POLITECNICO DI MILANO

School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science Degree in Computer Science and Engineering

Compatible Reward Inverse Reinforcement Learning

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

Supervisor: Prof. Marcello Restelli
Co-supervisor: Dott. Matteo Pirotta

Author:
Alberto Maria Metelli, 850141

Academic Year 2016-2017

A tutte le persone che mi hanno sostenuto

Abstract

Sequential decision making problems arise in a variety of areas in Artificial Intelligence. Reinforcement Learning proposes a number of algorithms able to learn an optimal behavior by interacting with the environment. The major assumption is that the learning agent receives a reward as soon as an action is performed. However, there are several application domains in which a reward function is not available and difficult to estimate, but samples of expert agents playing an optimal policy are simple to generate.

Inverse Reinforcement Learning (IRL) is an effective approach to recover a reward function that explains the behavior of an expert by observing a set of demonstrations. Most of the classic IRL methods, in addition to expert's demonstrations, require sampling the environment in order to compute the optimal policy for each candidate reward function. Furthermore, in most of the cases, it is necessary to specify a priori a set of engineered features that the algorithms combine to single out the reward function.

This thesis is about a novel model-free IRL approach that, differently from most of the existing IRL algorithms, does not require to specify a function space where to search for the expert's reward function. Leveraging on the fact that the policy gradient needs to be zero for any optimal policy, the algorithm generates a set of basis functions that span the subspace of reward functions that make the policy gradient vanish. Within this subspace, using a second-order criterion, we search for the reward function that penalizes the most a deviation from the expert's policy.

After introducing our approach for finite domains, we extend it to continuous ones. The proposed approach is compared to state-of-the-art IRL methods both in the (finite) Taxi domain and in the (continuous) Linear Quadratic Gaussian Regulator and Car on the Hill environments. The empirical results show that the reward function recovered by our algorithm allows learning policies that outperform both behavioral cloning and those obtained with the true reward function, in terms of learning speed.

Estratto in Lingua Italiana

Nell'ambito dell'Intelligenza Artificiale numerosi problemi richiedono che un agente operi decisioni in sequenza senza ricevere un riscontro immediato. In questi problemi, che si presentano ad esempio nel campo del controllo automatico, della robotica e della finanza, l'agente deve tenere conto dell'effetto ritardato delle sue decisioni, nonché del possibile non determinismo dell'ambiente. Naturalmente, al fine di pianificare una decisione efficace l'agente deve fare riferimento a qualche nozione di ottimo, tipicamente codificata mediante una funzione di utilità che associa ad uno specifico comportamento un numero che ne rappresenta l'aderenza all'obiettivo implicito dell'agente.

In questa tesi consideriamo il caso in cui la dinamica dell'ambiente e il comportamento dell'agente soddisfano la proprietà di Markov, cioè il prossimo stato dell'ambiente è indipendente dagli stati passati, noti che siano lo stato corrente e l'azione compiuta dall'agente; inoltre la decisione operata dall'agente dipende soltanto dallo stato corrente. Questa classe di problemi viene modellata efficacemente dai *Processi Decisionali di Markov.* La funzione di utilità, in questi casi, è definita in termini del *rinforzo immediato.* L'agente non appena compie un'azione sull'ambiente riceve un riscontro, il rinforzo. La funzione di utilità viene definita come il *rinforzo cumulativo*, cioè la somma (eventualmente scontata) dei rinforzi immediati raccolti dall'agente durante la sua esperienza nell'ambiente. Chiaramente, la scelta della funzione di rinforzo ha un impatto determinante sull'efficacia e sull'efficienza degli algoritmi impiegati per apprendere un comportamento ottimo.

Qualora il modello del rinforzo sia disponibile, l'*Apprendimento per Rinforzo* propone numerosi algoritmi in grado di apprendere la politica ottima, ovvero una prescrizione di azioni che massimizza il rinforzo cumulativo. Inoltre, se il modello dell'ambiente è noto, è possibile impiegare un approccio basato sulla *Programmazione Dinamica.* Tuttavia, nella maggior parte dei casi l'agente deve interagire con l'ambiente allo scopo di apprenderne la dinamica, implicitamente o esplicitamente. La letteratura si è occupata di questo aspetto estensivamente, proponendo algoritmi efficaci applicati oggigiorno ad una vasta gamma di settori, dalla robotica alla finanza, dalla logistica alle reti di telecomunicazione e all'intelligenza ambientale. Assumere che una funzione di rinforzo sia disponibile è tuttavia irrealistico in molti dei domini di interesse. Ad esempio, risulta particolarmente complesso progettare una funzione di rinforzo per descrivere l'attività di un guidatore; sarebbe infatti necessario definire quanto uno specifico comportamento sia conforme con la nozione intuitiva di “buon guidatore”.

Quando la funzione di rinforzo non è disponibile o difficile da progettare, ma risulta agevole generare esempi di esecuzione di un esperto (cioè di un agente che esegue la politica ottima), è possibile fare uso di tecniche di *Apprendimento per Imitazione*. In questo ambito è possibile distinguere tra due categorie di metodi: *Behavioral Cloning* e *Apprendimento per Rinforzo Inverso*. La prima si pone l’obiettivo di imparare la politica ottima mediante un approccio basato sull’apprendimento supervisionato; al contrario, l’apprendimento per rinforzo inverso è teso a ricostruire una funzione di rinforzo che rende ottimo il comportamento dell’esperto. Quest’ultimo, diversamente dal behavioral cloning, che è tipicamente più semplice, assicura buone proprietà di generalizzazione e trasferibilità in quanto il rinforzo può essere impiegato in nuovi ambienti. Pertanto, l’apprendimento per rinforzo inverso permette di ricostruire la politica ottima a posteriori.

Stato dell’arte

Una vasta gamma di approcci per l’apprendimento per rinforzo inverso sono stati proposti nella letteratura. Gli algoritmi basati sulla *feature expectation* [1, 113, 112] sono diretti a ricostruire una rappresentazione della funzione di rinforzo che induca una politica ottima simile al comportamento dell’esperto. Questi algoritmi, tuttavia, risultato inefficienti in termini di numero di campioni richiesti e necessitano di avere accesso all’ambiente per determinare la politica ottima per ogni funzione di rinforzo considerata. Recenti progressi hanno permesso di rimuovere questa limitazione: GIRL [94] fa uso del policy gradient [111] per determinare il valore dei parametri di una classe parametrica di funzioni di rinforzo. Inoltre, l’apprendimento per rinforzo inverso soffre del problema dell’ambiguità del rinforzo, cioè esistono infinite di funzioni di rinforzo per lo stesso problema che rendono l’esperto ottimo (incluse alcune funzioni banali). Allo scopo di fornire una risposta a questo problema, in [125] il principio della massima entropia è utilizzato in modo da estrarre un’unica funzione di rinforzo. L’algoritmo, tuttavia, richiede la conoscenza del modello dell’ambiente per problemi non deterministici.

