algorithm 3.1 is actually a specialized version of a more general method introduced in [51]. In the original paper the method is introduced to solve a Sparse Group Lasso problem, an extension of Group Lasso where sparsity is induced not only at the group level, but also inside each group. The modified formulation is

$$\widehat{W} = \arg\min_{W} \sum_{t=1}^{T} \|Y_t - X_t w_t\|_2^2 + \lambda \left(\sum_{i=1}^{d} (1-\alpha) \left\|w^i\right\|_2 + \alpha \left\|w^i\right\|_1 \right).$$

The addition of an ℓ_1 norm on the vectors will induce sparsity, but this modification does not have a rich theoretical analysis as the standard Group Lasso has. Because in Chapter 4 we will use some of this analysis to give performance bounds on the original algorithm of this thesis, we chose not to use Sparse Group Lasso.

The theoretical results on Group Lasso led us instead to consider another variant, described in [1]. Multi-Task Feature Learning solves a Group Lasso problem, while additionally being able to learn a suitable representation for the features. When run without the feature learning approach, it computes the same solution path as the Group Lasso, but as we will see its path is much longer. In order to obtain a more similar result, we modified MTFL to use Group Lasso penalties, and will introduce the new algorithm in Chapter 5.

3.3 Regularized Approximate Value Iteration

The introduction of regularization is a natural step in Approximate Value Iteration techniques. All the techniques we introduced in Section 2.4.3 rely on some form of representation, and almost all of the methods, especially the most flexible ones, have the risk of overfitting their solution to the data, or be unable to obtain a solution when the data are scarce. We already mentioned that many forms of regularization exist, for example the choice of a subset dictionary for kernel-based

methods using a non-parametric approximator [14]. In this section we will introduce more regularized techniques, that are in some way similar to the technique proposed in this thesis. In particular in Section 3.3.1 we will see how regularization is introduced to Least Squares Temporal Difference problems, and in Section 3.3.2 we will see how introducing regularization to a non-parametric approximator helps producing guarantees for Fitted Value Iteration.

3.3.1 Regularized Policy Iteration

In this thesis, we are interested in methods that use linear regression for their function approximation. In particular, we expect some kind of Least Squares linear regression problem, with or without regularization. Prime candidates for this type of problem are the various versions of Least Squares Temporal Difference used in Approximate Policy Iteration methods based on LSPI, and Fitted Value Iteration methods that uses linear approximators. The main difficulty in introducing regularization to LSPI is that the method has a clear interpretation as the fixed point of the projected Bellman error. As the name implies, the weighted Least Squares projection is essential to the interpretation of the algorithm, and simply replacing a Least Squares problem with another variant such as Lasso or Ridge will not work in a straightforward way. This is for example the approach taken in [34]. LARS-TD [28] is instead an algorithm that takes the approach of building a different projection problem

$$f(w) = \arg\min_u \frac{1}{2}\|\Phi u - (R + \gamma\Phi' w)\|_2^2 + \lambda \|u\|_1 ,$$

and seeks to find the fixed point $\widehat{w} = f(\widehat{w})$. In the original paper it is shown that this does not correspond to a standard Lasso problem, or any clear optimization problem at all, but that using necessary and sufficient conditions for the subgradient of the convex projection problem it is possible to characterize the solution. The authors then derive a modified LARS algorithm to satisfy these conditions and find a solution. The method is interesting for two principal reasons. Because it makes use of a LARS inspired optimization, it is very computationally efficient, in the order of $\mathcal{O}(nds^3)$. The other is that it uses a ℓ_1 based optimization, and therefore is trying to obtain the sample complexity advantage of a sparse solution. In particular, a simpler ℓ_2 regularized version of the same LSTD problem is introduced in the same paper. On the other hand, even considering the fact that this ℓ_2 regularization allows for a closed form solution, and is therefore extremely computationally efficient, the advantages of finding a sparse solution impact the TD algorithm in the same way as Lasso error bounds outperformed Ridge regression when the number of features greatly exceeds the number of samples.

In particular, a theoretical analysis for the finite sample case is provided in [19]. The main bound in the paper is

Theorem 3.3.1 (Performance Guarantees for Lasso-TD [19]). *Let $\{x_i\}_{i=1}^n$ be a trajectory generated by the MDP. Given the true Value function V^π and a linear approximation of it $\Phi\widehat{w}$ obtained with Lasso-TD we assume that the empirical Gram matrix $\frac{1}{n}\Phi^\mathsf{T}\Phi$ satisfies the $(3, S_w)$ assumption. Then for any other vector u that also satisfies the assumption, and for any $\delta > 0$, with probability $1-\delta$, we have*

$$\|V^\pi - \Phi\widehat{w}\|_n \leq \frac{1}{1-\gamma}\inf_u \left[\|V^\pi - \Phi u\|_n + \frac{12\gamma V_{max} L\sqrt{s}}{\psi}\left(\sqrt{\frac{2\log(2d/\delta)}{n}} + \frac{1}{2n}\right)\right]$$

where $\|V^\pi - \Phi w\|_n$ is the empirical norm $\left(\frac{1}{n}\sum_{i=1}^n (V^\pi(x_i) - \Phi(x_i)w)^2\right)^{1/2}$.

Intuitively, the $(3, S_w)$ assumption is a constraint on the ability of the Φ matrix to support a good representation of the target function. We will see a similar assumption, with a longer comment, in Section 4.3.1. Lasso-TD is just the name used for an ℓ_1 regularized LSTD in the theoretical paper, and solves essentially the same fixed point problem of LARS-TD. Therefore we can easily see from the bound that when the assumptions are met LARS-TD can perform well. If we take the squares of both sides, the error grows approximately linearly in s and $\log(d)$. This is the same growth rate that a single Lasso problem has on a regression problem. This relationship derives from the properties of ℓ_1 regularization, and the fact that the proof of Theorem 3.3.1 makes uses of several error bounds designed for Lasso similar to Equation 3.2.1, mostly introduced in [36]. It will be interesting to compare this result with the bounds for the multi-task setting we will introduce in Section 4.4.

A similar approach to LARS-TD is taken in [24]. The idea of finding the fixed point of a regularized projection operator combined with the Bellman operator makes it impossible to formulate the resulting problem as a standard regression problem. The authors in [24] choose instead to consider separately the problem of finding the projection and finding the fixed point of LSTD, and apply different regularization to each operator, an ℓ_2 regularization to the projection and an ℓ_1 regularization to the fixed point problem. The resulting mixed ℓ_2/ℓ_1 problem can be interpreted as a standard Lasso problem, and solved efficiently with any standard method that can solve Lasso.

3.3.2 Regularized Value Iteration

We already mentioned choosing a subset of samples in order to reduce overfitting in kernel modeling. Since kernel models can be expressed as a linear combination of kernels, with Kw being the approximator evaluation, standard regularization techniques on the w vector of weights can be applied. A remarkable result concerning regularization with kernel models is the representer theorem.

Theorem 3.3.2 (Representer Theorem). *Given a Reproducing Kernel Hilbert Space $\mathcal{K}$ based on the Mercer kernel K, a strictly monotonically increasing penalty function*

Ω *and a loss function* L*, the problem*

$$\hat{f} = \arg\min_f \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda\Omega(\|f\|_{\mathcal{K}})$$

admits a solution in the form

$$\hat{f}(x) = \sum_{i=1}^{n} K(x', x_i)w_i.$$

This theorem gives us guarantees on the representation power of kernel modeling, giving a sufficient condition for the kernel model to be able to exactly represent the solution based on the samples provided to the algorithm. Of course since this problem requires regularization it does not imply that any function can be surely represented with kernels, but only a regularized approximation. For a full exposition on kernel methods, the reader can refer to [49].

This regularization problem is used in [17] to compute a regularized function approximation of the Q function in a Fitted Q Iteration framework. The exact problem is defined as

$$Q_{k+1} = \arg\min_{Q\in\mathcal{K}} \frac{1}{n}\sum_{i=1}^{n}\left[r_i + \gamma \max_{a'\in\mathcal{A}} Q_k(x_i', a') - Q(x_i, a_i)\right]^2 + \lambda \|Q\|_{\mathcal{K}}^2 .$$

Introducing the matrices $K : K_{i,j} = K(x_i, a_i, x_j, a_j)$ and $K' : K'_{i,j} = K(x_i', a', x_i, a_i)$ and with the definition of the approximator $Q_i = Kw$ and the norm $\|Q\|_{\mathcal{K}}^2 = w^\mathsf{T} Kw$ the resulting problem is

$$\begin{aligned} w_{k+1} =& \arg\min_w \left\|R + \gamma K' w_k - Kw\right\|_2^2 + \lambda w^\mathsf{T} Kw \\ =& (K + \lambda I)^{-1}(R + \gamma K' w_k). \end{aligned}$$

Using this approximator, the author derives performance guarantees for the algorithm following a similar derivation to [35]. Since in Section 4.2 we will use the same starting point to derive the main result of this thesis, we will just report the performance bound for Kernel Fitted Q Iteration.

$$\begin{aligned} &\|V^* - V^{\pi_K}\|_\rho \leq \\ &\frac{2}{(1-\gamma)^2}\left[\gamma^{K/2}\|V^* - V_0\|_\infty + C\left[c_1\lambda \max_{k=1,\dots,K}\|T_k^* Q_0\|_{\mathcal{K}}^2 + \frac{c_2 V_{max}^4}{n\lambda^{d/l}} + \frac{c_3\log(1/\delta)}{nV_{max}^4}\right]^2\right]. \end{aligned}$$

c_1, c_2, c_3 are universal constants, while the C constant depends on the MDP and will be later defined as the MDP's discounted-average concentrability of future state distribution. The l variable in the bound correspond to a bound on the Lipschitz smoothness of the function. From the bound we can see that the non-parametric algorithm strikes a balance between the representation error induced by the

$\lambda \|T_k^* Q_0\|_{\mathcal{K}}^2$ term and the complexity of the approximator. Optimal rates for λ can be used to optimize the bound, that at this point depends on the original dimensionality d only through complex relationships. This is a clear example of the power of non-parametric approximation, and of its drawbacks in terms of interpretability.

Chapter 4

Sparse Fitted Q Iteration

4.1 Sparse Fitted Q Iteration

The main contribution in this thesis is the development and theoretical analysis of a new regularized Fitted Value Iteration algorithm. In particular our goal is to exploit an assumption of group sparsity in the underlying problem. As we introduced Fitted Q Iteration in Section 2.4.4, we will now detail the proposed algorithm.

4.1.1 Problem Formulation

SFQI is a multi-task Fitted Value Iteration algorithm, where the approximation of the Action-Value function is carried out with a group regularized linear approximator. In particular, we assume we have a set of T tasks, each with its own underlying MDP $\mathcal{M}_t$. All the MDPs share the same continuous features, so that each state space $\mathcal{X}_t$ is a subspace of $\mathbb{R}^d$. We assume all tasks have the same action space $\mathcal{A}$. Most importantly, each task has a different reward space R_t and dynamics $\mathcal{P}_t$. From each task we receive a set of n samples $\{x_i, a_i, x'_i, r_i\}_{i=1\dots n}$, and the goal is to learn a policy π_t for each task in order to maximize the expected sum of the discounted rewards $J_\pi = \mathbb{E}\left[\sum_{i=0}^{n} \gamma^i r_i\right]$. SF$Q$I is a linear F$Q$I algorithm, in the sense that it is assumed that the various Q_k that the algorithm will approximate at each iteration can be represented as a linear combination of the features.

The regression algorithm will minimize a squared loss in order to perform at each step a regression, and compute set of unknown regression parameters $w_t \in \mathbb{R}^d$ for each task. Moreover we expect the number of features d to be much higher than the number of samples n. For this reason a simple OLS Equation (3.1.2) cannot solve the regression problem, and a regularized approach is needed. We introduced in Section 3.3, methods using ℓ_1, ℓ_2 and mixtures of these norms to solve single-task MDPs. One of the goal of SFQI is to exploit instead the information coming from all of the tasks to uncover some kind of structured sparsity in the solution. We expect that the various Q_t functions can be represented as a linear combination not only

of a sparse subset of the d features, but that this subset is shared among all the tasks. In other words, we expect the solution to be sparse in the group sparse sense that we introduced in Section 3.2.3. More in detail, we will compute T separate representation of Q_t functions, each parametrized with a vector w_t. If we merge together the w_t weights as a matrix

$$W = \left[\begin{array}{c|c|c|c} w_{1,1} & w_{1,2} & \dots & w_{1,T} \\ w_{1,1} & w_{1,2} & \dots & w_{1,T} \\ \vdots & \vdots & \dots & \vdots \\ w_{d,1} & w_{d,2} & \dots & w_{d,T} \end{array}\right],$$

we want to penalize the weights as groups corresponding to the same feature, in our case rows w^i of the W matrix. This corresponds to a $l_{2,1}$ penalty, that we intend to use to set to zero most of the weights in the solution. With the choice of a quadratic loss and a $l_{2,1}$ regularization, the optimization problem reduces to a Group Lasso problem.

4.1.2 Algorithm Overview

We can now formally define the SFQI algorithm that is summarized in Algorithm 4.1. Since this is a Fitted Value Iteration, we will follow a similar structure to Algorithm 2.10, but this time we can fully characterize the approximation step. The basic structure of a discrete action FQI algorithm is preserved, but we need to make some small adjustments to take into account the multiple tasks.

Again we will obtain a different approximator for each discrete action $a \in \mathcal{A}$, so if the cardinality of the available actions is $|\mathcal{A}|$, we have $|\mathcal{A}|T$ vectors $w_{a,t}$, divided in W_a matrices. For the regression we merge the datasets $\{\{x_{it}, a_{it}, x'_{it}, r_{it}\}_{i=1\dots n}\}_{1\dots T}$ into matrices and vectors, where each row corresponds to a sample, and obtain $X_{a,t}, X'_{a,t} \in \mathbb{R}^{n_{a,t} \times d}, R_{a,t}, \in \mathbb{R}^{n_a}$. It is clear that we need at least one sample for each action in each task, otherwise the regression itself is not well defined.

When performing linear regression, a common and important detail is the addition of an artificial constant feature to the problem. This permits the regression algorithm to model an offset in the value of the Q function, which is an essential property in real applications. But if we introduce such a constant to our regularized problem, it will then incur in regularization, and this is usually a serious drawback in cases where the value function takes large values, and the weight of the offset needs to be large. An alternative solution, which is preferable in our case, is to subtract its average from the target of the regression, and then add it again when evaluating the function. This has the advantage of not being penalized by the regularization parameter, and is therefore a better choice. For this reason at iteration k we will also store a $B_{a,t,k} \in \mathbb{R}$ bias for each action and task. The algorithm returns the last weights $W_{a,K}, B_{a,t,K}$, that can be used to evaluate the policy in a new state by simple multiplication.

Algorithm 4.1 SFQI

input: $X_{a,t}, X'_{a,t}, R_{a,t}$, λ,γ,tol,K
output: $W_{a,K}, B_{a,t,K}$

```
Initialize W_{a,1} = 0 for all actions, B_{a,t,1} = 0 for all tasks and actions, k = 1
do
    for a ← 1, ..., |A| do
        W_{a,k-1} ← W_{a,k}
        for t ← 1, ..., T do
            B_{a,t,k-1} ← B_{a,t,k}
        end for
    end for
    k ← k + 1
    for t ← 1, ..., T and a ← 1, ..., |A| do
        TQ̂_{a,t} ← R_{a,t} + max_{a'} γ ( X'_{a',t} w_{a',t,k-1} + B_{a',t,k-1} )
        B_{a,t,k} ← (1/n_{a,t}) Σ TQ̂_{a,t}
        TQ̂_{a,t} ← TQ̂_{a,t} − B_{a,t,k}
    end for
    for a ← 1, ..., |A| do
        W_{a,k} ← arg min_W Σ_t ||TQ_{a,t} − X_{a,t} w_t||_2^2 + λ ||W||_{2,1}
    end for
while (max_a ||W_{a,k} − W_{a,k-1}||_2 ≥ tol or max_{a,t} ||B_{a,t,k} − B_{a,t,k-1}||_2 ≥ tol) and k < K
```

From the layout of the algorithm, it is clear that when $T = 1$, this method is equivalent to a Lasso regularized Fitted Q Iteration. We expect therefore SFQI to improve the limits of simply applying Lasso to each separate task. From the bounds in Equation 3.2.1, we know that the simple Lasso regression can recover a good solution when the number of features is exponential in the number of samples. We will show in Section 4.4 that SFQI can surpass this limit. The addition of multiple tasks introduces a novel source of information that can improve the quality of the solution, but this improvement cannot be measured on a single task basis. The motivating idea for SFQI is to improve the average performance across tasks, and does not exclude that a single task will perform slightly worse. This overall improvement in performance is the metric we will use when validating our results in the experimental section.

The choice of a Group Lasso problem as the regularization problem is motivated by recent results on the reconstruction capabilities of this approximation technique. We will introduce these results in Section 4.3, and as we will see, the group sparse assumption plays a central role in the error bound. The introduction of more tasks to SFQI does not guarantee the absence of negative transfer if the new tasks do

not share the same sparsity. To provide a partial solution to this problem, we can maintain the multi-task approach, while trying to expand the group sparsity assumption to a larger class of problems. This modification can be included in SFQI in a straightforward way by substituting the Group Lasso problem with some other multi-task method. In particular we will consider problems that are not sparse, but admit a sparse representation after a linear transformation of the parameters.

Having introduced the algorithm we can now derive some theoretical results on the quality of the policies computed by it. In order to do this we first need to introduce some preliminary results from [42, 35].

4.2 Finite Time Bounds for Fitted Value Iteration

We introduce now some notation and definitions from [42], that will give us the basis to proceed in the derivation of bounds on the policy performance. A space of bounded measurable functions over our state space $\mathcal{X}, B(\mathcal{X})$ is considered, and we are interested in functions bounded by $0 < V_{max} < +\infty$. As we observed in Section 2.1.5, under mild assumptions an optimal policy can be found for an MDP, and when the number of actions is finite, we can always express it as a deterministic policy. A reduced form can merge the continuous transition kernel P and the policy π into another transition kernel $P^\pi(dy|x) = P(dy|x, \pi(x))$. Two new operators are then introduced, a right-linear operator $P^\pi : B(\mathcal{X}) \to B(\mathcal{X})$ and a left-linear operator over the space $M(\mathcal{X})$ introduced in Section 2.1.2, $P^\pi : M(\mathcal{X}) \to M(\mathcal{X})$

$$(P^\pi V)(x) = \int_y V(y) P^\pi(dy|x),$$
$$(\mu P^\pi)(dy) = \int_x P^\pi(dy|x)\mu(dx).$$

Intuitively the first operator corresponds to computing the expected value of V after executing one step under policy π, while the second operator filters a distribution over states, for example the starting distribution, trough a single step of the policy. An operator resulting from a chain of transitions can also be formulated

$$(P^{\pi_1} P^{\pi_2})(dz|x) = \int_y P^{\pi_2}(dz|y) P^{\pi_1}(dy|x)$$

where the policy π is allowed to change between steps. It is important to note that the P^π operator allows us to express the T^π operator concisely,

$$(T^\pi V)(x) = R(x, \pi(x)) + \gamma(P^\pi V)(x). \tag{4.2.1}$$

We can now introduce the main assumptions used in [42], the original paper.

4.2.1 Assumptions

First we introduce some regularity assumptions on the MDP itself:

Assumption 4.2.1. *$\mathcal{X}$ is a bounded, closed subset of some Euclidean space.*

To introduce the next assumption the $\|\cdot\|_{p,\mu}$ of a function is defined as $\|f\|_{p,\mu}^p = \int |f(x)|^p \mu(dx)$. In the course of the proof we will use this norm to bound the error introduced at each step by approximations, and its propagation through the steps of the MDP. If $p = \infty$ we recover the known supremum-norm bounds [16], but, with pointwise comparisons, better bounds on the error can be derived. For this reason an assumption is needed that limits how much a sequence of steps of the MDP under a sequence of policies π_k can change a distribution ρ w.r.t. a reference distribution μ. To do this an assumption is imposed on the Radon-Nikodym derivative of the sequence of distributions generated by the application of P^{π_k} to ρ w.r.t. μ.

Assumption 4.2.2 (Discounted-average concentrability of future state distributions)**.** *Given ρ, μ and an arbitrary sequence of policies $\pi_k{}_1^K$, we assume that the distribution $\rho P^{\pi_1} P^{\pi_2} \dots P^{\pi_K}$ is absolutely continuous w.r.t. μ. Then we assume that*

$$c(K) \stackrel{\text{def}}{=} \sup_{\pi_1 \dots \pi_K} \left\| \frac{d\left(\rho P^{\pi_1} P^{\pi_2} \dots P^{\pi_K}\right)}{d\mu} \right\|_\infty \tag{4.2.2}$$

satisfies

$$C_{\rho,\mu} \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{m \geq 1} m \gamma^{m-1} c(m) < +\infty.$$

Assumption 4.2.2 is a limit on the growth of the deviation of the reference distribution at each step. That $c(m)$ grows when m increases is acceptable, as long as the geometric discount rate of γ can keep these deviations under control, resulting in a finite $C_{\rho,\mu}$. Several interpretations can be given for this coefficient, and in [42] the proposed one is a comparison with the top-Lyapunov exponent of the MDP. If the exponent is positive the trajectories of the MDP are sensible to perturbations, while if the exponent is negative the system is stable. They prove that when the exponent is negative then $C_{\rho,\mu}$ is finite.

4.2.2 Policy Performance Guarantees

We can now describe the main results needed from [42]. The problem considered is the estimation of the final performance of a policy π_K obtained after K iterations of Approximate FVI, in comparison with the optimal policy. To measure the performance, the optimal value function V^* is compared with the value function obtained, under a desired norm. That is, the quantity $\|V^* - V^{\pi_K}\|_{p,\mu}$ is bounded. At every step of the iteration a policy π_k greedy w.r.t. V_k is built. The proof begins by deriving bounds on $V^* - V^{\pi_K}$.

Theorem 4.2.3.

$$V^* - V^{\pi_K} \leq \gamma(I - \gamma P^{\pi_K})^{-1} \left\{ \sum_{k=0}^{K-1} \gamma^{K-k} \left[(P^{\pi^*})^{K-k} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right] |\varepsilon_k| \right.$$
$$\left. + \gamma^{K+1} \left[(P^{\pi^*})^{K+1} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_0} \right] |V^* - V_0| \right\}. \quad (4.2.3)$$

Proof. $T^* V_k - T^{\pi^*} V_k \geq 0$ since at the k-th step the value function might not have converged to its true values yet, so choosing the real optimal action might not coincide with the action selected by the optimal Bellman operator. At each step a function approximation is performed, and this operation introduces an approximation error. Therefore we cannot substitute $V_{k+1} = T^* V_k$ but need to introduce an error function ε_k such that $V_{k+1} = T^*_{V_k} + \varepsilon_k$. With this basic substitution, the Bellman equation (4.2.1) we can derive

$$V^* - V_{k+1} = T^{\pi^*} V^* - T^* V_k + \varepsilon_k = T^{\pi^*} V^* - T^{\pi^*} V_k + T^{\pi^*} V_k - T^* V_k + \varepsilon_k$$
$$\leq R + \gamma P^{\pi^*} V^* - R - \gamma P^{\pi^*} V_k + \varepsilon_k = \gamma P^{\pi^*}(V^* - V_k) + \varepsilon_k.$$

By induction on the sequence V_k

$$V^* - V_K \leq \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi^*})^{K-k-1} \varepsilon_k + \gamma^K (P^{\pi^*})^K (V^* - V_0). \quad (4.2.4)$$

Since π_k is the greedy policy w.r.t. V_k, we have $T^* V_k = T^{\pi_k} V_k$, moreover due to the optimality of $V^*, T^* V^* - T^{\pi_k} V^* \geq 0$, and we can derive

$$V^* - V_{k+1} = T^* V^* - T^* V_k + \varepsilon_k = T^* V^* - T^{\pi_k} V^* + T^{\pi_k} V^* - T^* V_k + \varepsilon_k$$
$$\geq R + \gamma P^{\pi_k} V^* - R - \gamma P^{\pi_k} V_k + \varepsilon_k = \gamma P^{\pi_k}(V^* - V_k) + \varepsilon_k.$$

Again by induction

$$V^* - V_K \geq \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_{k+1}}) \varepsilon_k + \gamma^K (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_0})(V^* - V_0). \quad (4.2.5)$$

Lastly, from all the previous inequalities and $T^* V_k - T^{\pi^*} V_k \geq 0$ we have

$$V^* - V_K = T^{\pi^*} V^* - T^{\pi^*} V_K + T^{\pi^*} V_K - T^* V_K + T^{\pi_K} V_K + T^{\pi_K} V^{\pi_K}$$
$$\leq \gamma P^{\pi^*}(V^* - V_K) + \gamma P^{\pi_K}(V_K - V^* + V^* - V^{\pi_K}).$$
$$(I - \gamma P^{\pi_K})(V^* - V^{\pi_K}) \leq \gamma (P^{\pi^*} - P^{\pi_K})(V^* - V_K).$$

The operator $(I - \gamma P^{\pi_K})^{-1}$ can be expressed as $\sum_{m \geq 0} \gamma^m (P^{\pi_K})^m$, and is therefore invertible and monotonic, so with a multiplication by $(I - \gamma P^{\pi_K})^{-1}$ we preserve the inequality. We obtain

$$V^* - V^{\pi_K} \leq \gamma (I - \gamma P^{\pi_K})^{-1} P^{\pi^*} (V^* - V_K)$$
$$- \gamma (I - \gamma P^{\pi_K})^{-1} P^{\pi_K} (V^* - V_K).$$

Using Equation (4.2.4) and Equation (4.2.5), and taking the absolute value of both sides we obtain Equation (4.2.3). □

We can now describe the main result of [42].

Theorem 4.2.4. *For any number of steps K, and under Assumption 4.2.2, if at each step the approximation error $|\widehat{V} - T^*V| < \epsilon$, then we have*

$$\|V^* - V^{\pi_K}\|_{2,\rho}^2 \leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \left[\frac{1}{1-\gamma^{K+1}} C_{\mu,\rho}\epsilon^2 + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}(2V_{max})^2\right]. \tag{4.2.6}$$

Proof. We begin the proof by introducing some shorter notation, to rewrite Equation (4.2.3) in a simpler form.

$$V^* - V^{\pi_K} \leq \frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\left[\sum_{k=0}^{K-1} \alpha_k A_k |\varepsilon_k| + \alpha_K A_K |V^* - V^0|\right],$$

with

$$\alpha_k = \frac{(1-\gamma)\gamma^{K-k-1}}{1-\gamma^{K+1}}, \text{ for } 0 \leq k < K, \text{ and } \alpha_K = \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}},$$
$$A_k = \frac{1-\gamma}{2}(I - \gamma P^{\pi_K})^{-1}\left[(P^{\pi^*})^{K-k} + P^{\pi_K}P^{\pi_{K-1}}\dots P^{\pi_{k+1}}\right], \text{ for } 0 \leq k < K,$$
$$A_K = \frac{1-\gamma}{2}(I - \gamma P^{\pi_K})^{-1}\left[(P^{\pi^*})^{K+1} + P^{\pi_K}P^{\pi_{K-1}}\dots P^{\pi_0}\right].$$

We have that

$$\begin{aligned}
&\|V^* - V^{\pi_K}\|_{2,\rho}^2 = \int \rho(dx)|V^*(x) - V^{\pi_K}(x)|^2 \\
&\leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \int \rho(dx)\left[\sum_{k=0}^{K-1} \alpha_k A_k|\varepsilon_k| + \alpha_k A_k|V^* - V^0|\right]^2(x) \\
&\leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \int \rho(dx)\left[\sum_{k=0}^{K-1} \alpha_k A_k|\varepsilon_k|^2 + \alpha_k A_k|V^* - V^0|^2\right](x),
\end{aligned}$$

where the first inequality follows from Theorem 4.2.3, and the second is an application of Jensen's Inequality applied to the convex function $(\sum x_i)^2$. We can bound $|V^* - V^0| \leq 2V^{max}$ and under Assumption 4.2.2 we have $\rho A_k \leq (1-\gamma)\sum_{m\geq 0}\gamma^m c(m + K - k)\mu$ and obtain

$$\begin{aligned}
&\|V^* - V^{\pi_K}\|_{2,\rho}^2 \\
&\leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \left[\sum_{k=0}^{K-1} \alpha_k(1-\gamma)\sum_{m\geq 0}\gamma^m c(m+K-k)\|\varepsilon_k\|_{2,\mu}^2 + \alpha_k(2V_{max})^2\right].
\end{aligned}$$

By the substitution of α, the definition $C_{\mu,\rho} = (2-\gamma)^2 \sum_{m\geq 1} m\gamma^{m-1}c(m)$ and taking $\epsilon = \max_k \|\varepsilon_k\|_{p,\mu}^p$ we obtain Theorem 4.2.4

$$\begin{aligned}\|V^* - V^{\pi_K}\|_{2,\rho}^2 &\leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \left[\frac{(1-\gamma)^2}{1-\gamma^{K+1}}\right. \\ &\qquad \left. \sum_{k=0}^{K-1}\sum_{m\geq 0} \gamma^{m+K-k-1} c(m+K-k)\|\varepsilon_k\|_{2,\mu}^2 + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}(2V_{max})^2\right] \\ &\leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2}\right]^2 \left[\frac{1}{1-\gamma^{K+1}} C_{\mu,\rho}\epsilon^p + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}}(2V_{max})^2\right].\end{aligned}$$

□

This result gives us insight on the performance of Fitted Value Iteration. The concentrability coefficient $C_{\rho,\mu}$ acts as a filter between the performance loss, measured according to ρ, and the error, measured according to μ, and expresses the intensity of the propagation of the error across iterations. The second term, depending on V_{max}, quickly reduces with the number of iterations K, and can be made small, at the risk of raising the maximum ϵ at some iteration. In order to quantify this error in term of number of samples, we will introduce in the next section further assumption on the V_k functions, and exploit them to obtain new bounds.

4.3 Oracle Inequalities under Group Sparsity

Before proceeding with the main result of this chapter, we must introduce another prerequisite result from [35]. The main assumption of this work is an underlying group sparsity in the functions that need to be approximated at each time step. Such a structure can be exploited with the Group Lasso regularization as explained in Section 4.1.1. Several guarantees were derived for this class of regularized problems, and we will now repeat the ones useful for our derivation, and adapt them to our notation. For the full proof, refer to the original paper.

4.3.1 Assumptions

We consider the problem of simultaneously perform linear regression on T tasks, indexed by $t \in \mathbb{N}_T$. Each task has a set on n samples, organized in a matrix X_t, where each row x_t^i corresponds to a sample. We consider the case when the design matrix X_t is deterministic, but as the authors mention, an extension to random matrices follows standard procedures. Without loss of generality, we assume that each column of the X_t matrix, corresponding to a particular feature, is normalized so that $(1/n)\,\mathrm{Diag}\left(X_t^\mathsf{T} X_t\right) = \mathbf{I}$. We construct a larger, block-diagonal matrix $X \in \mathbb{R}^{Tn\times Td}$ out of the X_t matrices, which will be the design matrix in our regression problem. The unknown regression parameters are T d-dimensional vectors, w_t, that

we stack together in a $w \in \mathbb{R}^{Td}$ vector. As in Section 3.1.2 we assume Gaussian, i.i.d noise with distribution $\mathcal{N}\left(0, \sigma^2\right)$. In our multi-task setting, the variables across tasks corresponding to the same feature are grouped together, and we obtain a partition of $\mathbb{N}_{Td}$ into d separate groups. Given a subset of these groups $J \subseteq \mathbb{N}_d$ we introduce $w_J : (w_t I\{t \in J\})$, where I is the indicator function. Finally we introduce $J(w) = \{t : w_t \neq 0\}$. We can now formulate the main assumption needed to guarantee precision in the reconstruction of the original sparse vector w^*.

Assumption 4.3.1 (Restricted Eigenvalues (RE) Assumption)**.** *There exists a positive number $\kappa_{Td} = \kappa_{Td}(s)$ such that:*

$$\min \left\{ \frac{\|X\Delta\|_2}{\sqrt{n}\,\|\Delta_J\|_2} : |J| \leq s, \Delta \in \mathbb{R}^{Td} \backslash \{0\}, \|\Delta_{J^c}\|_{2,1} \leq 3\,\|\Delta_J\|_{2,1} \right\} \geq \kappa_{Td}, \quad (4.3.1)$$

where J^c denotes the complement of the set of indices J.

Intuitively, this assumption gives us a weak constraint on the representation capability of the data. In an Ordinary Least Squares problem, the rank of the matrix $X^\mathsf{T}X$ must be strictly greater than 0. This can be expressed also as $\|X\Delta\|_2 > 0$, because the minimum quantity that this norm can take is equal to the norm of the vector Δ multiplied by the smallest singular value of X. In a Group Lasso setting, the number of features d is usually much larger than the number of samples, and the matrix X surely has some zero values. This assumption forces a much weaker assumption using a method similar to the definition of a matrix norm. If we refer to Appendix A, we can see that the condition $\frac{\|X\Delta\|_2}{\sqrt{n}\|\Delta_J\|_2}$ is similar to an induced norm, with the difference that the denominator has only the Δ_J. This vector is composed only by the non-zero groups of variable, and intuitively this norm will be larger than the smallest eigenvalue of the part of the matrix X related to the non-zero groups. κ_{Td} is therefore a lower bound on the capability of the matrix X to represent a solution not for the full OLS problem, but only for the sparse subset that compose the true solution.

Verifying whether this assumption holds on the actual data is not straightforward, but a number of sufficient conditions for its satisfaction exists. The positivity of the minimal eigenvalue of $X^\mathsf{T}X$ is of course usually not suitable. Many assumptions can guarantee the RE assumption, alternatives are proposed in [18] and [5].

4.3.2 Approximation error Bounds

We can now introduce the main result in the original paper.

Theorem 4.3.2. *Considering the multi-task setting defined in Section 2.5, with $d \geq 2$ and $T, n \geq 1$. Set*

$$\lambda = \frac{2\sqrt{2}\sigma}{\sqrt{nT}} \left(1 + \frac{\delta \log d}{T}\right)^{\frac{1}{2}},$$

where $\delta > \frac{5}{2}$, and set ϕ as the largest eigenvalue of $X^\mathsf{T}X$. Given $|J(w)|$ as the cardinality of $J(w)$, if $|J(w^*)| < s$ and Assumption 4.3.1 holds with $\kappa_{Td} = \kappa_{Td}(s)$, then with probability at least $1 - 2d^{1-2\delta/5}$

$$|J(\widehat{w})| \leq \frac{64\phi}{\kappa_{Td}^2} s \tag{4.3.2}$$

If in addition $\kappa_{Td}(2s) > 0$, then with the same probability

$$\frac{1}{T}\sum_{t=1}^{T} \|w_t^* - \widehat{w}_t\|_2^2 = \frac{1}{T}\|w^* - \widehat{w}\|_2^2 \leq \frac{1280\sigma}{\kappa_{Td}^4(2s)} \frac{s}{n}\left(1 + \frac{\delta \log d}{T}\right). \tag{4.3.3}$$

This theorem provides us with insight on the main factors that we need to consider when building our approximations. From the bound we can clearly see a direct dependence of the error on the number of nonzero features in the sparse representation. This is expected, because, fixed everything else, if we need to estimate more parameters we need additional information. This is clearly balanced by the number of samples at the denominator. The κ constant plays a scaling role, where better conditioned X matrices will result in larger constant, and therefore smaller bound. A similar but inverse role is played by the variance of the noise, that can increase the error. Both these quantities are difficult to estimate, and even more difficult to control. Ultimately when the sparsity assumptions that are at the basis of SFQI are met, they are just given constants.

The most important part of the bound is the $\frac{\delta \log d}{T}$. As expected, if we want to increase our confidence, the error bound worsen. But what is most interesting is the comparison of this bound with the equivalent result for LASSO, which to us is just Group Lasso with $T = 1$. As we previously mentioned in Equation (3.2.1), LASSO scales in the number of true features s and in the logarithm of total features $\log(d)$. This bound on Group Lasso instead tells us that as long as the number of tasks remains in the same order of magnitude as $\log(d)$, our error will remain more or less constant. Of course introducing a new task means introducing n new samples to the overall problem, and when samples are scarce this might be a problem. But as long as we keep adding tasks, we can exponentially increase the number of features without incurring the same problems that LASSO faces. One last remark is that although introducing new tasks can compensate the addition of more features, we must not forget that introducing a new task might increase the s factor, if the new task does not share the same sparsity pattern as the rest of the group. This limitation is what we seek to alleviate by learning a sparse representation, as we will see in Chapter 5.

4.4 Finite Time Bounds for Fitted Q Iteration under Group Sparsity

We can now state the main result of this thesis regarding the optimality of the policies obtained after K steps of Fitted Q Iteration with group regularization. We will

modify the results from Section 4.2 to use the Q action-value function instead of the V value function, and then derive some bound on the overall error committed at each step by all tasks using Equation 4.3.3. This will give us an explicit formula in terms of n, d and T that can help us guide when making choices in practical applications.