Gli approcci sinora considerati condividono la necessità di fornire a priori un insieme di feature che definiscano lo spazio dei rinforzi (ad es. la posizione dell’agente o la distanza dall’obiettivo). Come accade nell’apprendimento supervisionato, la scelta dello spazio di ipotesi è fondamentale per il successo dell’algoritmo. La letteratura recente si è concentrara nell’includere la costruzione delle feature negli algoritmi di apprendimento per rinforzo inverso. Alcuni lavori, come [35, 45] sfruttano le potenzialità del deep learning per estrarre implicitamente le feature in maniera black-box e costruire il rinforzo mediante approssimatori non lineari. Il problema dell’estrazione delle feature è stato affrontato esplicitamente soltanto da FIRL [64], un algoritmo che costruisce iterativamente le feature allo scopo di descrivere in maniera più accurata quelle regioni dell’ambiente ove il precedente insieme di feature risultava troppo impreciso. L'algoritmo, tuttavia, richiede la conoscenza del modello di transizione dell'ambiente.

A parte rare eccezioni [ad es. 9], le applicazioni più promettenti dell'apprendimento per rinforzo inverso ricadono nel settore della locomozione automatica [ad es. 82, 113, 101, 125, 76, 100]. Nelle applicazioni reali la trasferibilità del rinforzo diviene di vitale importanza. Ad esempio, nella guida automatica, ci si aspetta che l'agente si comporti in maniera adeguata anche se la configurazione del traffico non è mai stata sperimentata dall'esperto. Tuttavia, questo aspetto è stato finora affrontato soltanto per problemi semplici [64, 65] . Questi lavori dimostrano che una scelta non accurata delle feature può compromettere la possibilità di trasferire il rinforzo a nuovi domini.

Contributi

Nonostante negli ultimi anni l'apprendimento per rinforzo inverso abbia vissuto un notevole sviluppo, alcune domande rimangono aperte e alcuni problemi irrisolti. Crediamo che al fine di poter applicare le tecniche di apprendimento per rinforzo inverso a domini di interesse reale sia indispensabile disporre di un algoritmo in grado di produrre funzioni di rinforzo trasferibili, richiedendo soltanto un insieme di traiettorie generate da un dimostratore esperto. L'approccio che proponiamo in questa tesi rappresenta un tentativo di fornire una risposta congiunta al problema dell'estrazione delle feature e della costruzione della funzione di rinforzo, senza necessità di disporre del modello dell'ambiente. Ciò rappresenta una novità per la letteratura del settore in quanto, ad eccezione dei modelli basati sul deep learning, nessun algoritmo è in grado di costruire feature senza richiedere la conoscenza del modello di transizione.

Il principale contributo di questa tesi è algoritmico e sperimentale. Proponiamo un algoritmo CR-IRL (*Compatible Reward Inverse Reinforcement Learning*) capace di costruire uno spazio di approssimazione per il rinforzo sfruttando una condizione del primo ordine sul policy gradient ed estrarre una funzione di rinforzo, all'interno di questo spazio, mediante un criterio del secondo ordine sul policy Hessian. L'algoritmo richiede una rappresentazione parametrica della politica dell'esperto che può essere ottenuta mediante behavioral cloning. Naturalmente, la scelta della classe di politiche influenza l'estrazione dello spazio di approssimazione. Allo scopo di formalizzare l'abilità di una politica di ridurre la dimensione dello spazio dei rinforzi ottimi, viene introdotta la nozione di *rango di una politica* (policy rank), del quale è fornito un bound nel caso di problemi finiti. Sfruttando il fatto che il policy gradient deve annullarsi per tutte le Q-function massimizzate dall'esperto, ne viene estratto lo spazio di approssimazione, dal quale è possibile derivare lo spazio di approssimazione per la funzione di rinforzo. Tale trasformazione può essere effettuata invertendo l'equazione di Bellman, quando il modello dell'ambiente è noto, oppure mediante reward shaping. Quest'ultima possibilità, che non richiede la conoscenza del modello di transizione, comporta una riduzione della dimensionalità dello spazio di approssimazione, tuttavia essa definisce lo spazio delle advantage function che, come noto, favoriscono le proprietà di convergenza degli algoritmi di apprendimento. Allo scopo di individuare una funzione di rinforzo all'interno dello spazio di approssimazione, viene impiegato un criterio del secondo ordine basato sul policy Hessian. L'obiettivo è identificare la funzione di rinforzo che penalizza il più possibile le deviazioni dalla politica dell'esperto. Analiticamente il problema può essere formulato in termini di programmazione multi-obiettivo, da essa vengono derivati alcuni criteri di ottimalità mono-obiettivo ed infine introdotta un'euristica che fornisce un'approssimazione della funzione di rinforzo ottima.

La valutazione sperimentale è effettuata confrontando la velocità di apprendimento della funzione di rinforzo prodotta da CR-IRL e le funzioni di rinforzo generate da classici algoritmi di apprendimento per rinforzo inverso. Tale comparazione è effettuata sul dominio finito del Taxi e sui domini continui Regolatore Lineare-Quadratico e Car on the Hill. Le funzioni di rinforzo prodotte da CR-IRL permettono non soltanto di imparare politiche ottime, ma mostrano una velocità di apprendimento superiore rispetto alla funzione di rinforzo originale del problema. Inoltre le politiche apprese con CR-IRL evidenziano una performance superiore rispetto al puro behavioral cloning, nonostante CR-IRL ne faccia uso. Crediamo che ciò rappresenti uno degli aspetti chiave di CR-IRL, immediata conseguenza dell'utilizzo congiunto dell'apprendimento per rinforzo inverso e del behavioral cloning.