4.4.1 Performance Evaluation

As we mentioned in Section 2.1.5, and again in Section 4.1.1, the goal of a reinforcement learning algorithm is to maximize the reward collected during its episodes, taking into account a discount factor or an averaging constant if it is necessary. This is well defined by the J_π function and the various value functions, and is commonly used as a measure of the quality of the policy. In a multi-task setting this simple notion becomes more complex, because we are no longer evaluating a single task, but need to take into account the performance of all the task simultaneously. We do not expect SFQI to obtain an excellent performance in each of his tasks, since a single task might be significantly harder than the others, or the scarcity of samples for that task may be too strong for the group optimization to compensate. What we hope to improve instead, is the average performance across all the tasks, to show that the multi-task approach as an advantage on solving each task separately. Concretely, what we will evaluate is the average distance from the optimal action-value function Q_t^* of the approximated value functions Q_t

$$\frac{1}{T}\sum_{t=1}^{T} \|Q_t^* - Q_t^{\pi_K}\|_{2,\mu}^2 .$$

4.4.2 Finite Time Average Bounds for SFQI

We will follow the same steps as in Section 4.2.2. The greatest difference is that we are now working with state-action functions and not with state functions. Our definition of the μ probability distributions and the linear operators P^π need to change to reflect this. First, we will define

$$\mu_\pi(x,a) \stackrel{\text{def}}{=} \mu(x)\pi(x,a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}.$$

The reason for this definition is that once we define the $\|\cdot\|_{p,\mu_\pi}$ of a function over $\mathcal{X} \times \mathcal{A}$ as

$$\|f\|_{p,\mu_\pi}^p = \sum_a \int_x |f(x,a)|^p \mu_\pi(dx,a),$$

then we have that

$$\|V^\pi\|_\mu = \|Q^\pi\|_{\mu_\pi} .$$

That is, in the shift from V functions to Q functions we need not only a reference distribution over states, but a reference policy. We can then introduce two new

operators, a right-linear operator $P^\pi : B(\mathcal{X} \times \mathcal{A}) \to B(\mathcal{X} \times \mathcal{A})$ and a left-linear operator $P^\pi : S(\mathcal{X} \times \mathcal{A}) \to S(\mathcal{X} \times \mathcal{A})$

$$(P^{\pi_1} Q)(x,a) = \int_y P(dy|x,a) \sum_{a'} \pi_1(y,a') Q(y,a'),$$
$$(\mu_\pi P^{\pi_1})(dy,a') = \int_x \sum_a \mu_\pi(dx,a) P(dy|x,a)\, \pi_1(y,a').$$

Again, intuitively the first operator corresponds to computing the expected value of Q after executing one step under policy π_1, while the second operator generates a new μ_π composed by a new μ, filtered through a single step of the model, in reference with the new policy π_1. We can easily introduce also the operators resulting from a chain of transitions

$$(P^{\pi_1} P^{\pi_2} Q)(x,a) = \int_{y,z} P(dy|x,a)\, P(dz|y,\pi_1(y))\, Q(z,\pi_2(z)),$$
$$(\mu_\pi P^{\pi_1} P^{\pi_2})(dz,a') = \int_{x,y} \sum_a \mu_\pi(dx,a) P(dy|x,a)\, P(dz|y,\pi_1(y))\, \pi_2(z,a')$$

where we allowed the policy π_1, π_2 to change between steps. It is straightforward now to adapt Assumption 4.2.2

Assumption 4.4.1 (Discounted-average concentrability of future state distributions w.r.t. a policy)**.** *Given ρ_π, μ_π and an arbitrary sequence of policies $\{\pi_k\}_1^K$, we assume that the distribution $\rho P^{\pi_1} P^{\pi_2} \dots P^{\pi_K}$ is absolutely continuous w.r.t. μ. Then we assume that*

$$c(K) \stackrel{\text{def}}{=} \sup_\pi \sup_{\pi_1 \dots \pi_K} \left\| \frac{d(\rho_\pi P^{\pi_1} P^{\pi_2} \dots P^{\pi_K})}{d\mu_\pi} \right\|_\infty \tag{4.4.1}$$

satisfies

$$C_{\rho_\pi, \mu_\pi} \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{m \geq 1} m \gamma^{m-1} c(m) < +\infty.$$

Again this is a constraint on the growth rate of the ρ distribution's deviation after several steps of various policies. This time we evaluate a distribution in relation to a policy, so we have to take the maximum deviation over all possible policies to continue to have an upper bound. With the new definition of the P^π operator, we can again express the T^π operator,

$$(T^\pi Q)(x,a) = R(x,a) + \gamma (P^\pi Q)(x,a). \tag{4.4.2}$$

With this, Equation (4.2.4) and Equation (4.2.5) can be easily adapted to hold

$$Q^* - Q_K \leq \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi^*})^{K-k-1} \varepsilon_k + \gamma^K (P^{\pi^*})^K (Q^* - Q_0), \tag{4.4.3}$$

$$Q^* - Q_K \geq \sum_{k=0}^{K-1} \gamma^{K-k-1} (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_{k+1}}) \varepsilon_k + \gamma^K (P^{\pi_{K-1}} P^{\pi_{K-2}} \dots P^{\pi_0})(Q^* - Q_0). \tag{4.4.4}$$

and we obtain again the result of Theorem 4.2.3

$$Q^* - Q^{\pi_K} \leq \gamma(I - \gamma P^{\pi_K})^{-1} \left\{ \sum_{k=0}^{K-1} \gamma^{K-k} \left[(P^{\pi^*})^{K-k} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_{k+1}} \right] |\varepsilon_k| \right. \\ \left. + \gamma^{K+1} \left[(P^{\pi^*})^{K+1} + P^{\pi_K} P^{\pi_{K-1}} \dots P^{\pi_0} \right] |Q^* - Q_0| \right\}. \quad (4.4.5)$$

We are now ready to prove the main result of this chapter.

Theorem 4.4.2 (Finite Time Average Bounds for SFQI)**.** *For any number of steps K, and under Assumption 4.4.1 and Assumption 4.3.1, then with probability at least $(1 - 2d^{1-2\delta/5})^K$ we have*

$$\frac{1}{T} \|Q^* - Q^{\pi_K}\|_{2,\rho_\pi}^2 \\ \leq \left[\frac{2\gamma(1-\gamma^{K+1})}{(1-\gamma)^2} \right]^2 \left[\frac{1}{1-\gamma^{K+1}} C_{\mu_\pi,\rho_\pi} C \frac{1280\sigma}{\kappa_{Td}^4(2s)} \frac{s}{n} \left(1 + \frac{\delta \log d}{T} \right) + \frac{(1-\gamma)\gamma^K}{1-\gamma^{K+1}} (2Q_{max})^2 \right]. \quad (4.4.6)$$

Proof. We begin by reducing the notation, but this time we have to take into account multiple tasks.

$$\frac{1}{T} \sum_{t=1}^{T} Q_t^* - Q_t^{\pi_K} \leq \frac{1}{T} \sum_{t=1}^{T} \frac{2\gamma_t(1-\gamma_t^{K+1})}{(1-\gamma_t)^2} \left[\sum_{k=0}^{K-1} \alpha_{tk} A_{tk} |\varepsilon_{tk}| + \alpha_{tK} A_{tK} |Q_t^* - Q_t^0| \right],$$

with

$$\alpha_{tk} = \frac{(1-\gamma_t)\gamma_t^{K-k-1}}{1-\gamma_t^{K+1}}, \text{ for } 0 \leq k < K, \text{ and } \alpha_{tK} = \frac{(1-\gamma_t)\gamma_t^K}{1-\gamma_t^{K+1}},$$
$$A_{tk} = \frac{1-\gamma_t}{2}(I - \gamma_t P_t^{\pi_K})^{-1} \left[(P_t^{\pi^*})^{K-k} + P_t^{\pi_K} P_t^{\pi_{K-1}} \dots P_t^{\pi_{k+1}} \right], \text{ for } 0 \leq k < K,$$
$$A_{tK} = \frac{1-\gamma_t}{2}(I - \gamma_t P_t^{\pi_K})^{-1} \left[(P_t^{\pi^*})^{K+1} + P_t^{\pi_K} P_t^{\pi_{K-1}} \dots P_t^{\pi_0} \right].$$

We can then write

$$\frac{1}{T} \sum_{t=1}^{T} \|Q^* - Q^{\pi_K}\|_{2,\rho_\pi}^2 = \frac{1}{T} \sum_{t=1}^{T} \sum_a \int_x \rho_\pi(dx,a) |Q^*(x,a) - Q^{\pi_K}(x,a)|^2$$
$$\leq \frac{1}{T} \sum_{t=1}^{T} \sum_a \left[\frac{2\gamma_t(1-\gamma_t^{K+1})}{(1-\gamma_t)^2} \right]^2 \int_x \rho_\pi(dx,a) \left[\sum_{k=0}^{K-1} \alpha_{tk} A_{tk} |\varepsilon_{tk}| + \alpha_{tK} A_{tK} |Q_t^* - Q_t^0| \right]^2 (x,a)$$
$$\leq \frac{1}{T} \sum_{t=1}^{T} \sum_a \left[\frac{2\gamma_t(1-\gamma_t^{K+1})}{(1-\gamma_t)^2} \right]^2 \int_x \rho_\pi(dx,a) \left[\sum_{k=0}^{K-1} \alpha_{tk} A_{tk} |\varepsilon_{tk}|^2 + \alpha_{tK} A_{tK} |Q_t^* - Q_t^0|^2 \right] (x,a).$$

We can bound $|Q_t^* - Q_t^0| \leq 2Q_t^{max}$ and under Assumption 4.4.1 we have $\rho_\pi A_{tk} \leq (1-\gamma_t)\sum_{m\geq 0}\gamma_t^m c_t(m+K-k)\mu_\pi$ and obtain

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\|Q^* - Q^{\pi_K}\|_{2,\rho_\pi}^2 \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\left[\frac{2\gamma_t(1-\gamma_t^{K+1})}{(1-\gamma_t)^2}\right]^2\left[\sum_{k=0}^{K-1}\alpha_{tk}(1-\gamma_t)\sum_{m\geq 0}\gamma_t^m c_t(m+K-k)\right. \\
&\qquad\left.\int_x\sum_a \mu_\pi(dx,a)|\varepsilon_{tk}(x,a)|^2 + \alpha_{tK}(2Q_{max})^2\right].
\end{aligned}$$

And again using the definition of discounted average concentrability C_{ρ_π,μ_π}, we introduce separate constants for each task C_{t,ρ_π,μ_π}

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\|Q^* - Q^{\pi_K}\|_{2,\rho_\pi}^2 \\
&\leq \frac{1}{T}\sum_{t=1}^{T}\left[\frac{2\gamma_t(1-\gamma_t^{K+1})}{(1-\gamma_t)^2}\right]^2\left[\frac{1}{1-\gamma_t^{K+1}}C_{t,\mu_\pi,\rho_\pi}\right. \\
&\qquad\left.\int_x\sum_a \mu_\pi(dx,a)|\varepsilon_{tk}(x,a)|^2 + \frac{(1-\gamma_t)\gamma_t^K}{1-\gamma^{K+1}}(2Q_{max})^2\right].
\end{aligned} \tag{4.4.7}$$

If we could push in the summation over t, we could now exploit the group sparsity assumption to finally bound the error across all tasks, but we cannot do so due to the $\gamma, C_{\rho_\pi,\mu_\pi}$ terms that are task dependent. In this thesis we choose to simply take the maximum across all tasks of these terms, to achieve a simpler bound. If the task are similarly hard, then this is a reasonable choice. If one of the tasks has much worse constants than the others, then it would be better to build some kind of weighted average bound, and this is an interesting possibility for future development. After taking the maximum we can push in the summation, and we just need to bound $\int_x\sum_a \mu_\pi(dx,a)|\varepsilon_{tk}(x,a)|^2$. We rewrite it as $\int_x\sum_a \mu_\pi(dx,a)\|\varepsilon_{tk}\|_2^2$ and exploiting Equation (4.3.3) we obtain

$$\begin{aligned}
\int_x\sum_a \mu_\pi(dx,a)\|\varepsilon_{tk}\|_2^2 &= \sum_a\int_x \mu_\pi(dx,a)\left\|x^\mathsf{T}(w_{atk}^* - \widehat{w}_{atk})\right\|_2^2 \\
&\leq \max_a\int_x \mu(dx)\left\|x^\mathsf{T}(w_{atk}^* - \widehat{w}_{atk})\right\|_2^2.
\end{aligned}$$

We will now remove the subscript a, k since they are well defined, and introduce the

subscript i to indicate the i-th component of a vector

$$\int_x \mu(dx) \left\| x^\mathsf{T}(w_t^* - \widehat{w}_t) \right\|_2^2 = \int_x \mu(dx) \left(\left(\sum_{i=1}^d (x_i(w_{ti}^* - \widehat{w}_{ti}))^2 \right)^{1/2} \right)^2$$

$$= \int_x \mu(dx) \sum_{i=1}^d (x_i)^2 (w_{ti}^* - \widehat{w}_{ti})^2 \leq \int_x \mu(dx) \left(\sum_{i=1}^d x_i^4 \right)^{\frac{1}{2}} \left(\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 \right)^{\frac{1}{2}}$$

$$= C \left(\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 \right)^{\frac{1}{2}}.$$

Where we upper bounded $\int_x \mu(dx) \left(\sum_{i=1}^d x_i^4 \right)^{\frac{1}{2}}$ with a constant C. We will discuss more later why this maximization is sensible. We now continue our proof with

$$\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 \leq \sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 + \sum_{i=1}^d \sum_{j=1 \neq i}^d (w_{ti}^* - \widehat{w}_{ti})^2 (w_{tj}^* - \widehat{w_{tj}})^2 = \left(\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^2 \right)^2.$$

Therefore

$$\left(\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 \right)^{\frac{1}{2}} \leq \sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^2 = \|w_t^* - \widehat{w}_t\|_2^2 .$$

Putting it all together and using Equation (4.3.3)

$$\frac{1}{T} \sum_{t=1}^T \int_x \sum_a \mu_\pi(dx, a) \|\varepsilon_{tk}\|_2^2 \leq \frac{1}{T} \sum_{t=1}^T C \left(\sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^4 \right)^{\frac{1}{2}}$$

$$\leq C \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^d (w_{ti}^* - \widehat{w}_{ti})^2 = C \frac{1}{T} \sum_{t=1}^T \|w_t^* - \widehat{w}_t\|_2^2$$

$$\leq C \frac{1280\sigma}{\kappa_{Td}^4(2s)} \frac{s}{n} \left(1 + \frac{\delta \log d}{T} \right).$$

We can plug this into Equation (4.4.7) to obtain Equation (4.4.6) □

4.4.3 Remarks on SFQI

We will now proceed with a small discussion of the main implications of the derivation of Equation (4.4.6).

First we clarify the assumption we introduced when we bounded $\int_x \mu(dx) \left(\sum_{i=1}^d x_i^4 \right)^{\frac{1}{2}}$. The main advantage of SFQI over a regular FQI algorithm is the possibility of having

a number of features d that is exponential in the number of samples n. This is reflected in the final bound, where the dependence in the number of samples is logarithmic in d, but linear in C. There is a not so intuitive relationship between d and C, in the fact that as d grows, the number of members in the summation over d that makes up C grows. If the feature that are added are not carefully chosen, this might imply that C grows linearly with d, defeating the purpose of SFQI. In real applications however the features can be manipulated easily to maintain the C number small everywhere, or at least in the areas of interest spanned by $\mu(dx)$. A clear example is with discretization, where only a single feature, or a small subset of soft discretized features, are active at the same time.

From the main result we can see that, as long as the initial hypotheses that each of the $Q_{tk}(x) = x^\mathsf{T} w^*_{tk}$ is satisfied, then our ability of reconstructing the original vector does not depend on the shape of Q_{tk} itself. Concretely, this means that our λ could be set according only to the samples at our disposal, and not depending on the particular iteration the approximation is in. But the main advantage of SFQI to other, non multi-task methods, is surely the independence of the quality of the solution from the number of dimension d, as long as the number of useful tasks at our disposal grows linearly as d grows exponentially. This lets us overcome the limit of standard Lasso techniques, where the maximum number of features that can be introduced depend on the number of samples available for each task.

Lastly, another subtle interaction between this bound and the sample size comes from the s factor. This factor quantifies the underlying real group sparsity, and the number of samples needed grows linearly with s. This is reasonable since s represent the true dimensionality of the problem. This also impose us constraints on the addition of new tasks, since each new task added can in the worst case increase s linearly, and therefore be detrimental to the learning process. One of the motivating factor of the next chapter is exactly this problem. Since the number of samples necessary grows linearly with the minimum lower dimensional representation, it might be useful to learn such a lower dimensional representation, while at the same time maintaining some similarities with the Group Lasso penalty. In Chapter 5 we will introduce a convex multi-task feature learning algorithm similar to Group Lasso, and modifications to it to utilize the same penalties of Group Lasso in order to have an exact extension of Group Lasso, that could provide theoretical guarantees when used as a Group Lasso, and practical performance when solving the extended problem.

As a comparison of our method with other regularized RL techniques, we can easily see that in the case when $T = 1$, the bound reduces to something extremely similar to the one described in Theorem 3.3.1. This is a logical consequence because in our formulation of Group Lasso problem, a single task corresponds to a Lasso problem, and a similar bound appears. It is interesting to notice that our bound holds for any reference distribution that satisfies Assumption 4.4.1, so we are not

limited to the empirical norm but have a generalization. This is at the expense of using a stronger assumption on the data, Assumption 4.3.1, than the one used in Theorem 3.3.1.

The other Fitted Value Iteration algorithm that we presented in Section 3.3.2 uses a non-parametric approximator, so a direct comparison is hard to formulate. One advantage of the non-parametric setting is the weaker assumption needed for the method to work well, due to the use of the Representer Theorem. On the other hand, the representer theorem needs a regularized problem as its application, and this includes the term λ in the bound. The main drawback is that we need a small regularization to obtain a good non-parametric representation of the function, as indicated by the $(\lambda \max_{k=1,\dots,K} \|T_k^* Q_0\|_{\mathcal{K}}^2)$ term, but cannot lower the λ coefficient too much due to the term

$$\frac{V_{max}^4}{n\lambda^{d/l}}.$$

In particular, if we choose a small regularization when d is too large, and the function to represent is not too smooth, this term explodes again with the dimensionality. In our work we chose to restrict our class of approximator to linear combinations to maintain a better interpretability, and to use a Multi-Task setting to avoid the curse of dimensionality.

Chapter 5

Multi-Task Feature Learning with Group Lasso penalties

In this chapter we will develop a novel optimization algorithm to compute an extension of the Group Lasso problem, based on the convex multi-task feature learning algorithm proposed in [1]. Our goal is to adapt the penalties in the original algorithm to exactly reflect a Group Lasso penalty, in order to have a simpler way to satisfy the theoretical results, while still retaining the possibility of learning a feature representation that can improve the effects of the s factor on group sparsity suggested by Equation (4.4.6).

5.1 Multi-Task Feature Learning

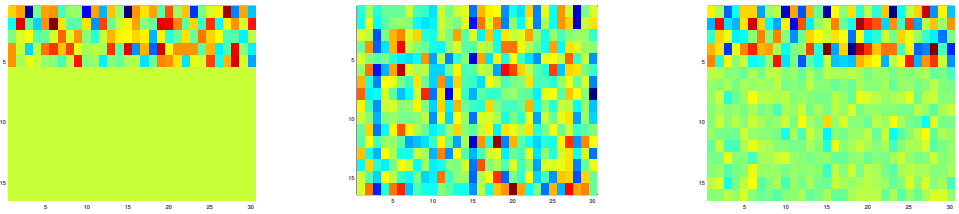
We introduce some additional notation for matrices classes. $\boldsymbol{O}^d$ indicates the set of all orthonormal matrices, while $\boldsymbol{S}^d_+$ the set of all symmetric, semi-definite positive matrices, $\boldsymbol{S}^d_{++}$ for the definite positive ones.

In the original MTFL paper, the Group Lasso's problem was extended to include a feature learning objective, which resulted in the following problem

$$\mathcal{E}_{\text{MTFL}}(A, U) = \sum_{t=1}^{T} \sum_{i=1}^{n} L(y_{ti}, \langle a_t, U^\mathsf{T} x_{ti} \rangle) + \lambda \left\| A \right\|^2_{2,1}, \tag{5.1.1}$$

$$\min \left\{ \mathcal{E}_{\text{MTFL}}(A, U) : U \in \boldsymbol{O}^d, A \in \mathbb{R}^{d \times T} \right\}. \tag{5.1.2}$$

Here L is some convex loss, A is the matrix of regression variables that we commonly call W, and U is an orthonormal matrix that represent the features learned. The new features $x' = U^\mathsf{T} x$ are a linear combination of the original features. This problem is not convex, although as we will see it is separately convex in both its variables, and especially the minimization over U is not immediate. For this reason the authors introduce an equivalent convex problem whose minimization results are more practical.



A UA $\widehat{A}$

Figure 5.1: Visualization of group sparsity recovery using MTFL.

In the original paper no further consideration on the class of learned features is done. To give a better understanding of the potential of MTFL we will explore the potential transformation that can be obtained with this algorithm. In particular, the matrices U that can be learned belong to the space of orthonormal matrices $\boldsymbol{O}^d$. Because the matrices are orthonormal $UU^\mathsf{T} = U^\mathsf{T}U = I$, or in other words all of their singular values are equal to 1. Therefore, their determinant will be 1 or -1. In linear algebra, orthogonal matrices with determinant 1 or -1 form the orthogonal group, while the subset of matrices with determinant 1 form the so called special orthogonal group. In lower dimensions, specifically for R^2 and R^3 these groups can be intuitively associated with rotation and reflections. In particular the special orthogonal group includes all possible transformations composed solely by a rotation, sometimes referred to as proper rotations to distinguish them from the rest of the orthonormal group of improper rotations composed by a rotation and a reflection. The use of MTFL will therefore allow SFQI to be more adaptable to these transformations, and recover a group sparse representation. The price of this flexibility is the loss of theoretical guarantees. In particular it is not proved how many samples are needed to correctly learn the matrix U. As we will see in the experiments, the introduction of the U matrix in experiments where the data is indeed rotated, improves performances, and the addition of more tasks seems to counterbalance the difficulty of learning the transformation.

In practice, we expect MTFL to perform similarly to what is reported in Figure 5.1. In the figure we show an initial matrix A, the non-sparse rotated equivalent UA, and the recovered sparse matrix $\widehat{A}$. In the case of SFQI, the rotation can happen both at the level of the features, as well at the level of the parameters. In particular, given a feature matrix X that can represent the target function in combination with a sparse matrix A, a poor choice of features can result in the rotated matrix $X' = XU^\mathsf{T}$ for which the corresponding best representation is $W = UA$. A normal Group Lasso method cannot easily find a good matrix W, because it is not sparse, while MTFL can learn the right rotation U and obtain a good performance. This is for example the case in the experiments performed in Section 6.4.

5.1.1 Equivalent Problems

We will now skip all the formal proofs, in order not to duplicate the exposition when we will need to follow the same pattern in the derivation of MTFL with Group Lasso penalties. In [37] a series of equivalent problems are introduced, in order to obtain a final problem with an efficient solution. The starting point is a dual formulation of Equation (5.1.1)

$$\begin{aligned}\mathcal{C}_{\text{MTFL}}(W, D) &= \sum_{t=1}^{T}\sum_{i=1}^{n} L(y_{ti}, \langle w_t, x_{ti}\rangle) + \lambda \sum_{t=1}^{T} \langle w_t, D^{+} w_t\rangle \\ &= \sum_{t=1}^{T}\sum_{i=1}^{n} L(y_{ti}, \langle w_t, x_{ti}\rangle) + \lambda \operatorname{trace}(W^{\mathsf{T}} D^{+} W) \end{aligned} \tag{5.1.3}$$

which, unlike Equation (5.1.1) is convex in its arguments, under certain assumptions. This function, together with the assumptions, compose the formulation of Problem (5.1.4)

$$\min\left\{\mathcal{C}_{\text{MTFL}}(W, D) : W \in^{d\times T}, D \in \boldsymbol{S}_{+}^{d}, \operatorname{trace}(D) \leq 1, \operatorname{Ran}(W) \subseteq \operatorname{Ran}(D)\right\}. \tag{5.1.4}$$

As we will see in the next section, the trace constraint is introduced to avoid the matrix D^{+} to simply have infinite trace, eliminating the regularization term. The range constraint instead is there due to the relationship between the optimal solutions of the primal and dual problem. This constraint is hard to enforce, and therefore gets substituted with a restriction of the space of matrices D from $\boldsymbol{S}_{+}^{d}$ to $\boldsymbol{S}_{++}^{d}$. Since now the range of D is the whole $\mathbb{R}^{d}$, the constraint is satisfied. Moreover, the matrix D is now invertible, and we obtain Problem (5.1.5)

$$\min\left\{\mathcal{C}_{\text{MTFL}}(W, D) : W \in^{d\times T}, D \in \boldsymbol{S}_{++}^{d}, \operatorname{trace}(D) \leq 1\right\}. \tag{5.1.5}$$

To solve this problem the original paper proposes an alternating minimization algorithm, where at each step a so called W-step is performed to minimize the regularized loss w.r.t. W, and then a D-step is performed to minimize the function w.r.t. D. Intuitively this correspond to minimizing the objective function with features fixed, and then use the new regression variables obtained to perform an unsupervised learning step and learn the features. Regarding the D-step, we need some way to guarantee that the D matrix will remain positive. For this reason the authors introduce a perturbed version of $\mathcal{C}_{\text{MTFL}}$, and the final problem to be solved is,

$$\mathcal{C}_{\varepsilon\text{-MTFL}}(W, D) = \sum_{t=1}^{T}\sum_{i=1}^{d} L(y_{ti}, \langle w_t, x_{ti}\rangle) + \lambda \operatorname{trace}(D^{-1}(WW^{\mathsf{T}} + \varepsilon I_d)), \tag{5.1.6}$$

$$\min\left\{\mathcal{C}_{\varepsilon\text{-MTFL}}(W, D) : W \in \mathbb{R}^{d\times T}, D \in \boldsymbol{S}_{++}^{d}\right\}. \tag{5.1.7}$$

We can now report the Alternating Minimization algorithm Algorithm 5.1. Intuitively

Algorithm 5.1 MTFL-original

input: $X_t, Y_t, \lambda, tol, \varepsilon, \alpha$
output: W, D
  Initialize $D = I_d/d, k = 1$
  do
    do
      $W_{k-1} \leftarrow W_k$
      $k \leftarrow k + 1$
      $W_k \leftarrow \arg\min_W \mathcal{C}_{\mathcal{E}\text{-MTFL}}(W, D_{k-1})$
      $D_k \leftarrow \frac{(W_k W_k^\intercal + \varepsilon I_d)^{\frac{1}{2}}}{\text{trace}\left((WW^\intercal + \varepsilon I_d)^{\frac{1}{2}}\right)}$
    while $\|W_k - W_{k-1}\|_2 \geq tol$ **and** $k < K$
    $\varepsilon \leftarrow \alpha\varepsilon$
  while $\varepsilon > tol$

the ε term plays the role of a barrier term, keeping the optimization away from a 0 trace D matrix with a $trace(D^{-1})$ term, and keeping the D matrix definite positive. Its presence is a perturbation of the original problem, and we decrease it over time until it converges to the global optimum. Empirically, starting directly with $\varepsilon = 0$, although as we will see it is not theoretically justified, still converges to the correct solution. In the next section we will start to introduce the original adaptation of the proof proposed in [1] to use a different penalty Ω, more similar to the Group Lasso penalty.

5.2 Group Lasso Penalties

If we compare this problem to Group Lasso, we can quickly see that the solution paths are similar, but with a not trivial relationship between them. If we restrict Problem (5.1.2) to use the identity matrix as U, or in other words we do not learn a new feature representation, we obtain a problem that is convex in A, but with the regularization $\Omega(A) = \|A\|_{2,1}^2$ that is the square of the Group Lasso penalty $\Omega(A) = \|A\|_{2,1}$. It is a known result of convex optimization that a necessary and sufficient condition for the optimality of a solution is that the derivative of the objective function in the solution must be null, or its subgradient must contain the null vector. We can exploit this property to find the relationship between the two problems. We will consider the case when $U = I$, which is the most similar to Group Lasso, and ignore the contribution of the loss functions, since they are the same and give the same contribution to the derivative. With λ_{GL}, W respectively a certain regularization parameter value, and the associated Group Lasso solution, and λ_{MTFL} the regularization parameter value that will make the MTFL solution coincide with

W, we have

$$\begin{aligned}\frac{d\lambda_{GL}\left\|W\right\|_{2,1}}{d\left\|W\right\|_{2,1}} &= \frac{d\lambda_{MTFL}\left\|W\right\|_{2,1}^2}{d\left\|W\right\|_{2,1}}\\ \lambda_{GL} &= 2\lambda_{MTFL}\left\|W\right\|_{2,1}\\ \frac{\lambda_{GL}}{2\left\|W\right\|_{2,1}} &= \lambda_{MTFL}. \end{aligned} \tag{5.2.1}$$

This means that if we compute the optimal solution to a MTFL problem without learning the features, and tuning λ_{MTFL} with the value of a known solution to a Group Lasso problem, we will get back to the same Group Lasso solution. In a setting where the only way to choose λ is empirically, this is not too relevant, although it can rise problems when λ_{GL} becomes large. In this case, the norm $\left\|W\right\|_{2,1}$ shrinks, further increasing the magnitude of λ_{MTFL}, and choosing the possible candidates for a cross validation test becomes harder. Moreover, if the choice of λ is guided by some prior information, such as knowing ranges that work well for Group Lasso, or by theoretical analysis then this highly non linear relationship can be detrimental. Because the relationship between the solutions depends on the norm of the optimal solution, and that, in turn, depends on the data, it is hard to relate the solutions of MTFL and Group Lasso. In the previous Chapter we made use of several results from the Group Lasso theoretical analysis, in particular it is not so immediate to extend Theorem 4.3.2 to this new penalty. For this reason we decided instead to change the optimization algorithm's penalty to resemble the Group Lasso penalty more closely. In the following we will provide a modified proof, with the same outline as in [1], but solving several issues in the proof in an original way. Formally, we are going to describe a method to minimize the following problem

$$\mathcal{E}(A,U) = \sum_{t=1}^{T}\sum_{i=1}^{n} L(y_{ti}, \langle a_t, U^{\mathsf{T}} x_{ti}\rangle) + \lambda \left\|A\right\|_{2,1}, \tag{5.2.2}$$

$$\min\left\{\mathcal{E}(A,U) : U \in \boldsymbol{O}^d, A \in \mathbb{R}^{d\times T}\right\}. \tag{5.2.3}$$

5.2.1 Equivalent Problems

We begin by introducing several different equivalent formulations for Problem (5.2.3), in order to find an equivalent formulation that can be efficiently solved. The first equivalent formulation we propose is

$$\begin{aligned}\mathcal{C}(W,D) &= \sum_{t=1}^{T}\sum_{i=1}^{n} L(y_{ti}, \langle w_t, x_{ti}\rangle) + \lambda\left(\sum_{t=1}^{T}\langle w_t, D^{+} w_t\rangle\right)^{\frac{1}{2}}\\ &= \sum_{t=1}^{T}\sum_{i=1}^{n} L(y_{ti}, \langle w_t, x_{ti}\rangle) + \lambda\left(\operatorname{trace}(W^{\mathsf{T}} D^{+} W)\right)^{\frac{1}{2}}. \end{aligned} \tag{5.2.4}$$

The main difference between Equation (5.1.3) and this formulation is the presence of a square root term in the penalty term. We will now see how this difference, in the context of the minimization problem

$$\min\left\{\mathcal{C}(W,D) : W \in^{d\times T}, D \in \boldsymbol{S}_+^d, \operatorname{trace}(D) \leq 1, \operatorname{Ran}(W) \subseteq \operatorname{Ran}(D)\right\} \tag{5.2.5}$$

can result in an equivalent problem to an extension of Group Lasso. First we introduce a lemma

Lemma 5.2.1. *For any $b = (b_1, \ldots, b_d) \in \mathbb{R}^d$ such that $b_i \neq 0, i \in \mathbb{N}_d$, we have that*

$$\min\left\{\left(\sum_{i=1}^d \frac{b_i^2}{\sigma_i}\right)^{\frac{1}{2}} : \sigma_i > 0, \sum_{i=1}^d \sigma_i \leq 1\right\} = \|b\|_1$$

and the minimizer is $\hat{\sigma}_i = \frac{|b_i|}{\|b\|_1}$.

Proof. From the Cauchy-Schwarz inequality

$$\begin{aligned}
\|b\|_1 &= \sum_{i=1}^d \sigma_i^{\frac{1}{2}} \sigma_i^{-\frac{1}{2}} b_i \\
&\leq \left(\sum_{i=1}^d (\sigma_i)^{\frac{1}{2}2}\right)^{\frac{1}{2}} \left(\sum_{i=1}^d \sigma_i^{-\frac{1}{2}2} b_i^2\right)^{\frac{1}{2}} \\
&\leq \left(\sum_{i=1}^d \sigma_i^{-1} b_i^2\right)^{\frac{1}{2}}.
\end{aligned}$$

The minimum is reobtained when the equality is valid, which is satisfied by $\sigma_i = \frac{|b_i|}{\|b\|_1}$

$$\left(\sum_{i=1}^d \sigma_i^{-\frac{1}{2}2} b_i^2\right)^{\frac{1}{2}} = \left(\|b\|_1 \sum_{i=1}^d \frac{b_i^2}{|b_i|}\right)^{\frac{1}{2}} = (\|b\|_1 \|b\|_1)^{\frac{1}{2}} = \|b\|_1.$$

□

We can now formally prove that

Theorem 5.2.2. *Problem* (5.2.3) *is equivalent to Problem* (5.2.5)*, in particular if $(\hat{A}, \hat{U})$ is an optimal solution of* (5.2.3)*, then*

$$(\hat{W}, \hat{D}) = \left(\hat{U}\hat{A}, \hat{U} \operatorname{Diag}\left(\frac{\|\hat{a}^i\|_2}{\|\hat{A}\|_{2,1}}\right)_{i=1}^d \hat{U}^\mathsf{T}\right)$$

is an optimal solution of (5.2.5)

Proof. Given a feasible solution of Problem (5.2.5) (W, D), let $D = U \operatorname{Diag}(\sigma_i)_{i=1}^d U^\mathsf{T}$ be an eigendecomposition and $A = U^\mathsf{T} W$. Then

$$\left(\operatorname{trace}(W^\mathsf{T} D^+ W)\right)^{\frac{1}{2}} = \left(\operatorname{trace}\left(\operatorname{Diag}(\sigma_i^+)_{i=1}^d A A^\mathsf{T}\right)\right)^{\frac{1}{2}} = \left(\sum_{i=1}^d \sigma_i^+ \|a^i\|_2^2\right)^{\frac{1}{2}}.$$

If $\sigma_i = 0$ for any eigenvalue, then $u_i \in \operatorname{null}(D)$ and by the range constraint and $A = U^\mathsf{T} W$ we can deduce $a^i = 0$ and exclude i from the summation. Therefore by Lemma 5.2.1:

$$\left(\sum_{i=1}^d \sigma_i^+ \|a^i\|_2^2\right)^{\frac{1}{2}} = \left(\sum_{a^i \neq 0}^d \frac{\|a^i\|_2^2}{\sigma_i}\right)^{\frac{1}{2}} \geq \left(\left(\sum_{a^i \neq 0}^d \|a^i\|_2\right)^2\right)^{\frac{1}{2}} = \|A\|_{2,1}$$

and $\mathcal{E}(A, U) \leq \mathcal{C}(W, D)$. If we apply the definition of σ_i proposed in Lemma 5.2.1, we see the infimum is attained, and we obtain the relationship between the optimal solutions of the two problems. Therefore the minimum of Problem (5.2.3) does not exceed the minimum of Problem (5.2.5). Conversely, suppose (A, U) is feasible for Problem (5.2.3). We let $W = UA$ and $D = U \operatorname{Diag}\left(\frac{\|a^i\|_2}{\|A\|_{2,1}}\right)_{i=1}^d U^\mathsf{T}$. Then

$$\begin{aligned}
&\left(\operatorname{trace}(W^\mathsf{T} D^+ W)\right)^{\frac{1}{2}} \\
&= \left(\operatorname{trace}(A^\mathsf{T} U^\mathsf{T} U \operatorname{Diag}(\|a^i\|_2^+ \|A\|_{2,1}) U^\mathsf{T} U A)\right)^{\frac{1}{2}} \\
&= (\|A\|_{2,1} \operatorname{trace}(\operatorname{Diag}(\|a^i\|_2^+) A A^\mathsf{T}))^{\frac{1}{2}} \\
&= (\|A\|_{2,1} \sum_{i=1}^d \|a^i\|_2^+ \|a^i\|_2^2)^{\frac{1}{2}} \\
&= (\|A\|_{2,1} \|A\|_{2,1})^{\frac{1}{2}} \\
&= \|A\|_{2,1}.
\end{aligned}$$

Therefore $\mathcal{C}(W, D) = \mathcal{E}(A, U)$, and the two problems have the same minimum. □

This proof shows that the constraints are needed due to the presence of the pseudo inverse of D. Concretely, its presence forces us to impose a trace constraint, to avoid the possibility of setting $\sigma_i = \infty$ obtaining a 0 penalty term. Similarly, if the range of W is not contained in the range of D then we can construct a D such that $W = D0$ and therefore $D^+ W = 0$, again erasing the penalty term. A simple way to resolve this problem is to restrict the space of possible D matrices, at the expense of having to substitute the minimum with an infimum, since now solution with $\sigma_i = 0$ will be only limiting points, and not part of the solution space. We obtain a new problem formulation, and the following corollary

Corollary 5.2.3. *Problem* (5.2.5) *is equivalent to*

$$\inf \left\{ \mathcal{C}(W, D) : W \in^{d \times T}, D \in \boldsymbol{S}^d_{++}, \operatorname{trace}(D) \leq 1 \right\}. \tag{5.2.6}$$

A sequence that minimizes (5.2.6) *converges to a minimum of Problem* (5.2.5)*.*

Proof. From Theorem 5.2.2 and the fact that the minimum is attained. □

To guarantee that the D matrix remains definite positive we again reintroduce a perturbed version of the $\mathcal{C}$ function. Moreover, since we will use it in the rest of the derivations, we substitute the generic convex loss function with the squared loss function, and obtain

$$\mathcal{C}_\varepsilon(W, D) = \sum_{t=1}^{T} \sum_{i=1}^{n} (y_{ti} - \langle w_t, x_{ti} \rangle)^2 + \lambda \left(\operatorname{trace}(D^{-1}(WW^\mathsf{T} + \varepsilon I_d)) \right)^{\frac{1}{2}} : \\ W \in \mathbb{R}^{d \times T}, D \in \boldsymbol{S}^d_{++}. \tag{5.2.7}$$

and the final problem that we will solve

$$\inf \left\{ \mathcal{C}_\varepsilon(W, D) : W \in \mathbb{R}^{d \times T}, D \in \boldsymbol{S}^d_{++}, \operatorname{trace}(D) \leq 1 \right\}. \tag{5.2.8}$$

We diverge now from the original proof, due to the differences in the formulations. In the original paper the optimization Problem (5.2.6) is convex, and is solved with the alternating minimization Algorithm 5.1. Due to the presence of the square root term in our formulation, this is not the case anymore. In the following sections we will show how a modified version of the alternating minimization algorithm can still obtain the global minimum, thanks to the fact that the function are separately convex, and that the penalty term is equivalent to a convex penalty.

5.2.2 Alternating Minimization Algorithm

We will now reintroduce the alternating minimization algorithm, see Algorithm 5.1 to take into account the different penalty. As we will see in the next sections, while the D-step remains unchanged, we will need to modify the W-step. We will leave the details of the W-step minimization for the last section, and will instead now focus on deriving again the formula for the D-step and prove the convergence to the global optimum of the algorithm.