Struttura della tesi

I contenuti della tesi sono organizzati in sei capitoli. Il Capitolo 1 presenta le motivazioni, gli obiettivi e i contributi della tesi. Il Capitolo 2 fornisce una visione di alto livello dei processi decisionali di Markov e dell'apprendimento per rinforzo. Vengono introdotte le principali definizioni e algoritmi per poi concentrarsi su quegli aspetti maggiormente utilizzati nel seguito. Il Capitolo 3 illustra il panorama dello stato dell'arte degli algoritmi di apprendimento per rinforzo inverso. La presentazione è organizzata per categorie di algoritmi accomunati da similarità negli approcci. L'obiettivo di questo capitolo è guidare il lettore nella comprensione delle motivazioni che hanno indotto questo lavoro. Il Capitolo 4 è dedicato alla presentazione di CR-IRL. Si parte dalla discussione della costruzione di una rappresentazione parametrica della politica dell'esperto per poi passare alle condizioni del primo ordine sul policy gradient finalizzate all'estrazione dello spazio di approssimazione della Q-function. A questo punto sono presentati i due approcci che permettono di ottenere lo spazio delle feature della funzione di rinforzo. Infine, è discusso il processo di selezione del rinforzo basato sui criteri del secondo ordine applicati al policy Hessian. Il Capitolo 5 fornisce la valutazione sperimentale di CR-IRL. Vengono introdotte le metriche utilizzate per valutare la performance degli algoritmi per poi presentare nel dettaglio esperimenti. Infine, nel Capitolo 6 riassumiamo i risultati ottenuti e proponiamo possibili estensioni di questo lavoro. L'Appendice A presenta brevemente la notazione matriciale per i processi decisionali di Markov finiti. L'Appendice B infine fornisce le dimostrazioni e derivazioni omesse nel testo della tesi.

Ringraziamenti

Desidero ringraziare il Prof. Marcello Restelli, relatore, e il Dott. Matteo Pirotta, correlatore; a loro si deve il mio primo contatto con il mondo della ricerca. La loro costante presenza e l'assiduo contributo creativo hanno reso questa esperienza una preziosa occasione di crescita personale e professionale.

Ringrazio i miei più cari amici, fuori e dentro l'università, per i piacevoli momenti di svago e confronto. Un pensiero particolare va agli "inquilini" del PoliCloud.

Il più grande ringraziamento va a tutta la mia famiglia, specialmente ai miei genitori. Grazie per avermi sempre sostenuto nelle mie scelte e per avermi insegnato che non c'è cosa più bella che perseguire le proprie passioni.

Alberto
Milano, 27 luglio 2017

Mathematical Notation

Vectors are denoted by lower case bold Roman letters, such as $\mathbf{x}$. All vectors are assumed to be column vectors. The i-th component of vector $\mathbf{x}$ is indicated with x_i. The notation $\mathbf{x}^T = (x_1, x_2, ..., x_n)$ denotes a row vector with n elements, while the corresponding column vector is written as $\mathbf{x} = (x_1, x_2, ..., x_n)^T$. Uppercase bold Roman letters, such as $\mathbf{M}$, denote matrices. The element of matrix $\mathbf{M}$ in position (i, j) is indicated with M_{ij}. The only exception is for the matrix representing the policy of an MDP that remains bold lowercase $\boldsymbol{\pi}$. A diagonal matrix built starting from vector $\mathbf{x}$ is denoted by $\mathrm{diag}(\mathbf{x})$. The trace of a matrix $\mathbf{M}$ is denoted as $\mathrm{tr}(\mathbf{M})$. The identity matrix is indicated with $\mathbf{I}$, while the single-entry matrix with 1 in position (i, j) and all zeros elsewhere is indicated with $\mathbf{J}^{ij}$. A tensor (at least 3-dimensional) is indicated with the bold Sans-Serif uppercase letter T. We will treat them simply as multidimensional arrays, thus its component in position $(i_1, i_2, ..., i_n)$ is given by $\mathsf{T}_{i_1 i_2 \dots i_n}$. We employ a Matlab-like notation to index rows and columns of matrices: given a matrix $\mathbf{M}$, its i-th row is $\mathbf{M}_{i:}$, while its j-th column is $\mathbf{M}_{:j}$. The same notation extends to tensors, e.g., given a 3-dimensional tensor T of shape $n \times m \times k$, $\mathsf{T}_{ij:}$ is a k-dimensional vector whereas $\mathsf{T}_{:i:}$ is an $n \times k$ matrix. The symbol $\geq$ when applied to properly sized vectors or matrices means component-wise inequality, whereas the expression $\mathbf{A} \succeq \mathbf{B}$ where $\mathbf{A}$ and $\mathbf{B}$ are square matrices of the same size, means that $\mathbf{A} - \mathbf{B}$ is positive semidefinite. If P is a logical predicate, the symbol $\mathbb{1}(P)$ is the indicator function which is one if P is true, zero otherwise. When two functions are directly proportional one another, i.e., $f = kg$, we use the notation $f \propto g$.

The gradient of function f w.r.t. a vector of independent variables $\mathbf{x}$ is indicated with $\nabla_{\mathbf{x}} f(\mathbf{x})$; it is assumed to be a column vector. The Hessian matrix of function f w.r.t. a vector of independent variables $\mathbf{x}$ is indicated with $\mathcal{H}_{\mathbf{x}} f(\mathbf{x})$. Functionals and operators are indicated with calligraphic letters e.g., $\mathcal{F}, \mathcal{O}$, their application is denoted by square brackets e.g., $\mathcal{F}[f]$, $\mathcal{O}[f]$.

The set of all probability measures over a measurable space $(\Omega, \mathcal{B}(\Omega))$ is denoted by $\Delta(\Omega)$, where Ω is the sample space and $\mathcal{B}(\Omega)$ is the Borel σ-algebra over Ω. A probability measure is indicated with the calligraphic letter e.g., $\mathcal{P}$. The symbol $\mathbb{P}$ is used to indicate a generic probability when there is no need to define the probability space explicitly. The probability density function of a random variable x is indicated with the lowercase letter e.g., p_x, the subscript is often omitted. There are two exceptions to this rule throughout the thesis: the probability density function of the transition model and the one of the reward function of an MDP, they are indicated with the uppercase letters P and R respectively. When the density function of a random variable x is conditioned to the value of another random variable y we adopt the notation $p_x(\cdot|y)$, the same notation holds for probability measures. The expected value of a function $\mathbf{f}(x,y)$ taken w.r.t. random variable (or random vector) x distributed according to the probability density p is indicated as $\mathbb{E}_{x\sim p}[\mathbf{f}(x,y)]$, whereas the covariance matrix is denoted as $\mathrm{Var}_{x\sim p}[\mathbf{f}(x,y)]$. Whenever there is no ambiguity, the latter are abbreviated with $\mathbb{E}_x[\mathbf{f}(x,y)]$ and $\mathrm{Var}_x[\mathbf{f}(x,y)]$. When a probability density function depends on a (deterministic) parameter $\boldsymbol{\theta}$, the parameter is reported as subscript, like $p_{\boldsymbol{\theta}}$.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Many real-world problems require the solver to make a sequence of decisions without receiving an immediate feedback. This type of problems, named *sequential decision making* problems, arise in several fields, such as automation control, robotics, games, and finance. They require the solving agent to account for the delayed effect of its decisions, along with the possible non-determinism of the environment. Clearly, in order to plan a good decision the agent must refer to some notion of optimality. In real-life, optimality is not necessarily expressed in a formal manner, however, when a sequential decision making problem is tackled from the point of view of Artificial Intelligence (AI) we need to specify some notion of *goal*. A goal-directed agent is able not only to react to the modifications of the environment, but also to display a proactive behavior, intended to achieve its goal. Differently from classic optimal control, no assumption is made on the knowledge of the dynamics of the environment, therefore the controller cannot be designed a priori, but needs to be learned from the interaction with the environment. In those cases the notion of goal is typically encoded by means of a *utility function*, derived from economics, where it represents the customer satisfaction towards a good. In the AI context, the role of the utility function is crucial: it quantifies how much a specific behavior of the agent complies with its implicit goal and drives the learning process. A bad choice for the utility function might have dramatic effects on the final performance. Therefore, one major challenge is how to design a good utility function.