5.2.3 Feature Learning Step

We will now introduce the D-step for the objective function $\mathcal{C}_\varepsilon$. As we will see, it is a convex optimization problem, and the minimum is attained with the same solution as $\mathcal{C}_{\varepsilon\text{-MTFL}}$

Algorithm 5.2 MTFL-GL

input: $X_t, Y_t, \lambda, tol, \varepsilon, \alpha$
output: W, D
  Initialize $D = I_d/d, k = 1$
  do
    do
      $W_{k-1} \leftarrow W_k$
      $k \leftarrow k + 1$
      $W_k \leftarrow \arg\min_W \mathcal{C}_\varepsilon(W, D_{k-1})$
      $D_k \leftarrow \frac{(W_k W_k^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}}{\operatorname{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)}$
    while $\|W_k - W_{k-1}\|_2 \geq tol$ **and** $k < K$
    $\varepsilon \leftarrow \alpha\varepsilon$
  while $\varepsilon > tol$

Theorem 5.2.4. *The minimization of D-step of Algorithm 5.2 is attained with*

$$D_\varepsilon(W) = \frac{(WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}}{\operatorname{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)}.$$

Proof. We set $C \in \boldsymbol{S}_+^d = (WW^\mathsf{T} + \varepsilon I_d)$ We can quickly check that the minimization problem

$$\min\left\{\left(\operatorname{trace}(D^{-1}C)\right)^{\frac{1}{2}} : D \in \boldsymbol{S}_{++}^d, \operatorname{trace}(D) \leq 1\right\}$$

is convex by taking the derivative of the function w.r.t. to D two times. We follow standard derivations, refer to [39]

$$\begin{aligned}
\frac{\partial}{\partial D} &= -\frac{1}{2}\left(\operatorname{trace}(D^{-1}C)\right)^{-\frac{1}{2}} D^{-1}CD^{-1}, \\
\frac{\partial}{\partial'' D} &= -\frac{1}{4}\left(\operatorname{trace}(D^{-1}C)\right)^{-\frac{3}{2}} D^{-1}CD^{-2}CD^{-1} + \left(\operatorname{trace}(D^{-1}C)\right)^{-\frac{1}{2}} D^{-1}CD^{-2}, \\
&= \left(\operatorname{trace}(D^{-1}C\right)^{-\frac{1}{2}} D^{-1}CD^{-2}\left(I - \frac{CD^{-1}}{4\operatorname{trace}(CD^{-1})}\right),
\end{aligned}$$

and since the second D and C matrix are d.p., and the last term is d.p. because it is the difference of an identity matrix minus a matrix with a less than 1 trace, the second derivative is strictly positive. If we substitute $D = U\operatorname{Diag}(\sigma)U^\mathsf{T}, U \in \boldsymbol{O}^d, \sigma \in \mathbb{R}_{++}^d$, we can then minimize w.r.t. σ and U. First, we find the infimum over σ with an

application of Lemma 5.2.1

$$\begin{aligned}
&\inf\left\{\left(\operatorname{trace}\left(C^{\frac{1}{2}}U\operatorname{Diag}(\sigma)^{-1}U^{\mathsf{T}}C^{\frac{1}{2}}\right)\right)^{\frac{1}{2}} : \sigma_i \in \mathbb{R}^d_{++}, \sum_{i=1}^d \sigma_i \leq 1\right\} \\
=&\inf\left\{\left(\sum_{i=1}^d \sigma_i^{-1}\|C^{\frac{1}{2}}u_i\|_2^2\right)^{\frac{1}{2}} : \sigma_i \in \mathbb{R}^d_{++}, \sum_{i=1}^d \sigma_i \leq 1\right\} \\
=&\left(\sum_{i=1}^d \|C^{\frac{1}{2}}u_i\|_2\right) = \|U^{\mathsf{T}}C^{\frac{1}{2}}\|_{2,1}.
\end{aligned}$$

We will now minimize w.r.t. U

$$\inf\left\{\|U^{\mathsf{T}}C^{\frac{1}{2}}\|_{2,1} : U \in \boldsymbol{O}^d\right\} = \operatorname{trace}(C^{\frac{1}{2}}).$$

To show that the minimizing U is composed of eigenvectors of C, we need to show that the trace$(\cdot)$ operator is an inner product associated with the vector space of square matrices. This is easily shown, as the axioms required for a map to be an inner product are satisfied by the trace

$$\begin{aligned}
&\operatorname{trace}(AB) = \operatorname{trace}(BA) && \text{symmetry,}\\
&\operatorname{trace}(aAB) = a\operatorname{trace}(BA) \\
&\operatorname{trace}((A+B)C) = \operatorname{trace}(AC) + \operatorname{trace}(BC) && \text{linearity in the first argument,}\\
&\operatorname{trace}(AA) \geq 0 \\
&\operatorname{trace}(AA) = 0 \Rightarrow A = 0 && \text{positive definiteness.}
\end{aligned}$$

Using $1 = \operatorname{trace}(u_i^{\mathsf{T}}u_i) = \operatorname{trace}(u_iu_i^{\mathsf{T}})$ and similarly $u_i^{\mathsf{T}}u_i = u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}}$ we can write

$$\begin{aligned}
\|C^{\frac{1}{2}}u_i\|_2 &= \left(\operatorname{trace}(u_i^{\mathsf{T}}C^{\frac{1}{2}}C^{\frac{1}{2}}u_i)\right)^{\frac{1}{2}} \\
&= \left(\operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}C^{\frac{1}{2}})\right)^{\frac{1}{2}}\left(\operatorname{trace}(u_iu_i^{\mathsf{T}})\right)^{\frac{1}{2}} \\
&= \left(\operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}}C^{\frac{1}{2}})\right)^{\frac{1}{2}}\left(\operatorname{trace}(u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}})\right)^{\frac{1}{2}} \\
&\geq \left(\left(\operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}})\right)^{\frac{1}{2}}\left(\operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}})\right)^{\frac{1}{2}}\right) \\
&= \left(\left(\operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}u_iu_i^{\mathsf{T}})\right)^{\frac{1}{2}}\right)^2 \\
&= \operatorname{trace}(C^{\frac{1}{2}}u_iu_i^{\mathsf{T}}) = u_i^{\mathsf{T}}C^{\frac{1}{2}}u_i.
\end{aligned}$$

Where the inequality is an application of the Cauchy-Schwarz Theorem using the trace as an inner product. The Cauchy-Schwarz inequality is always strict except when u_i is an eigenvector of $C^{\frac{1}{2}}$, in that case $u_i^{\mathsf{T}}C^{\frac{1}{2}}u_i = a$, and the minimum is attained. Therefore C and D will have the same eigenvectors. D's eigenvalues are

decided by the minimization of Lemma 5.2.1, namely $\sigma_i = \frac{\|C^{\frac{1}{2}} u_i\|_2}{\|U^\intercal C^{\frac{1}{2}}\|_{2,1}}$, constant, so at the minimum D will have the same eigenvalues and eigenvectors of C, with the eigenvalues normalized to 1. The minimizer will therefore be a normalized version of C and the minimum value will be

$$\operatorname{trace}(D^{-1}C)^{\frac{1}{2}} = \operatorname{trace}\left(\left(\frac{C^{\frac{1}{2}}}{\operatorname{trace}\left(C^{\frac{1}{2}}\right)}\right)^{-1} C\right) = \left(\operatorname{trace}(C^{\frac{1}{2}})\operatorname{trace}(C^{-\frac{1}{2}}C)\right)^{\frac{1}{2}}$$

$$= \left(\operatorname{trace}(C^{\frac{1}{2}})\operatorname{trace}(C^{\frac{1}{2}})\right)^{\frac{1}{2}} = \operatorname{trace}(C^{\frac{1}{2}}).$$

Similar reasoning can be made to obtain the same results with

$$\min\left\{\left(\operatorname{trace}(D^{+}C)\right)^{\frac{1}{2}} : D \in \boldsymbol{S}_{+}^{d}, \operatorname{trace}(D) \leq 1, \operatorname{Ran}(C) \subseteq \operatorname{Ran}(D)\right\}.$$

The difference is that, when we apply Lemma 5.2.1 we exclude from the summations all the i whose $\|C^{\frac{1}{2}} u_i\|_2 = 0$, and all the other σ_i are guaranteed to be non-zero from the range constraint. □

From this proof, when we are not learning features, and therefore the matrix D is constrained to be diagonal, we can derive the following corollary

Corollary 5.2.5. *The minimization of D-step of Algorithm 5.2, when the matrix D is constrained to be diagonal, is attained with*

$$D_\varepsilon(W) = \operatorname{Diag}\left(\frac{\|w^i\|_\varepsilon}{\|W\|_{\varepsilon,1}}\right)_{i=1}^{d}.$$

Proof. We follow the same steps as in the previous proof when minimizing over σ, but we cannot minimize over U since it is constrained to be the identity matrix. We only need to apply the definition of σ_i from Lemma 5.2.1 $\sigma_i = \frac{\|C^{\frac{1}{2}} u_i\|_2}{\|U^\intercal C^{\frac{1}{2}}\|_{2,1}}$, where now the u_i vectors are indicator vectors, to prove the corollary. □

5.2.4 Global Optimality

We will now consider the case of function $\mathcal{C}$, and prove that the minimum found by the alternating minimization algorithm is indeed the global minimum. Although the function itself is not convex, it is separately convex, but more importantly the minimization over D involves only the penalty term, and has a closed form solution that we can substitute. A similar setting is described in [37] where the authors analyze a whole family of regularized functions that have a penalty in the form of

$$\Omega(W) = \inf\left\{\Gamma(W, \sigma) : \sigma \in \Sigma\right\},$$

where Γ is a convex function over Σ, a convex subset of $\mathbb{R}^d$. In our different case

$$\Gamma(W, D) = \left(\text{trace}\left(D^{-1}(WW^\mathsf{T} + \varepsilon I_d)\right)\right)^{\frac{1}{a}},$$

and taking the infimum and setting $\epsilon = 0$, after substituting the definition of D from Theorem 5.2.4 we obtain

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{i=1}^{n}(y_{ti}, \langle w_t, x_{ti}\rangle)^2 + \lambda\left(\text{trace}\left(\left(\frac{(WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}}{\text{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)}\right)^{-1}(WW^\mathsf{T} + \varepsilon I_d)\right)\right)^{\frac{1}{2}} \\
&= \sum_{t=1}^{T}\sum_{i=1}^{n}(y_{ti}, \langle w_t, x_{ti}\rangle)^2 + \lambda\left(\text{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)\text{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)\right)^{\frac{1}{2}} \\
&= \mathcal{S}_\varepsilon(W) = \sum_{t=1}^{T}\sum_{i=1}^{n}(y_{ti}, \langle w_t, x_{ti}\rangle)^2 + \lambda\,\text{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right). \qquad (5.2.9)
\end{aligned}$$

This penalty is equivalent to a spectral penalty on the singular values σ_i of the W matrix, and we will prove in the next section that is a convex function. Since this is now a convex optimization, a necessary and sufficient condition for its optimality is that its derivative is null in the optimum. Given matrices (W_k, D_K) at iteration k, computed according to Algorithm 5.2 we can have two situations. If the W_k is not optimal the derivative of the convex function (5.2.9) is not zero. Therefore we will have $W_{k+1} \neq W_k$ and the algorithm will continue in its iterations. Since, as we will see in the next section, at each step the value of the objective function decreases, we will not consider the couple (W_k, D_k) again. If instead W_k is optimal, then $W_{k+1} = W_k$. Therefore also $D_{k+1} = D_k$ and we will terminate. In other words, we cannot terminate in a local optima, because the couple $(W_k, D_{k)}$ allows us to compute the true gradient of a convex function at each step. Another interpretation of the alternating minimization algorithm is a gradient descent in the W_k space, where at each step a D_k matrix is computed to obtain the gradient. In the next section we will prove that the objective function decreases with each consecutive application of D-steps and W-steps. Therefore instead of setting a stepsize for the gradient update, the alternating minimization algorithm simply minimizes each step as much as possible. We will now prove that the algorithm always converges to the minimizer of Problem (5.2.5)

5.2.5 Convergence

This section covers the main result of this chapter, the convergence of Algorithm 5.2 to a minimizer of Problem (5.2.5), and from there to a minimizer of Problem (5.2.3). In particular we want to prove the following two theorems

Theorem 5.2.6. *For every $\varepsilon > 0$ the sequence $\{(W_k, D_\varepsilon(W_k)) : k \in \mathbb{N}_K\}$ converges to the minimizer of Problem* (5.2.8).

Since Problem (5.2.8) reduces to (5.2.6) when $\varepsilon \to 0$, and then (5.2.6) converges to (5.2.5), we can state the following result.

Theorem 5.2.7. *Consider the sequence of functions* $\{\mathcal{C}_{\varepsilon_\ell} : \ell \in \mathbb{N}\}$ *such that* $\varepsilon_\ell \to 0$ *as* $\ell \to \infty$. *Any limiting point of the minimizer of the sequence, under the constraints of Problem* (5.2.8), *is an optimal solution to* (5.2.5).

Proof of Theorem 5.2.6 and Theorem 5.2.7. For convenience we remind

$$\mathcal{S}_\varepsilon(W) = \mathcal{C}_\varepsilon(W, D_\varepsilon(W)) = \sum_{t=1}^{T}\sum_{i=1}^{n}(y_{ti}, \langle w_t, x_{ti}\rangle)^2 + \lambda \operatorname{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right).$$

We also define a formalization of the W-step,

$$g_\varepsilon(W) = \min\left\{\mathcal{C}_\varepsilon(V, D_\varepsilon(W)) : V \in \mathbb{R}^{d\times T}\right\}.$$

Since $\mathcal{S}_\varepsilon(W) = \mathcal{C}_\varepsilon(W, D_\varepsilon(W))$ and $D_\varepsilon(W) = \min \mathcal{C}_\varepsilon(W, \bullet)$ we can derive:

$$\mathcal{S}_\varepsilon(W_{(k+1)}) \leq g_\varepsilon(W_{(k)}) \leq \mathcal{S}_\varepsilon(W_{(k)}). \tag{5.2.10}$$

We can also show that $\mathcal{S}_\varepsilon$ has a unique minimum.

Proposition 5.2.8. $\mathcal{S}_\varepsilon$ *is strictly convex for every* $\varepsilon > 0$

Proof. The term $\operatorname{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)$ can be seen as a spectral operator, and a convex one. Given a singular value decomposition $U\Sigma V^\mathsf{T} = W$

$$\begin{aligned}
f(\sigma) &= \operatorname{trace}\left((WW^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)\\
&= \operatorname{trace}\left(((U\operatorname{Diag}(\sigma)V^\mathsf{T}(U\operatorname{Diag}(\sigma)V^\mathsf{T})^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right)\\
&= \operatorname{trace}\left(((U\operatorname{Diag}(\sigma)V^\mathsf{T}(V\operatorname{Diag}(\sigma)U^\mathsf{T}) + \varepsilon U I_d U^\mathsf{T})^{\frac{1}{2}}\right)\\
&= \operatorname{trace}\left(((U(\operatorname{Diag}(\sigma)^2 + \varepsilon I_d)U^\mathsf{T}))^{\frac{1}{2}}\right)\\
&= \sum_{i=1}^{d}\left(\sigma_i^2 + \varepsilon\right)^{\frac{1}{2}}.
\end{aligned}$$

To prove that $f(\sigma)$ is convex, we compute the Hessian, beginning with the gradient

$$\frac{\partial f(\sigma)}{\partial \sigma_i} = \frac{\sigma_i}{(\sigma_i^2 + \varepsilon)^{1/2}}.$$

Since already the first derivative depends only on σ_i, the Hessian will be a diagonal matrix with diagonal

$$\frac{\partial f(\sigma)}{\partial^2 \sigma_i} = \frac{1}{(\sigma_i^2 + \varepsilon)^{1/2}}\left(1 - \frac{\sigma_i^2}{\sigma_i^2 + \varepsilon}\right),$$

which is strictly positive for $\varepsilon > 0$. Therefore the Hessian will be d.p. and the function is strictly convex. □

For the convergence of Algorithm 5.2, we also need some guarantees on g_ε.

Lemma 5.2.9. *The function g_ε is continuous for every $\varepsilon > 0$.*

Proof. We will proceed by proving that a more general function

$$G_\varepsilon(D) = \min \mathcal{C}_\varepsilon(W, D) : V \in \mathbb{R}^{d\times T}, D \in \boldsymbol{S}_{++} \tag{5.2.11}$$

is continuous. To do this we will follow a similar approach as in [2]. First we rewrite $\mathcal{C}_\varepsilon(W, D)$ to be a function of a vector w and not on a matrix W, similarly to what we did in Section 4.3. We build again a new X matrix, which is a block diagonal matrix with all the data matrices X_t on the diagonal, a vector Y with all the response vector stacked, and a matrix $\overline{D}$ which is again block diagonal with T copies of the D matrix on the diagonal. We also introduce the $\varepsilon \operatorname{trace}(D^{-1}) = \varepsilon^D$ term, to have a shorter notation. We can then formulate

$$\mathcal{C}_\varepsilon(w, D) = (Y - Xw)^\mathsf{T}(Y - Xw) + \lambda \left(w^\mathsf{T}\overline{D}^{-1}w + \varepsilon^D\right)^{\frac{1}{2}}.$$

We need now to prove that the minimum of this function is continuous w.r.t. to D. We begin to characterize the minimum in terms of a generic loss function $L(w)$. Since $D \in \boldsymbol{S}^d_{++}$, the vector $c = \overline{D}^{-1}w$ is uniquely identified and $w = Dc$. We can then introduce the regularized loss function

$$L_\lambda(\overline{D}) = \min \left\{ L(\overline{D}c) + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}} : c \in \mathbb{R}^{Td} \right\}$$

This is a regularized problem with a weighted norm as its regularization, when the matrix inducing the norm is d.p., as in our case, this problem is convex in c if the loss itself L is convex. A convex function L admits a Legendre-Fenchel transformation L^*, and the following relationships hold

$$L(w) = \sup \left\{ w^\mathsf{T}v - L^*(v) : v \in \mathbb{R}^{Td} \right\},$$
$$L^*(v) = \sup \left\{ w^\mathsf{T}v - L(w) : w \in \mathbb{R}^{Td} \right\}.$$

If the loss $L(w)$ is bounded from below, such as a square loss, then $L^*(0) = -\inf \left\{ L(w) : w \in \mathbb{R}^{Td} \right\} < \infty$. We can now define

$$h(c, v) = v^\mathsf{T}Kc - L^*(v) + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}},$$
$$V = \{v : L^*(v) < \infty, v \in \mathbb{R}^{Td}\}$$

and by applying Von Neumann minimax Theorem [2] we can write

$$\begin{aligned} L_\lambda(\overline{D}) &= \min \left\{ L(\overline{D}c) + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}} : c \in \mathbb{R}^m \right\} \\ &= \min \left\{ \sup \left\{ c^\mathsf{T}\overline{D}v - L^*(v) + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}} : v \in V \right\} : c \in \mathbb{R}^m \right\} \\ &= \sup \left\{ \min \left\{ c^\mathsf{T}\overline{D}v + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}} : c \in \mathbb{R}^m \right\} - L^*(v) : v \in V \right\}. \end{aligned}$$

We have now to characterize $\min\left\{w^\mathsf{T}v + \lambda(c^\mathsf{T}\overline{D}c + \varepsilon^D)^{\frac{1}{2}}\right\}$ for a given v. First we pass through two variable substitutions. Since the $\overline{D}$ arises from the d.p. matrix D, the terms $\overline{D}^{\frac{1}{2}}$ is well defined. Let $\overline{D} = UEU^\mathsf{T}$ be an eigendecomposition, then we introduce $c' = E^{\frac{1}{2}}U^\mathsf{T}c$ and $v' = E^{\frac{1}{2}}U^\mathsf{T}v$. The minimization problem can be rewritten as

$$\min\left\{v'^\mathsf{T}c' + \lambda(c'^\mathsf{T}c + \varepsilon^D)^{\frac{1}{2}}\right\}.$$

This problem is convex, and the regularizer is smooth, thus a necessary and sufficient condition for the minimum to exist is the nullity of the derivative.

$$v' + \lambda\frac{c'}{(c'^\mathsf{T}c' + \varepsilon^D)^{\frac{1}{2}}} = 0.$$

We can easily see that the vector to minimize c' is normalized by its denominator, and to obtain the null vector we need the c' vector to approximate $-v'$. The normalization depends on λ and ε, thus we introduce $\alpha \in \mathbb{R}_+$, substitute $c' = -\alpha v'$ and rewrite the equation as

$$\begin{aligned}
v' - \lambda\frac{\alpha v'}{(\alpha^2 v'^\mathsf{T}v' + \varepsilon^D)^{\frac{1}{2}}} &= 0 \\
v' = \lambda\frac{\alpha}{(\alpha^2 v'^\mathsf{T}v' + \varepsilon^D)^{\frac{1}{2}}}v' & \\
1 = \lambda\frac{\alpha}{(\alpha^2 v'^\mathsf{T}v' + \varepsilon^D)^{\frac{1}{2}}} & \\
1 = \lambda^2\frac{\alpha^2}{(\alpha^2 v'^\mathsf{T}v' + \varepsilon^D)} & \\
\alpha^2 v'^\mathsf{T}v' + \varepsilon^D = \lambda^2\alpha^2 & \\
\alpha^2 = \frac{\varepsilon^D}{\lambda^2 - v'^\mathsf{T}v'}. &
\end{aligned}$$