We consider the case in which the dynamics of the environment and the behavior of the agent satisfy the Markov property, i.e., the next state of the environment does not depend on the past states and actions given the current state and action, as well as the decision of the agent is determined as a function of the current environment state only. This requirement is apparently unrealistic, however we can always assume the Markov property provided that the representation of the environment state is sufficiently rich (e.g., we can define a rich state as the concatenation of all the states and actions visited so far, in this way we embed the whole history in a single state). Such problems can be cast into the *Markov Decision Processes* (MDP) framework. The utility function for MDPs is defined in terms of the *immediate reward*. Whenever the agent performs an action it receives from the environment a feedback, the immediate reward, which depends, in the most general case, on the starting state, on the action performed and on the landing state. The utility, from a given state, is defined as the cumulative reward, i.e., the sum (possibly discounted) of the rewards collected along the visited states. Now, rather than designing the utility function directly, we aim to design a suitable reward function.

Whenever the reward model is available we can resort to *Reinforcement Learning* (RL) algorithms to learn an optimal policy, i.e., a prescription of actions that maximizes the cumulative reward. Furthermore, if the model of the dynamics of the environment is known, *Dynamic Programming* (DP) approaches can be used. However, in most of the real cases the environment dynamics is unknown, therefore the agent has to interact with the environment in order to learn it, implicitly or explicitly. Literature has extensively dealt with RL methods, proposing powerful and effective algorithms applied nowadays to a large variety of field, from robotics to finance, from logistics to telecommunication networks to ambient intelligence. However, assuming that a reward function is available is sometimes too demanding. If we consider RL as an algorithmic approach inspired by the learning mechanisms developed by living beings, we realize that, in real-life, the reward function is totally hidden and it is discovered only during the learning process. Nonetheless, biological agents are able to progressively improve their performance by just interacting with the environment and, sometimes, imitating skilled agents (experts).

This provides the primal motivation for *Inverse Reinforcement Learning* (IRL). The problem can be seen as an instance of inverse optimal control: given a set of expert's demonstrations we aim to determine the utility function, or the reward function, optimized by the expert. Clearly, the requirement on the availability of expert's demonstrations might not be fulfilled, preventing the application of IRL to unexplored domains. However, if we aim to construct an agent that behaves like (or better than) a human in a particular domain we can consider the human as an expert, overcoming the possible lack of demonstrations. Indeed, designing a suitable reward function is sometimes hard. The agent designer might have only a very rough idea of the utility function to optimize making impossible to use the classic RL approach. IRL belongs to a class of methods that learn an optimal behavior by exploiting expert's examples, named *Apprenticeship Learning* (AL). The agent, of course, can directly learn a mapping between states and actions, without passing through the reward function. However, this approach, which goes under the name of *Behavioral Cloning* (BC), does not typically display good generalization properties, since it is constrained to the specific set of demonstrations and to the given domain. Whereas, IRL recovers a reward function that is a more parsimonious, transferable and succinct description of the optimal behavior.

A heterogeneous range of approaches to IRL has been proposed in the literature. The feature expectation algorithms [1, 113, 112] aim to recover a representation of the reward function that induces an optimal policy close to the expert behavior. These approaches, however, might result inefficient since they require to solve iteratively the forward RL problem. Recent works focused on removing this limitation. Policy-gradient approaches, like GIRL [94], exploit the policy gradient [111] to recover the optimal parameters of a class of parametric reward functions. Moreover, IRL suffers from the problem of reward ambiguity, i.e., there exists an infinite number of reward functions for the same problem making the expert optimal (including some trivial ones, such as the constant functions). In [125] the maximum entropy principle is applied to single out a unique reward function, but the knowledge of the transition model is required for non-deterministic problems.

The mentioned approaches share the necessity to define a priori a set of hand-crafted features, spanning the reward space (e.g., position of the agent, the distance from the goal). As in supervised learning, the choice of the hypothesis space is substantial for the success of the algorithm. Recent literature focused on incorporating feature construction in IRL algorithms. In [35, 45] the capabilities of deep learning are exploited to implicitly extract reward features in a black-box way and built a reward function using non-linear approximators. The problem of feature extraction has been explicitly addressed only by Feature Construction for Inverse Reinforcement Learning (FIRL) [64], a model-based algorithm that iteratively constructs reward features that better explain regions where the old features were too coarse.

Besides rare exceptions [e.g., 9], the main successful applications of IRL come from the field of autonomous locomotion: simulated highway driving [82, 113], urban navigation [125], aerial imagery based navigation [101], human path planning [76] and quadruped locomotion [100]. In real-world applications we need to ensure the reward transferability, i.e., the possibility to plug the recovered reward function into new environments preserving the ability to recover the optimal policy. In the highway driving, for instance, we expect the agent to behave well even if the traffic pattern has never been experimented by the expert. Several works [64, 65] demonstrate that a non-accurate choice of the reward features might make the agent perform poorly on transfer environments even if it performs well on the training environment.

Motivation

Although IRL made a significant advance in the last decade, several issues remain open. First, most of the effective IRL algorithms [e.g., 1, 113, 112] require knowing the model of the environment or, at least, having access to the environment in order to estimate the optimal policy for each candidate reward function. This produces a significant increase of the sample complexity, preventing these algorithms from scaling to real-world applications. Second, reward ambiguity still represents a notable question. Several approaches have been designed to define an optimatility criterion within the space of the rewards making the expert optimal [e.g., 125, 82]. The maximum entropy principle [125] has been extensively used also with deep architectures thanks to its good learning properties [122]. However, a shared definition of “good” reward is still missing. Finally, several experiments proved that the choice of the function space where to search for the reward function is substantial for the quality of the recovered reward, also in terms of transferability [64, 65]. Engineered features are typically very informative, but hard to design. The deep learning methods overcome this limitation, however, by resorting to a “black-box” approach they do not take advantage of the structure of the underlying MDP in the phase of feature construction. FIRL is the only algorithm that directly exploits the structure of the MDP to extract features; nevertheless, it requires the knowledge of the environment model.

Goal

The goal of this thesis is to design a model-free Inverse Reinforcement Learning algorithm that is able to construct automatically the features and single out a reward function, requiring solely a set of expert’s demonstrations. Our approach is intended to provide a joint answer to the problem of reward feature construction and reward recovery in a fully model-free manner. This represents a novelty for IRL literature, since, as far as we know and apart from deep leaning approaches, no IRL algorithm is able to extract reward features with no knowledge of the environment model.

Contribution

The contribution of this thesis is mainly algorithmic and experimental. Our algorithm, named *Compatible Reward Inverse Reinforcement Learning* (CR-IRL), is able to extract a set of basis functions spanning the space of the optimal rewards by exploiting a first-order condition on the policy gradient. The usage of first-order conditions in inverse optimal control is not new [32, 94], however, we employ them not to directly recover the reward function but to extract a set of optimal features. This step requires the knowledge of a parametric representation of the expert’s policy, that can be estimated from the trajectories with behavioral cloning. Clearly, the parametrization of the expert’s policy influences the recovered approximation space. Therefore, we introduce the notion of *policy rank* that quantifies the ability of the expert’s policy to reduce the space of the optimal reward functions and we provide a bound for the case of finite problems. This represents the main theoretical advance of this work. Once the set of reward features is recovered, we propose a second-order criterion, based on the policy Hessian, to single out the reward function that penalizes as much as possible deviations from the expert’s demonstrated behavior. Eventually, since the recovered reward function is defined only in terms of the visited states and actions, we need to extend it over unexplored regions, either offline or online. The experimental evaluation proposes a comparison with the state-of-the-art IRL methods on classic benchmark problems both in the finite and continuous domains: the Taxi problem [29], the Linear Quadratic Gaussian Regulator [30] and the Car on the Hill environment [33]. We aim to examine both the phases of CR-IRL: feature construction and reward recovery. The empirical results show that the recovered reward function outperforms the original reward function of the problems and those produced with popular IRL methods. In particular, by penalizing deviations from the expert's policy, CR-IRL is able to output a reward function with faster learning curve than that of the original reward function. Moreover, even though CR-IRL exploits behavioral cloning in the early stages, the recovered reward can be used to learn policies that outperform the mere imitation of the expert.

1.1 Overview

The contents of this thesis are organized in the following five chapters. We start in Chapter 2 with an overview of the different aspects of Markov decision processes and reinforcement learning. We introduce the definitions and we discuss several aspects, such as value functions and optimal policies. Moreover, we outline the techniques to solve MDPs, starting from dynamic programming and presenting the most popular reinforcement learning approaches, along with function approximation methods. The focus of the presentation is directed to the aspects we will exploit in the subsequent chapters. In particular, we illustrate in details Reward Shaping, Function Approximation and Policy Search methods. Furthermore, in this chapter, we introduce the notation that we will use throughout the thesis.

In Chapter 3 we depict the landscape of the state-of-the-art inverse reinforcement learning methods. The algorithms are presented in overlapping categories that group methods sharing similarities in the approach. For each category we mainly address a representative algorithm that is discussed in details, underlining pros and cons, whereas some of the most remarkable extensions are only mentioned. A comparative discussion of the properties of the algorithms is provided at the end of the chapter. The goal of this chapter is to guide the reader in a conscious understanding of the fundamental motivations of this work.

Chapter 4 is devoted to the extensive description of CR-IRL. We start discussing the problem of recovering a parametric representation of the expert's policy from the set of available demonstrations. Then we illustrate the procedure to recover an approximation space for the value function by means of a first-order condition on the policy gradient. We introduce, at this point, the notion of policy rank and we derive a bound that holds for finite MDPs. Then we propose two approaches for building the approximation space for the reward function. The first, which requires the knowledge of the transition model, is based on reversing the Bellman Equation and the second, model-free, leverages on reward shaping. Finally, we discuss the reward selection via second-order criteria. We suggest different optimality criteria, all based on the policy Hessian, and we design an efficient heuristic to obtain a near-optimal reward.

In Chapter 5 we evaluate CR-IRL against popular IRL methods. We first introduce the metrics we adopt to compare the performance of the algorithms. Then, for the considered domains, we analyze the performance in terms of learning speed and as a function of the number of expert's trajectories.

In Chapter 6 we summarize the most relevant achievements of this thesis and we highlight the points of strength and weakness of the proposed approach. Furthermore, we suggest possible extensions of this work.

Appendix A briefly outlines the matrix notation for finite MDPs that we will use in the thesis. Appendix B, instead, reports the proofs and derivations omitted in the text.

Chapter 2

Markov Decision Processes and Reinforcement Learning

In the context of Machine Learning, *Reinforcement Learning* (RL) is probably the framework that is closer to the nature of learning for living beings. Human beings learn how to perform a task even without a teacher, by just *interacting* with the environment (e.g., a baby learning how to walk). As common sense suggests, exercising this connection improves the ability to perform the task. Thus, interaction is a precious source of knowledge, that teaches the agent the cause-effect relation characterizing the environment and allows it to select the most appropriate actions in order to achieve its *goal*. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence [109].

In this thesis we refer to the reinforcement learning framework [109, 15, 14, 114], i.e., the computational approach to goal-directed learning from interaction (Figure 2.1). The typical reinforcement learning problem is formulated in terms of an agent acting in an environment deciding which action to perform in order to maximize a payoff signal. We assume the agent receives a reward for performing an action; in the most interesting cases actions affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. Thus, the agent has to learn how to map situations (or states) to actions, in a trial and error fashion. Differently from other machine learning areas, like supervised learning, the agent is not told which action to perform, but it has to discover the most profitable ones. Problems with these characteristics are best described in the framework of *Markov Decision Processes* (MDPs) [96, 114].