If $\lambda^2 < v'^\mathsf{T}v'$, then $\alpha^2 < 0$ which is impossible for real numbers. Therefore for a solution to exists we have $\lambda^2 \geq v'^\mathsf{T}v'$. The case $\lambda^2 = v'^\mathsf{T}v'$ is again unfeasible. It follows that the only way to have a solution is to have $\lambda^2 > v'^\mathsf{T}v' = v^\mathsf{T}\overline{D}v$. Geometrically, this translate into having the v vector inside the open ellipsoid defined by $\overline{D}$ and its radius λ^2. This can be interpreted as an underlying constraint on the outer maximization problem, because if the inner problem does not have a null derivative in some point, then it is convex and unbounded, and its objective value will be $-\infty$. Since the vector $v = 0$ satisfies the inequality, and produces a solution with an objective function greater than $-\infty$, the solution of the outer problem v will never lie outside or on the surface of the ellipsoid, even without any explicit

constraint. We can now compute the objective of the inner problem in terms of v,

$$
\begin{aligned}
&-\left(\frac{\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}\|v'\|^2+\lambda\left(\frac{\varepsilon^D}{\lambda^2-\|v'\|^2}\|v'\|^2+\varepsilon^D\right)^{\frac{1}{2}}\\
&=-\left(\frac{\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}\|v'\|^2+\left(\frac{\lambda^2\varepsilon^D\|v'\|^2}{\lambda^2-\|v'\|^2}+\frac{(\lambda^2-\|v'\|^2)\lambda^2\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}\\
&=-\left(\frac{\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}\|v'\|^2+\left(\frac{(\lambda^2\|v'\|^2-\lambda^2\|v'\|^2+\lambda^4)\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}\\
&=\left(\frac{\varepsilon^D}{\lambda^2-\|v'\|^2}\right)^{\frac{1}{2}}(\lambda^2-\|v'\|^2)=\left(\varepsilon^D(\lambda^2-\|v'\|^2)\right)^{\frac{1}{2}}=\left(\varepsilon^D(\lambda^2-v^{\mathsf{T}}\overline{D}v)\right)^{\frac{1}{2}}.
\end{aligned}
$$

By substituting the optimal value of α, and using the definition of the Legendre-Fenchel dual function of the square loss

$$L(w)=\|w-y\|^2, L^*(v)=\frac{1}{4}\|v\|^2+y^{\mathsf{T}}v.$$

The outer optimization becomes

$$L_\lambda(\overline{D})=\sup\left\{\left(\varepsilon^D(\lambda^2-v^{\mathsf{T}}\overline{D}v)\right)^{\frac{1}{2}}-\frac{1}{4}\|v\|^2-y^{\mathsf{T}}v : v^{\mathsf{T}}\overline{D}v<\lambda^2\right\}.$$

We can see that the objective function is continuous, and we can also prove that optimization problem is concave, as a sum of concave functions. $-\|v\|^2-yv$ is concave, and we can show that $f(v)=(\lambda^2-v\overline{D}v)^{\frac{1}{2}}$ has a negative Hessian. To do this we compute a generic xHx and prove that it is always negative

$$
\begin{aligned}
&\frac{\partial f(v)}{\partial v_i}=\frac{\overline{D}^i v}{(\lambda^2-v^{\mathsf{T}}\overline{D}v)^{\frac{1}{2}}},\\
&H_{i,j}=\frac{\partial f(v)}{\partial v_i\partial v_j}=-\frac{\overline{D}_{i,j}}{(\lambda^2-v^{\mathsf{T}}\overline{D}v)^{\frac{1}{2}}}-\frac{\overline{D}^i v\overline{D}^j v}{(\lambda^2-v^{\mathsf{T}}\overline{D}v)^{\frac{1}{2}}},\\
&xHx=\sum_{i,j}H_{i,j}x_ix_j=-\frac{x^{\mathsf{T}}\overline{D}x}{(\lambda^2-v^{\mathsf{T}}\overline{D}v)^{\frac{1}{2}}}-\frac{x^{\mathsf{T}}\overline{D}vv^{\mathsf{T}}\overline{D}x}{(\lambda^2-v^{\mathsf{T}}\overline{D}v)^{\frac{1}{2}}}<0.
\end{aligned}
$$

For the next result, we first need to introduce the following result

Lemma 5.2.10. *For any function $f(x,y)$, continuous in x and concave in y, then $g(x)=\max_y f(x,y)$ is continuous in x.*

Proof. From the definition of continuity

$$\forall e\ \exists\delta : |x_1-x_2|<\delta\Rightarrow|g(x_1)-g(x_2)|<e.$$

We have $g(x_1) = f(x_1, y_1), g(x_2) = f(x_2, y_2)$, where y_1, y_2 are the optimal solutions of the maximization problem. By adding and subtracting mixed terms

$$|f(x_1, y_1) - f(x_2, y_1) + f(x_2, y_1) - f(x_1, y_2) + f(x_1, y_2) - f(x_2, y_2)| \leq$$
$$|f(x_1, y_1) - f(x_2, y_1)| + |f(x_2, y_1) - f(x_1, y_2)| + |f(x_1, y_2) - f(x_2, y_2)| \leq e.$$

Since f is continuous in x, the first and third term can be bounded in term of δ. To bound the second term first we assume $f(x_2, y_1) > f(x_1, y_2)$, and since $f(x_2, y_2) \geq f(x_2, y)\ \forall y$ because of the convexity and the definition of g we derive

$$f(x_2, y_1) - f(x_1, y_2) \leq f(x_2, y_2) - f(x_1, y_2) < e$$

due to the continuity of f. A symmetrical derivation can be followed if $f(x_2, y_1) < f(x_1, y_2)$. □

Using Lemma 5.2.10, we can prove that $L_\lambda(\overline{D})$ is indeed continuous in $\overline{D}$. We only need to be careful since the two functions $L_\lambda(\overline{D}_1), L_\lambda(\overline{D}_2)$ have to follow restrictions in the solutions based on their arguments. Due to the fact that the feasible region is open, we can guarantee that when $|\overline{D}_1 - \overline{D}_2| < \delta$ then the optimal solutions v_1, v_2 are feasible for both problems. Formally we want to prove that $v_1^\mathsf{T} \overline{D}_2 v_1 < \lambda^2$. Since the ellipsoid is open $v_1^\mathsf{T} \overline{D}_1 v_1 + e = \lambda^2$ for some small value e. we can then write

$$v_1^\mathsf{T} \overline{D}_2 v_1 - v_1^\mathsf{T} \overline{D}_1 v_1 < e,$$
$$v_1^\mathsf{T} (\overline{D}_2 - \overline{D}_1) v_1 < e,$$

which is satisfied when δ is small enough. □

We can now prove Theorem 5.2.6. By Equation (5.2.10) the sequence $\{\mathcal{S}_\varepsilon(W_k) : k \in \mathbb{N}\}$ is nonincreasing, and is bounded by the loss function from below. As $n \to \infty$, $\mathcal{S}_\varepsilon$ will converge to some value $\widetilde{\mathcal{S}_\varepsilon}$. Since the loss is bounded, and $\{\mathcal{S}_\varepsilon(W_k)\}$ as a whole is bounded, we can also deduce that the sequence $\left\{\text{trace}\left((W_k {W_k}^\mathsf{T} + \varepsilon I_d)^{\frac{1}{2}}\right) : k \in \mathbb{N}\right\}$ is bounded, and therefore $\{W_k : k \in \mathbb{N}\}$ is bounded too. Hence there must be a subsequence $\left\{W_{(k_\ell)} : \ell \in \mathbb{N}\right\}$ which is convergent and whose limit we denote as $\widetilde{W}$. We can now adapt Equation (5.2.10) to show that $\mathcal{S}_\varepsilon(W_{(k_\ell+1)}) \leq g_\varepsilon(W^{(k_\ell)}) \leq \mathcal{S}_\varepsilon(W_{(k_\ell)})$, and therefore $g_\varepsilon(W_{(k_\ell)})$ converges to $\widetilde{\mathcal{S}_\varepsilon}$. Thus, by the continuity of $\mathcal{S}_\varepsilon$ and g_ε, $g_\varepsilon(\widetilde{W}) = \mathcal{S}_\varepsilon(\widetilde{W})$. By the definition of g_ε, $g_\varepsilon(\widetilde{W})$ is the minimum value $\mathcal{C}_\varepsilon(\bullet, D_\varepsilon(\widetilde{W}))$ can take, and it is equal to $\widetilde{\mathcal{S}_\varepsilon}$. Therefore $\widetilde{W}$ is the minimizer of $\mathcal{C}_\varepsilon(\bullet, D_\varepsilon(\widetilde{W}))$. Moreover $D_\varepsilon(\widetilde{W})$ is the minimizer of $\mathcal{C}_\varepsilon(\widetilde{W}, \bullet)$, subject to the constraints of Problem (5.2.8). Since the regularizer in $\mathcal{C}_\varepsilon$ is smooth any directional derivative of $\mathcal{C}_\varepsilon$ is the sum of its directional derivatives w.r.t. W and D [54]. Hence $(\widetilde{W}, D_\varepsilon(\widetilde{W}))$ is the minimizer of $\mathcal{C}_\varepsilon$. Moreover since the sequence $\{W_k : k \in \mathbb{N}\}$ is bounded, it converges to it as a whole and not only as a subsequence. To prove Theorem 5.2.7, we let $(W_{\ell_k}, D_{\varepsilon_{\ell_k}}(W_{\ell_k}))$ be a limiting subsequence of the minimizers of $\{\mathcal{C}_{\varepsilon_\ell} : \ell \in \mathcal{N}\}$.

As $k \to \infty$ the subsequence approaches its limit $(\widetilde{W}, \widetilde{D})$. From the definition of $\mathcal{S}_\varepsilon$, we can see that the function is decreasing in ε, and as $\varepsilon \to 0$ it converges to $\overline{\mathcal{S}} = \min\{\mathcal{S}_0(W)\}$. Because $\mathcal{S}_\varepsilon$ is continuous in both ε and W, and $\mathcal{S}_{\varepsilon_{\ell_n}} \to \overline{\mathcal{S}}$, we obtain that $\mathcal{S}_0(\widetilde{W}) = \overline{\mathcal{S}}$. □

5.2.6 Loss Minimization Step

Since we have proved that the algorithm converges to the global solution, we need now a method to actually compute the W-step in order to run the algorithm. We start by remarking that the problem

$$\mathcal{C}_\varepsilon(W, D) = \sum_{t=1}^{T}\sum_{i=1}^{n}(y_{ti} - \langle w_t, x_{ti}\rangle)^2 + \lambda\left(\operatorname{trace}(W^\mathsf{T} D^{-1} W) + \varepsilon \operatorname{trace}(D^{-1})\right)^{\frac{1}{2}}$$

is convex in W. Since this is a convex problem, we can use any off the shelf minimizer to solve it, but generic solver, even when they have information regarding derivatives, can be slow. For this reason we exploit the convexity, and again look for the solution that corresponds to the null derivative. First we perform again change of variables, introducing the matrices $X, Y, \overline{D}$, the vector w and substituting $v = \overline{D}^{-1/2} w, \overline{X} = X\overline{D}^{1/2}, \varepsilon^D = \varepsilon \operatorname{trace}(D^{-1})$. We obtain

$$\mathcal{C}_\varepsilon(v, \overline{D}) = \left\|Y - \overline{X}v\right\|_2^2 + \lambda(v^\mathsf{T} v + \varepsilon^D)^{\frac{1}{2}}.$$

By taking the derivative w.r.t. v

$$2\overline{X}^\mathsf{T}\overline{X}v + \lambda\frac{v}{(v^\mathsf{T} v + \varepsilon^D)^{\frac{1}{2}}} - 2\overline{X}^\mathsf{T} Y = 0.$$

Again, we could take the norm of the left hand side, and since it is a convex problem, we could minimize it with off the shelf solvers. But another technique that proved to work well, makes use of a single Singular Value Decomposition for each task, and then needs only to solve a scalar iterative problem, followed by a single matrix inversion. We know that the problem is convex, but we will now show that it is strictly convex. A sufficient condition for strict convexity is that the Hessian is d.p.. To compute the Hessian

$$\begin{aligned}
\frac{\partial \mathcal{C}_\varepsilon(v, \overline{D})}{\partial v_i} &= \sum_n -2x_{ni}y_n + 2\sum_j x_{ni}x_{nj}v_j + \lambda\frac{v_i}{(\sum_j v_j^2 + \varepsilon^D)^{1/2}},\\
\frac{\partial^2 \mathcal{C}_\varepsilon(v, \overline{D})}{\partial v_i \partial v_j} &= \sum_n 2\sum_j x_{ni}x_{nj} - \frac{\lambda}{2}\frac{v_i v_j}{(\sum_z v_z^2 + \varepsilon^D)^{3/2}},\\
\frac{\partial^2 \mathcal{C}_\varepsilon(v, \overline{D})}{\partial^2 v_i} &= \sum_n 2\sum_j x_{ni}x_{nj} - \lambda\left(\frac{1}{(\sum_z v_z^2 + \varepsilon^D)^{1/2}} - \frac{v_i^2}{2(\sum_z v_z^2 + \varepsilon^D)^{3/2}}\right),\\
H &= 2X^\mathsf{T} X + \frac{\lambda}{(v^\mathsf{T} v + \varepsilon^D)^{1/2}}\left(I - \frac{vv^\mathsf{T}}{2(v^\mathsf{T} v + \varepsilon^D)^{1/2}}\right).
\end{aligned}$$

which is defined for all v, and its strictly d.p.. The function will therefore have a unique global minimum, which will satisfy the null derivative equation. We can then introduce the C_ε operator

$$C_\varepsilon v = (2\overline{X}^\mathsf{T}\overline{X} + \frac{\lambda}{(v^\mathsf{T}v+\varepsilon)^{1/2}}I)^{-1}2\overline{X}^\mathsf{T}Y.$$

We can now express the operator better in terms of a singular value decomposition $\overline{X} = U\Sigma V^\mathsf{T}$

$$\begin{aligned}
v^* &= (2\overline{X}^\mathsf{T}\overline{X} + \frac{\lambda}{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}I)^{-1}2\overline{X}^\mathsf{T}Y \\
v^* &= (2V\Sigma U^\mathsf{T}U\Sigma V^\mathsf{T} + \frac{\lambda}{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}I)^{-1}2\overline{X}Y \\
v^* &= (2V\Sigma^2 V^\mathsf{T} + \frac{\lambda}{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}VV^\mathsf{T})^{-1}2\overline{X}Y \\
v^* &= \left(V \operatorname{Diag}\left(2\sigma_i^2 + \frac{\lambda}{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}\right)_{i=1}^d V^\mathsf{T}\right)^{-1} 2\overline{X}Y \\
v^* &= \left(V \operatorname{Diag}\left(\frac{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}{2\sigma_i^2(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}+\lambda}\right)_{i=1}^d V^\mathsf{T}\right) 2\overline{X}Y.
\end{aligned}$$

We are now going to prove that v^* can be obtained by successive applications of C_ε. If we consider a perturbation v^+, v^- such that $\|v^+\|_2 = \|v^*\|_2 + \epsilon > \|v^*\|_2$ or $\|v^-\|_2 = \|v^*\|_2 - \epsilon < \|v^*\|_2$, then

$$\begin{aligned}
\|v^*\|_2 \leq \|C_\varepsilon v^+\|_2 < \|v^+\|_2, \\
\|v^-\|_2 < \|C_\varepsilon v^-\|_2 \leq \|v^*\|_2 .
\end{aligned}$$

Now defining $b_i = v_i^\mathsf{T} 2\overline{X}Y$ and exploiting the fact that

$$\|v^*\|_2 = \|C_\varepsilon v^*\|_2 = \left(\sum_i \left(\left(\frac{(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}}{2\sigma_i^2(v^{*\mathsf{T}}v^*+\varepsilon)^{1/2}+\lambda}\right)b_i\right)^2\right)^{1/2},$$

we can derive

$$\begin{aligned}
\|C_\varepsilon v^+\|_2^2 &= \sum_i \left(\left(\frac{(v^{+\mathsf{T}}v^+ + \varepsilon)^{1/2}}{2\sigma_i^2(v^{+\mathsf{T}}v^+ + \varepsilon)^{1/2}+\lambda}\right)b_i\right)^2 \\
&= \frac{(v^{+\mathsf{T}}v^+ + \varepsilon)}{(v^{*\mathsf{T}}v^* + \varepsilon)} \sum_i \frac{(v^{*\mathsf{T}}v^* + \varepsilon)}{(2\sigma_i^2(v^{+\mathsf{T}}v^+ + \varepsilon)^{1/2}+\lambda)^2} b_i^2 \\
&< \frac{(v^{+\mathsf{T}}v^+ + \varepsilon)}{(v^{*\mathsf{T}}v^* + \varepsilon)} \sum_i \frac{(v^{*\mathsf{T}}v^* + \varepsilon)}{(2\sigma_i^2(v^{*\mathsf{T}}v^* + \varepsilon)^{1/2}+\lambda)^2} b_i^2 = \frac{(v^{+\mathsf{T}}v^+ + \varepsilon)}{(v^{*\mathsf{T}}v^* + \varepsilon)}(v^{*\mathsf{T}}v^* + \varepsilon - \varepsilon) \\
&= v^{+\mathsf{T}}v^+ + \varepsilon - \varepsilon\left(\frac{v^{+\mathsf{T}}v^+ + \varepsilon}{v^{*\mathsf{T}}v^* + \varepsilon}\right) = v^{+\mathsf{T}}v^+ - \varepsilon\left(\frac{v^{+\mathsf{T}}v^+ + \varepsilon}{v^{*\mathsf{T}}v^* + \varepsilon} - 1\right) \leq v^{+\mathsf{T}}v^+.
\end{aligned}$$

Conversely

$$\begin{aligned}
\|C_\varepsilon v^-\|_2^2 &= \sum_i \left(\left(\frac{(v^{-\mathsf{T}} v^- + \varepsilon)^{1/2}}{2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda} \right) b_i \right)^2 \\
&= \frac{(v^{-\mathsf{T}} v^- + \varepsilon)}{(v^{*\mathsf{T}} v^* + \varepsilon)} \sum_i \frac{(v^{*\mathsf{T}} v^* + \varepsilon)}{(2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda)^2} b_i^2 \\
&> \frac{(v^{-\mathsf{T}} v^- + \varepsilon)}{(v^{*\mathsf{T}} v^* + \varepsilon)} \sum_i \frac{(v^{*\mathsf{T}} v^* + \varepsilon)}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)^2} b_i^2 \\
&= \frac{(v^{-\mathsf{T}} v^- + \varepsilon)}{(v^{*\mathsf{T}} v^* + \varepsilon)} (v^{*\mathsf{T}} v^* + \varepsilon - \varepsilon) = v^{-\mathsf{T}} v^- + \varepsilon - \varepsilon \left(\frac{v^{-\mathsf{T}} v^- + \varepsilon}{v^{*\mathsf{T}} v^* + \varepsilon} \right) \\
&= v^{-\mathsf{T}} v^- + \varepsilon \left(1 - \frac{v^{-\mathsf{T}} v^- + \varepsilon}{v^{*\mathsf{T}} v^* + \varepsilon} \right) \geq v^{-\mathsf{T}} v^-.
\end{aligned}$$

We now need to prove that the operator will not overshoot, we do so by deriving the following pointwise inequalities

$$\begin{aligned}
&\frac{(v^{*\mathsf{T}} v^* + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i = \frac{1}{2\sigma_i^2} \frac{2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i \\
&= \frac{1}{2\sigma_i^2} \frac{2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda - \lambda}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i = \frac{1}{2\sigma_i^2} \left(1 - \frac{\lambda}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i \right) \\
&\leq \frac{1}{2\sigma_i^2} \left(1 - \frac{\lambda}{(2\sigma_i^2 (v^{+\mathsf{T}} v^+ + \varepsilon)^{1/2} + \lambda)} b_i \right) = \frac{(v^{+\mathsf{T}} v^+ + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{+\mathsf{T}} v^+ + \varepsilon)^{1/2} + \lambda)} b_i
\end{aligned}$$

and

$$\begin{aligned}
&\frac{(v^{-\mathsf{T}} v^- + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda)} b_i = \frac{1}{2\sigma_i^2} \frac{2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda)} b_i \\
&= \frac{1}{2\sigma_i^2} \frac{2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda - \lambda}{(2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda)} b_i = \frac{1}{2\sigma_i^2} \left(1 - \frac{\lambda}{(2\sigma_i^2 (v^{-\mathsf{T}} v^- + \varepsilon)^{1/2} + \lambda)} b_i \right) \\
&\leq \frac{1}{2\sigma_i^2} \left(1 - \frac{\lambda}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i \right) = \frac{(v^{*\mathsf{T}} v^* + \varepsilon)^{1/2}}{(2\sigma_i^2 (v^{*\mathsf{T}} v^* + \varepsilon)^{1/2} + \lambda)} b_i.
\end{aligned}$$