Clearly, to identify the best action in each state the agent has to exploit, implicitly or explicitly, an internal representation of the environment. When the dynamics of the environment is known, the standard approach to solve MDPs is to use *Dynamic Programming* (DP) [15, 14], turning the problem in a *model-based control* problem. However, apart from the cases in which the MDP has a limited number of states and actions, dynamic programming is infeasible. Furthermore, the knowledge of the

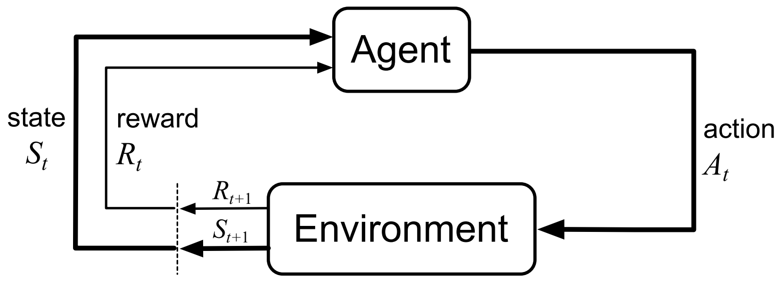

Figure 2.1: The Reinforcement Learning framework (from [109]).

environment dynamics is a demanding assumption rarely fulfilled in real-world applications. RL, on the contrary, deals with *model-free control* enabling applications on large and high-dimensional state-action spaces. The key idea is to exploit samples to represent in a compact and approximate way the underlying environment dynamics.

In this thesis we will consider two classes of RL methods: *Approximate Dynamic Programming* (ADP) [109, 114] and *Policy Search* (PS) [121]. The former can be seen as sample-based version of DP: it aims to approximate a *value function* that represents the expected cumulative reward the agent will get by starting in a given state (and possibly performing a given action). Once the optimal value function is estimated, the optimal policy can be computed. When the space of the states (or actions) is infinite it is not possible to compute the value function for all the states (or actions). RL proposes powerful *function approximation* methods to represent concisely the value functions with no significant performance degradation. Literature proposed a surge of approaches to function approximation that range from the simple linear approximators fed with handcrafted features to the recent deep learning architectures [e.g., 67, 17, 73], able to automatically extract features and estimate the value function.

Policy search, differently from approximate dynamic programming, searches directly for an optimal policy (sometimes, like in actor-critic frameworks, both policy and value function are represented). Indeed, in approximate dynamic programming the value function is used as an intermediate tool to recover the optimal policy; but estimating the value function might be a complex task. Policy search has been addressed as an optimization problem, i.e., searching for a policy in a predetermined space that maximizes the cumulative reward. A variety of optimization methods have been employed to this purpose [e.g., 38, 26, 80].

This chapter is intended to present the main topics of reinforcement learning that we are going to use throughout the thesis. In Section 2.1 we introduce the concept of Markov decision process, the notion of value function and we formalize the optimality criteria. Section 2.2 briefly presents dynamic programming, the standard approach to solve MDPs, when the model of the environment is available. Section 2.3 is devoted to the presentation of the most relevant model-free RL algorithms. In Section 2.4

we discuss the basics of function approximation methods and we focus on the Proto-Value Functions (PVF). Finally, Section 2.5 is devoted to policy search methods which are at the basis of the algorithm proposed in this work.

Not all the material contained in this chapter is essential to understand the following chapters; the reader already skilled in the subject can focus on Reward Shaping (Section 2.1.8), Function Approximation (Section 2.4), in particular Proto-Value Functions (Section 2.4.2) and Policy Search (Section 2.5).

2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a model for sequential decision making problems in which a goal-directed agent interacts with an environment by making actions and sensing perceptions. We consider *discrete time* MDPs in which, at each time step, the agent has to decide which action to perform in order to maximize its utility function. Whenever an action is performed the environment evolves into a new state and the agent receives a payoff, named *immediate reward*. Usually the utility function of an agent is defined in terms of the *cumulative reward*, thus the decision on which action to perform in each state cannot depend only on the immediate reward: an agent may decide to sacrifice the immediate reward to gain more long-term reward.

2.1.1 Definitions

Different definitions of MDP are presented in literature [109, 96, 114]. We consider the following definition inspired to the one presented in [114].

Definition 2.1. *A* Markov Decision Process *is a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{T}, \mu, \gamma)$*, where:*

- $\mathcal{S}$ *is a non-empty measurable set, named* state space*;*
- $\mathcal{A}$ *is a non-empty measurable set, named* action space*;*
- $\mathcal{P}$ *is a function* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ *named* transition model*, that for all* $s \in \mathcal{S}$ *and for all* $a \in \mathcal{A}$ *assigns* $\mathcal{P}(\cdot|s, a)$ *a probability measure over* $\mathcal{S}$*,* $P(\cdot|s, a)$ *is the corresponding probability density function;*
- $\mathcal{R}$ *is a function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta(\mathbb{R})$*, named the* reward model*, that for all* $s, s' \in \mathcal{S}$ *and for all* $a \in \mathcal{A}$ *assigns* $\mathcal{R}(\cdot|s, a, s')$ *a probability measure over* $\mathbb{R}$*,* $R(\cdot|s, a, s')$ *is the corresponding probability density function and* $R(s, a) = \mathop{\mathbb{E}}_{\substack{s' \sim P(\cdot|s,a) \\ r \sim R(\cdot|s,a,s')}} [r]$ *is the state-action expected reward;*
- $\mathcal{T}$ *is a non-empty totally ordered set, named set of the* decision epochs*;*

- μ *is a probability measure over* $\mathcal{S}$*, named* distribution of the initial state*,* $\mu(\cdot)$ *is the corresponding probability density function;*

- $\gamma \in [0, 1]$ *is the* discount factor*.*

The state space $\mathcal{S}$ and the action space $\mathcal{A}$ define the sensor and actuator possibilities of the agent; they can be either finite or infinite, discrete or continuous. Sometimes not all the actions are performable in all states, in this case it is convenient to define the set $\mathcal{A}(s)$ for all $s \in \mathcal{S}$ which is the set of actions available in state s, therefore $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}(s)$.

The dynamics of the environment is represented by the transition model that, given the current state s and the current action a, assigns for every Borel set $S' \in \mathcal{B}(\mathcal{S})$ the quantity $\mathcal{P}(S'|s, a)$ that is the probability to end up in a state contained in S' by performing action a in state s. $P(\cdot|s, a)$ is the density function induced by $\mathcal{P}$; if the MDP is discrete $P(s'|s, a)$ is actually the probability to end up in state $s' \in \mathcal{S}$. The term *Markovian* is used in this context since the environment satisfies the Markov property: the future state does not depend on the past states and actions given the current state and action.[1]

The immediate payoff is modeled by means of a scalar reward,[2] that given the current state s, the current action a and the next state s' assigns for all Borel sets $A \in \mathcal{B}(\mathbb{R})$ the quantity $\mathcal{R}(A|s, a, s')$ that is the probability to get a payoff in A by starting from state s, performing action a and ending up in state s'. $R(\cdot|s, a, s')$ is the corresponding density function. In most common applications the state-action expected reward $R(s, a)$ is used, it is defined as the expected value of the reward taken over all next states s' and all real rewards r. Sometimes it is convenient to consider the state-action-state expected reward $R(s, a, s') = \mathbb{E}_{r \sim R(\cdot|s,a,s')}[r]$. We will also assume that the immediate reward is upper bounded in absolute value, i.e., $\|R\|_\infty = \max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} |R(s, a)| \leq M < +\infty$.

The time is modeled as a discrete set of decision epochs, typically represented as a sequence of natural numbers $\mathcal{T} = \{0, 1, ..., T\}$ where T is called *horizon* and can be either finite $T \in \mathbb{N}$ or infinite $T = \infty$. In the former case the MDP is said to be finite horizon, otherwise it is called infinite horizon. An MDP is said *episodic* if the state space contains a *terminal* (or *absorbing*) state, i.e., a state s from which no other states can be reached (for all actions $a \in \mathcal{A}$, $P(s|s, a) = 1$). Typically, it is assumed that all actions performed in a terminal state yield zero reward.

A state-action-reward sequence $\tau = \{(s_t, a_t, r_t)\}_{t=0}^{T(\tau)}$ is called *trajectory*, it can be either finite or infinite. The set of all trajectories is $\mathbb{T} = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^\infty$ which is the

[1]Formally, a stochastic process X_t is Markovian if $\mathbb{P}(X_{t+1} = z|X_t = y, X_{t-1} = y_{t-1}, ..., X_0 = y_0) = \mathbb{P}(X_{t+1} = z|X_t = y)$. In other words X_t is a *sufficient statistic* for the future, meaning that it captures all the information from the history.

[2]We will always consider rewards as scalar quantities. Sutton [109] suggested that any goal can be well thought as the maximization of a cumulative scalar reward. This assumption may be not always satisfied but it is so flexible and simple that we won't disprove it.

disjoint union of the set of finite-length trajectories $\mathbb{T}_* = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^*$ and infinite-length trajectories $\mathbb{T}_\omega = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^\omega$. Sometimes we will refer to the trajectory as a sequence of state-acton pairs only (rewardless trajectories). A finite interleaved sequence of states and actions terminating with a state and containing $t \in \mathcal{T}$ actions is called *history* of length t. The set of histories of length t is $\mathbb{H}_t = (\mathcal{S} \times \mathcal{A})^t \times \mathcal{S}$, while the set of all histories is $\mathbb{H}_* = \bigcup_{t=0}^{\infty} H_t$.

2.1.2 Decision rules and policies

A decision maker (or agent) acting in an MDP at each decision epoch t observes the state of the environment s_t and chooses which action a_t to perform. The agent can select its actions at any stage based on the observed history. A rule describing the way actions are selected is called a *decision rule* or behavior. A decision rule d_t tells the agent how to choose the action at a given decision epoch $t \in \mathcal{T}$. In the most general case, an agent bases its decision on the history (*history dependent* decision rule). If the decision rule prescribes a single action it is said *deterministic*, $d_t : \mathbb{H}_t \to \mathcal{A}$; if it provides a probability distribution over the set of actions is said *stochastic*, $d_t : \mathbb{H}_t \to \Delta(\mathcal{A})$. Clearely, deterministic decision rules are a particular case of the stochastic ones. An important role in the theory of MDPs is played by the decision rules for which the action to perform (or the probability distribution over the actions) depends only on the current state. This class defines the *Markovian* decision rules, $d_t : \mathcal{S} \to \mathcal{A}$ ($d_t : \mathcal{S} \to \Delta(\mathcal{A})$ in case of stochastic decision rules). We can now introduce the definition of *policy*.

Definition 2.2. *A* policy *is a sequence of decision rules, one for each decision epoch*

$$\pi = \{d_t : \mathcal{S} \to \Delta(\mathcal{A})\}_{t \in \mathbb{T}}.$$

A policy is *stationary* when the decision rules do not depend on the decision epoch, i.e., $d_t = d$ for all $t \in \mathcal{T}$, in this case we will indicate d with π. If the policy is Markovian $\pi(a|s)$ represents the probability density to execute action a in state s, if it is also deterministic we will indicate with $\pi(s)$ the action prescribed in state s.

Definition 2.3. *A* Markovian stationary policy *is a function $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that for all states $s \in \mathcal{S}$ assigns a probability measure $\pi(\cdot|s)$ over $\mathcal{A}$. If the policy is deterministic then $\pi : \mathcal{S} \to \mathcal{A}$.*

Markovian stationary policies play an important role in the theory of infinite horizon MDPs. Whenever not differently specified we will use the term policy to refer to Markovian stationary policy.

An MDP $\mathcal{M}$ equipped with a policy π induces a probability distribution over the set of trajectories $\mathbb{T}$. Given a trajectory $\tau = \{(s_t, a_t, r_t)\}_{t=0}^{T(\tau)}$, its probability density function is:

$$p(\tau|\mu, \pi, \mathcal{P}) = \mu(s_{\tau,0}) \prod_{t=0}^{T(\tau)-1} \pi(a_{\tau,t}|s_{\tau,t}) P(s_{\tau,t+1}|s_{\tau,t}, a_{\tau,t}), \qquad \forall \tau \in \mathbb{T}. \tag{2.1}$$

Whenever the context will make it clear, we will indicate this probability density function with $p(\tau)$ only.

2.1.3 Markov Reward Processes

Any MDP paired with a Markovian policy π induces a *Markov Reward Process* (MRP), defined by the tuple $(\mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma, \mu)$. For all $s \in \mathcal{S}$, $\mathcal{P}^\pi(\cdot|s)$ is a probability measure over $\mathcal{S}$. The corresponding density function P^π is obtained by marginalizing transition model P of the MDP over the actions:

$$P^\pi(s'|s) = \int_{\mathcal{A}} \pi(a|s) P(s'|s,a) \mathrm{d}a, \qquad \forall s, s' \in \mathcal{S}. \tag{2.2}$$

$P^\pi(s'|s)$ represents the probability density to reach state s' from state s in one step. Symmetrically, $\mathcal{R}^\pi(\cdot|s,s')$ for all $s, s' \in \mathcal{S}$ is a probability measure over $\mathbb{R}$; the corresponding density function R^π obtained from the reward model of the MDP as:

$$R^\pi(r|s,s') = \int_{\mathcal{A}} \pi(a|s) R(r|s,a,s') \mathrm{d}a, \qquad \forall r \in \mathbb{R}, \quad \forall s, s' \in \mathcal{S}. \tag{2.3}$$

In an MRP the notion of action disappears. Thus, MRPs are suitable models for uncontrolled processes, i.e., stochastic phenomena taking place in the environment and producing rewards at each time step, while MDPs are good models for controllable processes, in which the intervention of an agent, by means of an action, is possible.