This proves that by repeatedly applying the C_ε operator we will converge to the optimal $\|v^*\|_2$ value, and therefore to the optimal v^*. We just need to do the necessary inverse substitution $w = \overline{D} v$ to obtain the optimal solution. To compute efficiently the C_ε operator, we introduce the following lemma

Lemma 5.2.11. *For any* $\lambda > 0$

$$(2X^{\mathsf{T}} X + \lambda I_d)^{-1} 2X^{\mathsf{T}} Y = 2X^{\mathsf{T}} (XX^{\mathsf{T}} + \lambda I_n)^{-1} Y.$$

Proof.

$$\begin{aligned}
4X^\mathsf{T}XX^\mathsf{T} + 2\lambda X^\mathsf{T} =& 4X^\mathsf{T}XX^\mathsf{T} + 2\lambda X^\mathsf{T} \\
2X^\mathsf{T}(2XX^\mathsf{T} + \lambda I_n) =& (2X^\mathsf{T}X + \lambda I_d)2X^\mathsf{T} \\
(2X^\mathsf{T}X + \lambda I_d)^{-1}2X^\mathsf{T} =& 2X^\mathsf{T}(2XX^\mathsf{T} + \lambda I_n)^{-1} \\
(2X^\mathsf{T}X + \lambda I_d)^{-1}2X^\mathsf{T}Y =& 2X^\mathsf{T}(2XX^\mathsf{T} + \lambda I_n)^{-1}Y.
\end{aligned}$$

□

We can now exploit the fact that the matrix $\overline{X}$ is block diagonal, to compute each update step separately. This is because $\|v\|_2^2$ is simply the sum of the contributions of each task $\|v_t\|_2^2$. We can exploit this, and instead of computing the necessary SVD of $\overline{X}$, we compute the smaller SVDs of $\overline{X}^\mathsf{T}\overline{X}$ or $\overline{X}\overline{X}^\mathsf{T}$, depending on the dimensionality of the matrices. Since now the matrices are square, the resulting problem can also be solved as an eigenproblem, resulting in the decomposition $V\Sigma V^\mathsf{T}$. We need to consider the fact that the values in the Σ matrix will be the square of the singular values of $\overline{X}$, but that is all that we need. If $d \geq n_t$

$$\|v_t\|_2^2 = \sum_i \left(\frac{(v^\mathsf{T}v + \varepsilon)^{1/2}}{(2\sigma_i(v^\mathsf{T}v + \varepsilon)^{1/2} + \lambda)} \right)^2 \left(2v_i^\mathsf{T} X_t^\mathsf{T} Y_t \right)^2$$

and if $d < n_t$

$$\|v_t\|_2^2 = \sum_i \left(\frac{\sqrt{\sigma_i}(v^\mathsf{T}v + \varepsilon)^{1/2}}{(2\sigma_i(v^\mathsf{T}v + \varepsilon)^{1/2} + \lambda)} \right)^2 \left(2v_i^\mathsf{T} Y_t \right)^2 .$$

This improves the computation on many levels. The computation of the SVD are now done on matrices whose rank is higher, because we are reducing the problem to a smaller dimensionality. This improves numerical precision and speed. Moreover, we can carry out all the T SVD decompositions, the most computing intensive part, in parallel. We can also parallelize up to a certain level the iterations of the C_ε operator, synchronizing the different updates only when a new common value for $\|v\|_2$ is needed, although this is not a significant improvement since all the operations are performed on scalars.

5.3 Comparison With Existing Methods

We can now compare MTFL-GL with other Group Lasso solver, for the Group Lasso part, and with standard MTFL for the feature learning part.

To compute the complexity of MTFL-GL we will first estimate the complexity of the D-step and W-step separately. The D-step requires computing the square root of the matrix $WW^\mathsf{T} + \varepsilon I$. This can be efficiently carried out by considering that we want to avoid computing the intermediate representation. Given a SVD of the W

matrix $U\Sigma V^\mathsf{T}$, we can apply all the spectral operators, such as the ε addition, the trace normalization and the square root, directly to Σ, and compute the final result $U\Sigma' U^\mathsf{T}$. The computational cost of a SVD with modern techniques for a matrix $\mathbb{R}^{d\times T}$ is in the order of $\mathcal{O}(d^2T + T^3)$ [21]. This cost is shared by MTFL-GL, and by the original MTFL. Since the construction of the final matrix requires computing the multiplication of $d \times d$ matrices, and that matrix multiplication is $\mathcal{O}(d^{2.8})$, this is a lower bound for the cost of a single iteration. Luckily, SVD decompositions can be block decomposed, and can enjoy some level of parallelism.

In the original MTFL, the W-step could be decomposed in T separate problems thanks to the separability of the loss function, and of the regularizer for a fixed D. Each subproblem was equivalent to a Ridge regression, that can be efficiently solved through a Cholesky decomposition. Using Lemma 5.2.11, the cost of solving a single task becomes $\mathcal{O}(\min(d^3, n^3))$. All the tasks can run in parallel, to exploit the computational power of distributed systems.

In MTFL-GL, the presence of the square root binds all the tasks together. In particular, when D is diagonal, the W-step can be seen as a Group Lasso problem with a single group with cardinality $d \times T$. This is the reason why we choose to move as much precomputation as possible outside of the C_ε loop, in order to exploit all parallelization possibilities as much as possible. Therefore, by using the eigenvalues based formulation, the cost of a W-step of MTFL-GL reduces to the cost of computing each separate eigenproblem for a cost similar to inversion, and in the order of $\mathcal{O}(\min(d^3, n^3))$. The subsequent iteration to find the norm of the optimal solution operates on scalars, and can be computed efficiently even serially. In the end, the final solution still needs to be computed by Cholesky decomposition, and this brings the computational cost of MTFL-GL to be at least twice as much as the original MTFL, although in the same order of magnitude.

When computing the full feature learning problem there is not any definitive advantage on using MTFL-GL instead of MTFL. The consideration used in Equation (5.2.1) regarding the larger sensibility of MTFL to changes in λ when operating with extremely sparse vectors remains. This reduced sensibility is much more interesting instead in the case when the choice of λ can be tuned theoretically, as it is the case when the algorithm does not learn a feature representation.

The comparison with other Group Lasso algorithms is made more difficult by the lack of guarantees on the convergence rate for MTFL and therefore for MTFL-GL. We can therefore only compare some partial results on computation time of each step. Block Gradient Descent Group Lasso, as presented in Algorithm 3.1, has an inner loop cost of $\mathcal{O}(nd)$, much smaller than the cost of an eigendecomposition. but this cost needs to be considered with the possibility of multiple evaluations in an iteration before reaching convergence. In addition, when $d \gg n$, the $\mathcal{O}(n^3)$ might still be competitive. All algorithms need to compute values for each group or task, but MTFL-GL can parallelize the heaviest computational part. Another interesting

Algorithm 5.3 Block Coordinate Descent Group Lasso

input: X, Y, λ, tol
output: $\widehat{W}$

```
Initialize Ŵ, compute eigendecompositions of X^T X
do
    Ŵ_old = Ŵ
    for i ← 1, ..., d do
        r_{-i} = Y − Σ_{j≠i} X^j w^j^T
        if ||X^i^T r_{-i}||_2 ≤ λ then
            ŵ^i ← 0
        else
            Compute the root δ using Newton's method.
            Compute w^i using δ
        end if
    end for
while ||Ŵ_old − Ŵ||_{2,1} ≥ tol
```

consideration is that the W-step of MTFL-GL can be formulated as a single group Group Lasso problem when $\epsilon \sim 0$, which is often the case, and therefore we can use Block Gradient Descent to obtain an approximated solutions in $\mathcal{O}(ndT)$ time.

A much more similar algorithm for Group Lasso is presented in [44]. In this work the algorithm BCD-GL computes an eigendecomposition for each block, and instead of iterations that cost $\mathcal{O}(nd)$ each, the inner loop is substituted with Newton's method to find the unique root of a scalar polynomial dependent on the values obtained in the eigendecomposition.

Newton's method is extremely fast, and each inner loop is dominated by the initial residual computation. In practice, this method trades the multiple $\mathcal{O}(nd)$ iterations for the cost of an initial eigendecomposition, executed only once. Further analysis of this method could provide possible optimizations for MTFL-GL. In practice, the greatest cost of MTFL-GL is the eigendecompositions that needs to be computed at each iteration. If it was possible to exploit the properties of the D matrix, its s.d.p and diagonality, in order to reduce this cost, a big improvement could be obtained. Some results for SVD can be found in [20], although their main concern is over numerical precision.

Chapter 6

Experiments

We will now analyze the performances of Sparse Fitted Q Iteration on an artificial problem. The goal of these experiments is to verify the actual advantage of the Multi-Task approach versus solving each task independently. In order to do this we will consider the average regret across tasks when compared to the optimal policy. In particular the performance index for the experiments will be

$$\widetilde{R}^{\pi}(x) = \mathop{\mathbb{E}}_{\pi^*}\left[\sum_{i=0}^{N} r_i | x_0 = x\right] - \mathop{\mathbb{E}}_{\pi}\left[\sum_{i=0}^{N} r_i | x_0 = x\right], \widetilde{R}^{\pi} = \frac{1}{n}\sum_{i=1}^{n} \widetilde{R}^{\pi}(x_i).$$

This quantity measures how much worse our performance is compared to the optimal policy while starting from an initial state, and averages this performance across several states. Because we use a multi-task setting, we will consider the average value across tasks

$$\widetilde{R} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{n}\sum_{i=1}^{n} \widetilde{R}^{\pi_t}(x_i). \tag{6.0.1}$$

We will now introduce a detailed description of our experiments.

6.1 Setting

We will test Sparse Fitted Q Iteration in the artificial problem usually called Chain Walk. The agent is placed on a line and needs to reach a goal from a given starting position. The chain is a continuous interval with range $[0, 8]$, and the goal can be situated at any point in the interval $[2, 6]$. We chose to avoid placing the goal at one of the extremes to guarantee that both actions are useful in certain positions. The agent has 2 actions at her disposal, a_1 and a_2, that correspond to a step in each direction. When choosing action a_1 the state of the environment, represented by the agent's position, transitions from x to $x + \epsilon$ where ϵ is some random noise with Gaussian distribution $\mathcal{N}(0, 0.01)$. Given a goal $g = y$, the agent receives a reward 0

for every step, and a reward 1 when the future state x' is close to g, according to the formula $|x' - y| \leq 0.5$.

The dataset is collected in two steps. First, two transitions that end in the goal, starting from a state outside the goal, taking a single step are added to the dataset. These two samples belongs to the two different actions, to uniquely characterize the tasks, and to provide the minimum necessary information to the algorithm. In practice, if we do not guarantee that both actions have at least one sample for the regression SFQI is not well defined. In addition providing reward samples for both actions allows the agent to learn that both actions can be useful. The remaining samples for each task, samples from $2, \ldots, n$ are sampled uniformly from all the states outside the goal. Each sample consists of an initial state randomly selected, then a uniformly random action is selected and one transition is observed, adding the future state and the reward.

The evaluation of the performance is executed by selecting $m \leq n$ samples out of the training set, and running the optimal policy and each candidate policy computed with SFQI form the starting states $\{x_i\}_{i=1}^{m}$. The expected discounted sum of rewards is computed with the result of this additional simulation, and the expectation is approximated with an average over multiple restarts from x_i. This evaluation process is repeated for a series $\{\lambda_i\}_{i=1}^{l}$ of candidate regularization.

The whole evaluation process is repeated multiple times to take into account randomness in the generation of the samples and the results are averaged across runs. Confidence intervals are then computed from such samples. In the end, the λ_i with the best average performance across tasks is chosen.

6.2 Multi-Task Contribution

To test the first contribution of this thesis, we compare the performance of the Multi-Task approach compared to the performances of the single task approach. In the case of SFQI, this translates to using Group Lasso as an approximator for the Fitted Value Iteration algorithm when testing the Multi-Task approach, and the use of Lasso on each separate task for the other. To construct a scenario where the bound of Theorem 4.4.2 is useful, we add noisy, unnecessary features to simulate the introduction of many features that are not relevant to the problem. At this stage, we preserve the group sparsity by simply appending the noise dimensions to the true features. The noise features are uniform, random numbers between $[-0.25, 0.25]$. The true features are 17 uniformly spaced RBFs between $[0, 8]$, with a spacing of 0.5. The RBF are computed as

$$\mathrm{RBF}(x, y) = \exp(-\frac{(x-y)^2}{0.1}).$$

We tested the performance of Group Lasso and Lasso for varying numbers of tasks and noise dimensions. The choice of the candidate values of λ was guided

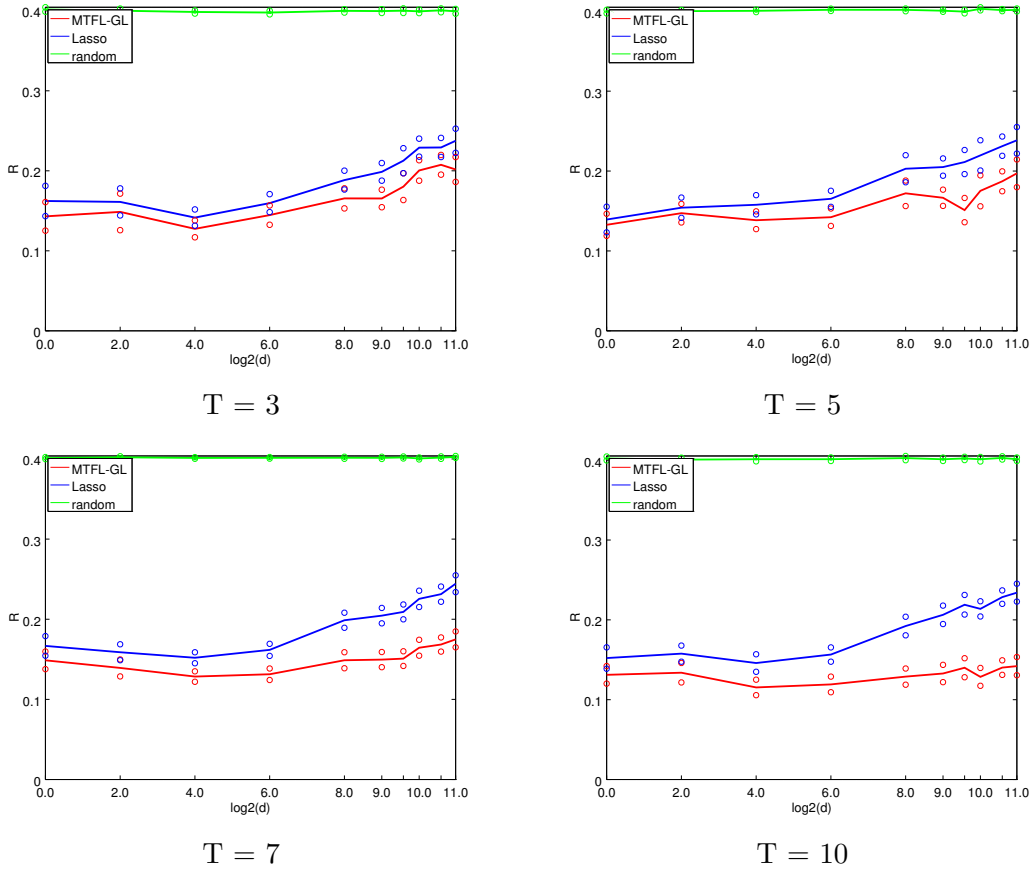


Figure 6.1: Results of Experiments in Section 6.2. On the y axis we have the average regret computed according to Equation (6.0.1). On the x axis we have the total number of dimensions d, including noise dimensions, on a logarithmic scale. For each graph T corresponds to the number of tasks learned at the same time in the experiment.

by the conditions for Theorem 4.3.2. In particular $s = 17$ is the small number of sparse features necessary to represent the function, $n = 30$ is the number of samples, $\log(d) \in [0, 10]$ is the total number of dimensions including the noise and $T \in [3, 10]$ is the number of tasks. Therefore using

$$\frac{2\sqrt{2}\sigma}{\sqrt{30T}}\left(1 + \frac{\log d}{T}\right)^{\frac{1}{2}}$$

we obtain an order of magnitude of 10^{-1} if we consider unit σ. Our candidate λ are therefore chosen as logarithmically evenly distributed between 10^{-3} and 10^{0}.

In Figure 6.1 we report the results of this experiment.

As expected, the Group Lasso solution outperforms Lasso when the number of tasks increases. The stability of the bound is also particularly evident in the $T = 7$ setting, where the performance remains stable until $\frac{\log(d)}{T}$ remains small, but the performance drop for larger numbers of noise dimensions.

6.3 Feature Learning Limits

In this section we test the intuition that the U matrix learned reflects a rotation in the original parameters. That is, we want to test that given a sparse matrix A, and given $Y = XUA$, MTFL will recover the true sparse matrix A. A similar experiment was introduced in [1], but the analysis was different because of three factors. First, in the original paper the method is compared to a standard Ridge regression, which is not the most suitable comparison for a Group Lasso inspired algorithm. We include both Group Lasso and Lasso in addition to Ridge in out MTFL comparison. Second, the U matrix that they expect to learn is diagonal, or in other words their experiment tries to simply do feature selection with MTFL. We will instead use a random U matrix. Third, we will introduce a larger number of noise dimensions to the problem to investigate its limits. In particular the experiments consists of sampling 5 parameters $\{a_1, \ldots, a_5\}$ with $a \sim \mathcal{N}(0, [1, 0.64, 0.49, 0.36, 0.25])$. We sample T vectors, padding them with zeroes such that $a_t \in \mathbb{R}^d$. This is the sparse representation we want to recover. The A matrix is multiplied by a random orthonormal matrix U, obtained by orthonormalization of random uniform samples, to obtain a W matrix that is not group sparse, and will therefore be a hard task for Lasso and Group Lasso, while allowing MTFL to reconstruct the original matrix. For each task, 10 $x_i \in \mathbb{R}^d$ uniformly random samples between $[0, 1]$ are selected to form the training set. 100 and 200 similar samples are drawn respectively for the validation set and test set. Finally, the target samples are computed as $y_i = x_i w_t + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.01)$.

The results of the experiments are reported in Figure 6.2, for various numbers of unnecessary dimensions d, at the variation of the number of tasks. The quality of the solution recovered is measured as the Mean Residual Sum of Squares averaged across tasks, defined as

$$\frac{1}{T} \sum_{t=1}^{T} \frac{1}{n} \sum_{i=1} n(y_i - x_i w_t)^2. \tag{6.3.1}$$

As expected MTFL outperforms the other methods. More importantly, when the number of noise dimensions is not too high we have a clear improvement with the introduction of additional features. Comparing the results for $d = 25$ and $d = 100$, we see that the error increases with the number of noise dimensions and decreases with the number of tasks. The error increases more quickly than the $\log(d)/T$ ration that Group Lasso had for the sparse setting. This is a reasonable conjecture considering that the task of learning d^2 features is more complicated than simply learning the sparse representation. This is easily visualized in Figure 6.3, where we plot the error for a fixed number of tasks, 100, while the number of noise dimensions grows.

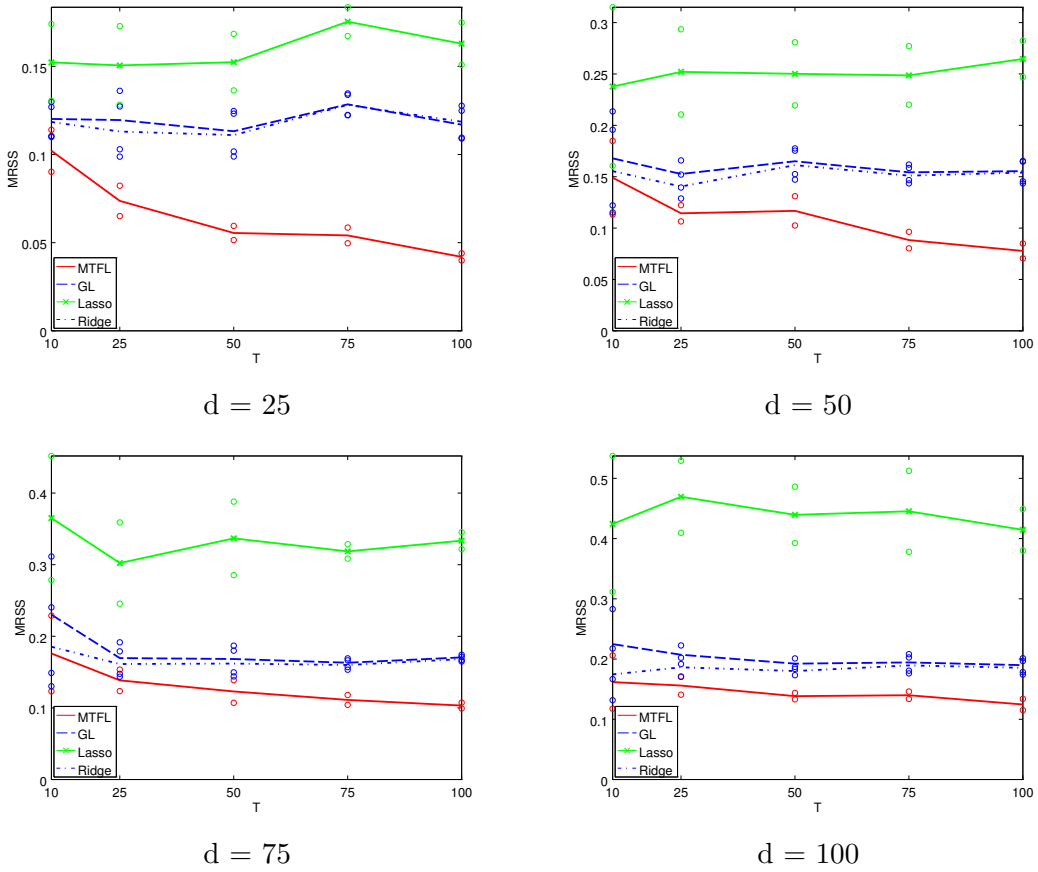


Figure 6.2: Results of Experiments in Section 6.3. On the y axis we have the Mean Residual Sum of Squares, computed according to Equation (6.3.1). On the x axis we have the number of tasks learned at the same time T. For each graph d corresponds to the total number of dimensions used for each experiment, including noise dimensions.

6.4 Feature Learning Contribution

In this setting we test the capability of MTFL-GL to efficiently learn a good feature representation to compensate offsets in the group sparsity of the MDPs. The only modification w.r.t. the previous experiment is in the construction of the features. In particular, we construct a random U matrix, and use it to multiply the transitions together with the noise dimensions $x_i \in \mathbb{R}^d$. We expect MTFL to learn a matrix W, such that $W = U^\mathsf{T} A$, where A is the original matrix that ignored the noise dimensions that would have been obtained in Section 6.2. If such learning succeeds, the new actions will be chosen on the base of $XUU^\mathsf{T}A$, which is the optimal solution. The results for various numbers of tasks are reported in Figure 6.4. As we can see, increasing the number of tasks improves the resistance of MTFL to the number of unnecessary dimensions, while the rotation introduces enough distortion to cause GL to perform worse even in low dimensional settings. Moreover there seems to be no

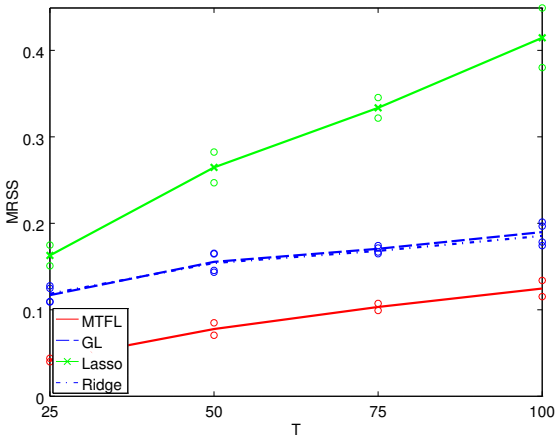
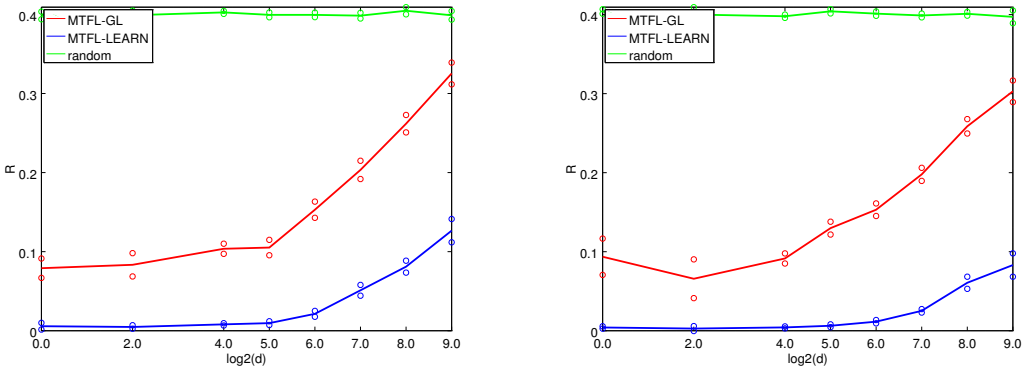


Figure 6.3: Results of Experiments in Section 6.3, MRSS computed while learning 100 tasks simultaneously, at the variation of d, the total number of dimensions including unnecessary ones.



T = 100 T = 200

Figure 6.4: Results of Experiments in Section 6.4 On the y axis we have the average regret computed according to Equation (6.0.1). On the x axis we have the total number of dimensions d, including noise dimensions, on a logarithmic scale. For each graph T corresponds to the number of tasks learned at the same time in the experiment.

improvement to introducing more tasks, as GL will be driven by the s factor and will not depend on the number of tasks. It is interesting to add to this last comment the insight that s is instead growing exponentially, if the U matrix mapped exactly the $\mathbb{R}^s$ original space to $\mathbb{R}^d$. In particular the performance of GL decreases almost linearly in the exponential number of features. It is possible that the linear features produced by the rotation can be still grouped in some group sparse sense, although larger than the optimal one. This is because the combination of true features with noise might produce features that are still informative.

Chapter 7

Conclusions

We will now provide some final considerations on the results achieved in this thesis and possible future developments.

7.1 Contributions

In the introduction of this thesis, the mission statement for this work was to investigate the consequences of group sparsity in an RL setting. The exact definition of group sparsity that we decided to use throughout the thesis is that of a shared, low-dimensional representation that is common across a set of tasks. This makes our work one of the first results in a joint setting that draws heavily from RL, optimization and statistics. In particular in the RL setting the multi-task approach has started to gain more popularity only after the Transfer Learning literature had reached a good level of maturity. Optimization, and especially optimization applied to statistics in the setting of regularized regression, is instead a popular field. Many recent developments related to algorithms that compute group sparse solutions contributed to the background of this thesis.

Among the many possible combinations of RL and optimization methods that could prove useful in a group sparse setting, we chose to integrate FVI with a regularized linear approximator. The simplicity of linear approximators allows for excellent interpretability, and has also received extensive interest regarding the quality of the approximation. This results led us to choose a Group Lasso problem as the formalization of our group sparse assumption, in order to build results valid in an RL settings equivalents to those already obtained for the optimization problem. The contribution related to this goal is the performance bounds provided in Chapter 4. The bound shows clearly that the Multi-Task approach can provide better guarantees compared to solving each task separately, and that the sparsity assumption used in previous single-task settings extends nicely to the group sparse setting. As far as we know, SFQI, is one of the first RL algorithm to exploit both the multi-task properties of a problem, as well as their sparsity. This is clear in the comparison with other,

single-task methods such as Lasso-TD, that do not offer generalization guarantees, or Kernel FQI, that does not have the same interpretability as parametric linear approximators.

The theoretical results provided in this thesis also gave us a useful indication to choose extensions to the method. The use of a multi-task approach proves in Theorem 4.4.2 that a large number of dimension can be managed as long as new tasks are added. But if the inclusion of a new tasks increases the s factor, or in other word if the new task does not share features with the previous, the method will perform poorly. This drawback could be mitigated by recovering a group sparse solution, or in other word learn a suitable feature representation that would allow for the group sparsity to emerge. Our solution was to extend the Group Lasso setting to a larger Multi Task Feature Learning problem, with a small modification that included exactly Group Lasso as a particular case, but led to several new algorithmic problems. These new challenges have been efficiently solved in MTFL-GL. In particular the introduction of feature learning does not imply a much larger computational cost, because as we saw in its derivation, MTFL-GL can still obtain good performances compared to other Group Lasso solvers. On the other hand, as proved experimentally, the introduction of feature learning does provide a better performance when the conditions of group sparsity are not exactly met.

Finally, the experimental section of this thesis provides a rigorous validation on a simple artificial problem of the new theoretical and algorithmic results introduced. While the results obtained for the group sparse setting confirm that in practice the theoretical bounds are reflected in real problems, the new experiments on feature learning provide insights on this alternative, more complex approach. The extended algorithm had no guarantee, so we provided an initial experimental estimate of the complexity of learning these features in a Supervised Learning setting, and then tested it on a more RL oriented task. Overall the experiments shows that SFQI is a promising method, capable of providing a performance improvement over simple sparse regularization.

7.2 Future Developments

Based on the theoretical results several interesting problems can be derived

Weaker assumptions One of the key assumptions for the validity of our bound is Assumption 4.3.1, the Restricted Eigenvalues assumption. In the Lasso regression literature, the RE assumption is one of the members of a small family of assumptions, some stronger some weaker. A comprehensive list is given in [60]. For example Theorem 3.3.1, derived for Lasso-TD, uses the weaker assumption called Compatibility Assumption. One interesting line of research is to test whether weaker assumptions from the Lasso literature can be adapted to hold for Group Lasso settings.

Active Learning Another possible development is hinted by the content of the assumption. A good quality representation depends on a handful of samples in a low dimensional setting. It would be interesting to develop Active Learning techniques that can provide guarantees on the satisfaction of this assumption. Active Learning [50] is the framework in ML that covers all kinds of algorithms interested in collecting good quality data for the learning process.

The other original contribution of this thesis is the derivation of MTFL-GL, an original Group Lasso solver based on MTFL. Because computing Group Lasso solutions is a problem that still receives attention in the optimization community, it might be interesting to analyze its performances. Two possible paths are immediately interesting

Convergence Rate Most optimization algorithms, such as the Newton method, give guarantees on the convergence rate to the solution. It would be interesting to perform a real comparison to obtain similar results for MTFL-GL. Such results could be probably extended to MTFL.

Precomputations The main drawback of our algorithm is the need to recompute an eigendecomposition at each step. Solving this problem using partially precomputed solution would drastically improve performance. Partial results regarding SVD computations are presented in [20] and could provide useful information for this problem.

The main assumption of this work is the underlying group sparsity of the group of tasks. With the introduction of MTFL-GL we tried to bridge the gap between the theoretical results and an extension capable of coping with a problem that is not group sparse but can be represented as one. Early experimental results show that such a gap is not trivial to fill, and that the resulting performances are influenced by a number of factors. Possible directions are

Additional Experiments The first direction of research is surely to perform more extensive experiments. The class of representations that MTFL can recover is not universal, so it would be important to find classes of problems that can be recovered and classes that defeat the algorithm. Moreover additional experimental analysis can clarify the underlying process of feature learning, giving more insight in the number of samples and tasks needed to correctly recover the sparse representation.

Alternative Formulations If in some cases the MTFL framework is not capable of providing good performances, other candidates might provide good alternatives. In our thesis we limited the representation learning as an optimization between features. That excludes more complicated relationships across tasks. From the ML field, [66] introduces a more general, non-convex setting that uses an

additional matrix in the optimization process, in order to learn relationships among tasks that are independent of feature representation. The regression field provides many alternative extensions to Group Lasso, such as the already mentioned Sparse Group Lasso [51] and Graph Lasso [26].

Appendix A

Norms

A.1 Vector Norm

We will report some well-known definitions about vector norms for the reader.

Definition A.1.1. *Given a vector subspace $\mathbb{S}^d$ defined over real numbers, a norm is a function $\|\cdot\| : \mathbb{S}^d \to \mathbb{R}$, that for any vectors $x, y \in \mathbb{S}$, satisfies the following properties:*

$$\begin{aligned} &\|x\| \geq 0, \|x\| = 0 \quad \textit{iff} \quad x = 0 && \textit{definite positiveness,} \\ &\|ax\| \leq |a|\|x\| && \textit{absolute homogeneity,} \\ &\|x + y\| \leq \|x\| + \|y\| && \textit{triangular inequality.} \end{aligned}$$

Using this definition we can define the generic ℓ_p norm, with $1 \leq p < \infty$ is

Definition A.1.2. *The ℓ_p norm of a vector, with ℓ_p with $1 \leq p < \infty$ is defined as*

$$\|w\|_p = \left(\sum_{i=1}^{d} (|w_i|^p) \right)^{1/p}$$

From this definition we obtain the following definitions

Definition A.1.3. *The ℓ_2 norm of a vector is defined as*

$$\|w\|_2 = \sum_{i=1}^{d} |w_i|$$

Definition A.1.4. *The ℓ_2 norm of a vector is defined as*

$$\|w\|_2 = \left(\sum_{i=1}^{d} \left(|w_i|^2\right) \right)^{1/2} = \left(w^\mathsf{T} w\right)^{1/2}$$

The special cases ℓ_0 and ℓ_∞ are instead defined as

Definition A.1.5. *Given an indicator function $I(x) = 1$ iff $x \neq 0$ and 0 otherwise, the ℓ_0 norm of a vector is defined as*

$$\|w\|_2 = \sum_{i=1}^{d} I(w_i)$$

Definition A.1.6. *The ℓ_∞ norm of a vector is defined as*

$$\|w\|_\infty = \max_{i=1}^{d} |w_i|$$

A.2 Matrix Norms

Definition A.2.1. *Given a matrix $W \in \mathbb{R}^{d \times T}$, composed by d rows w^i, the $\ell_{2,1}$ norm of the matrix is defined as*

$$\|W\|_{2,1} = \sum_{i=1}^{d} \left\|w^i\right\|_2$$

Bibliography

[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[2] Andreas Argyriou, Charles A Micchelli, and Massimiliano Pontil. Learning convex combinations of continuously parameterized basic kernels. In *Learning Theory*, pages 338–352. Springer, 2005.

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

[4] Dimitri P. Bertsekas. *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[5] Peter J Bickel, Ya'acov Ritov, and Alexandre B Tsybakov. Simultaneous analysis of lasso and dantzig selector. *The Annals of Statistics*, pages 1705–1732, 2009.

[6] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[7] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[8] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning.

[9] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[10] Rémi Coulom. Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In *Proceedings of the 13th International Conference on Algorithmic Learning Theory*, ALT '02, pages 403–414, London, UK, 2002. Springer-Verlag.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[12] L.E. Dubins and L.J. Savage. *How to gamble if you must: inequalities for stochastic processes*. McGraw-Hill series in probability and statistics. McGraw-Hill, 1965.

[13] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[14] Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *ICML*, pages 154–161. AAAI Press, 2003.

[15] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 201–208, New York, NY, USA, 2005. ACM.

[16] Damien Ernst, Pierre Geurts, Louis Wehenkel, and Michael L Littman. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(4), 2005.

[17] Amir Massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *Proceedings of the 2009 conference on American Control Conference*, ACC'09, pages 725–730, Piscataway, NJ, USA, 2009. IEEE Press.

[18] Eric Gautier and Alexandre B Tsybakov. High-dimensional instrumental variables regression and confidence sets. *arXiv preprint arXiv:1105.2454*, 2011.

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

[20] Gene Golub, Knut Solna, and Paul Van Dooren. Computing the svd of a general matrix product/quotient. *SIAM Journal on Matrix Analysis and Applications*, 22(1):1–19, 2000.

[21] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[22] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[23] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[24] Matthew W Hoffman, Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Regularized least squares temporal difference learning with nested ℓ_2 and ℓ_1 penalization. In *Recent Advances in Reinforcement Learning*, pages 102–114. Springer, 2012.

[25] Vasile I Istratescu. *Fixed point theory: an introduction*, volume 7. Springer, 2001.

[26] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. Group lasso with overlap and graph lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 433–440. ACM, 2009.

[27] Tobias Jung and Daniel Polani. Least squares svm for least squares td learning. In *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy*, pages 499–503, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

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

[29] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press, 2000.

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

[31] Michail G. Lagoudakis, Ronald Parr, and Michael L. Littman. Least-squares methods in reinforcement learning for control. In *SETN*, volume 2308 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2002.

[32] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

[33] Alessandro Lazaric, Mohammad Ghavamzadeh, et al. Bayesian multi-task reinforcement learning. In *ICML-27th International Conference on Machine Learning*, pages 599–606, 2010.

[34] Manuel Loth, Manuel Davy, and Philippe Preux. Sparse temporal difference learning using lasso. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 352–359. IEEE, 2007.

[35] Karim Lounici, Massimiliano Pontil, Sara Van De Geer, Alexandre B Tsybakov, et al. Oracle inequalities and optimal inference under group sparsity. *The Annals of Statistics*, 39(4):2164–2204, 2011.

[36] Pascal Massart and Caroline Meynet. An l1-oracle inequality for the lasso. *arXiv preprint arXiv:1007.4791*, 2010.

[37] Charles A Micchelli, Jean Morales, and Massimiliano Pontil. A family of penalty functions for structured sparsity. In *NIPS*, pages 1612–1623, 2010.

[38] Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer ' s theorem , feature maps , and smoothing. *East*, 4005(Lecture Notes in Computer Science):154–168, 2006.

[39] Thomas P Minka. Old and new matrix algebra useful for statistics. *See www. stat. cmu. edu/minka/papers/matrix. html*, 2000.

[40] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Mach. Learn.*, 49(2-3):291–323, November 2002.

[41] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine learning*, 49(2-3):291–323, 2002.

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

[43] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, NY, April 1994.

[44] Zhiwei Qin, Katya Scheinberg, and Donald Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, 5(2):143–169, 2013.

[45] Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[46] Martin Riedmiller. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

[47] Sheldon M. Ross. *Introduction to Stochastic Dynamic Programming: Probability and Mathematical*. Academic Press, Inc., Orlando, FL, USA, 1983.

[48] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[49] Bernhard Scholkopf and Alex Smola. Learning with kernels, 2002.

[50] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52:55–66, 2010.

[51] Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

[52] 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. MIT Press, 1995.

[53] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[54] James Stewart. *Multivariable calculus: concepts and contexts*. Cengage Learning, 2009.

[55] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[56] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[57] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[58] Andreĭ Tikhonov. *Solutions of ill-posed problems*.

[59] Stephan Timmer and Martin Riedmiller. Fitted q iteration with cmacs. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 1–8. IEEE, 2007.

[60] Sara A Van De Geer, Peter Bühlmann, et al. On the conditions used to prove oracle results for the lasso. *Electronic Journal of Statistics*, 3:1360–1392, 2009.

[61] C Watkins. *Learning from delayed rewards*. PhD thesis, Kings College, 1989.

[62] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

[63] 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. ACM, 2007.

[64] Huizhen Yu and Dimitri P Bertsekas. Convergence results for some temporal difference methods based on least squares. *Automatic Control, IEEE Transactions on*, 54(7):1515–1531, 2009.

[65] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[66] Yi Zhang and Jeff G Schneider. Learning multiple tasks with a sparse matrix-normal penalty. In *NIPS*, pages 2550–2558, 2010.