2.1.4 Markov Chains

The triple $(\mathcal{S}, \mathcal{P}^\pi, \mu)$, i.e., an MRP devoid of the reward, is a *Markov Chain* (MC) (or *Markov Process*). In a MC we can define the n-steps transition model $\mathcal{P}^\pi_n$ obtained from the 1-step transition model $\mathcal{P}^\pi$ recursively. For the density functions the following equality holds:

$$P^\pi_n(s'|s) = \int_{\mathcal{S}} P^\pi_{n-1}(s'|s'') P^\pi(s''|s) \mathrm{d}s'', \qquad \forall s, s' \in \mathcal{S}, \quad \forall n = 2, 3, ..., \tag{2.4}$$

where $P^\pi_1(s'|s) = P^\pi(s'|s)$. $P^\pi_n(s'|s)$ is also called n-steps transition kernel: it represents the probability density to reach state s' staring from state s in n steps. Given the distribution of the initial state μ is possible to compute the probability that the agents finds itself in a state s after n steps:

$$\mu^\pi_n(s) = \int_{\mathcal{S}} P^\pi_n(s|s') \mu(s') \mathrm{d}s', \qquad \forall s \in \mathcal{S}. \tag{2.5}$$

Eventually, we introduce the notion of *stationary distribution.* A stationary distribution of a Markov chain induced by a policy π is defined (if exists) as:

$$\mu^\pi(s) = \int_{\mathcal{S}} P^\pi(s'|s) \mu^\pi(s') \mathrm{d}s', \qquad \forall s \in \mathcal{S}. \tag{2.6}$$

For further details about Markov chains refer to [114, 96].

2.1.5 State occupancy and distribution

An infinite horizon MDP along with a policy defines distribution on the state space as defined by [111]. When the MDP is undiscounted, i.e., $\gamma = 1$, the *undiscounted future state distribution* is given by:

$$d_\mu^\pi(s) = \lim_{t \to +\infty} \frac{1}{t} \sum_{i=0}^{t} \mathbb{P}(s_i = s | \mu, \pi, \mathcal{P}), \qquad \forall s \in \mathcal{S}, \tag{2.7}$$

where $\mathbb{P}(s_i = s|\mu, \pi, \mathcal{P})$ is the probability that the i-th state of a trajectory is state s, given the distribution of the initial state μ, policy π and the transition model $\mathcal{P}$. Clearly, $\mathbb{P}(s_i = s|\mu, \pi, \mathcal{P}) = \mu_i^\pi(s)$, we use different symbols to remain coherent with the notation introduced in [111]. $d_\mu^\pi(s)$ represents the long-term probability that the state of the environment is s. In the case of a discounted MDP, i.e. $\gamma \in [0, 1)$, the γ*-discounted future state occupancy* is defined as:

$$d_{\mu,\gamma}^\pi(s) = \sum_{t=0}^{+\infty} \gamma^t \mathbb{P}(s_i = s | \mu, \pi, \mathcal{P}), \qquad \forall s \in \mathcal{S}. \tag{2.8}$$

The γ*-discounted future state distribution* is obtained by multiplying for $(1 - \gamma)$ the occupancy. The definitions can be extended in the state-action space, by embedding the policy. The state-action future occupancy, discounted or not, is given by:

$$\delta_{\mu,\gamma}^\pi(s, a) = d_{\mu,\gamma}^\pi(s)\pi(a|s), \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{2.9}$$

2.1.6 Value functions

The goal of an agent in an MDP is to play a policy that maximizes some optimality criterion, typically encoded using a *utility function.* A utility function assigns a real number to a trajectory, representing how much that trajectory is "good" for the agent, i.e., the degree of compliance of the trajectory to the agent's goal.[3] In most of the cases the utility function of the agent is the cumulative reward, i.e., the sum (possibly discounted) of the rewards collected along the trajectory. Different definitions have been proposed on the basis of the properties of the MDP [15, 96].

Cumulative reward functions

The simplest idea is to define the utility function as the sum of the rewards collected along the trajectories, named *expected total reward* of a policy π:

$$J^\pi = \mathbb{E}_\tau \left[\sum_{t=0}^{T(\tau)} R(s_{\tau,t}, a_{\tau,t}) \right],$$

[3]The notion of utility function comes from economics and it is the most used method to encode in a mathematically simple and formal way the goal of an agent.

where $\mathbb{E}_\tau$ is an abbreviation for $\underset{\tau \sim p(\cdot|\mu,\pi,\mathcal{P})}{\mathbb{E}}$. The main limitation of the previous equation is that the series might not converge for infinite horizon MDPs ($T(\tau) = +\infty$). To avoid the potential divergence we can resort to *mean expected reward*, defined as:

$$J^\pi = \lim_{t \to +\infty} \mathbb{E}_\tau \left[\frac{1}{t} \sum_{i=0}^{t} R(s_{\tau,i}, a_{\tau,i}) \right].$$

If the reward is upper bounded in absolute value, i.e. $\max_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} |R(s,a)| \leq M < +\infty$, then the mean expected reward is guaranteed to converge for infinite-horizon MDPs [96].

The cumulative reward function used more diffusely in literature is the *discounted expected cumulative reward*, also known as *expected return*:

$$J^\pi = \mathbb{E}_\tau \left[\sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t}) \right] = \mathbb{E}_\tau \big[R(\tau) \big], \tag{2.10}$$

where $R(\tau)$ is the *trajectory return* defined as $R(\tau) = \sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t})$.

The usage of a discount factor $\gamma \in [0, 1)$ has the evident mathematical advantage to prevent the cumulative reward from diverging, under the condition that the immediate reward is upper bounded in absolute value. The discount factor is open to multiple interpretations. From an economical/financial point of view the discount factor accounts for the fact that an agent might be more interested in a payoff obtained in the near future rather than a payoff obtained far in the future. Values of γ close to 0 lead to a "miopic" evolution (at the limit in which $\gamma = 0$ the agent is interested only in the immediate reward and the solution of the problem is obtained with a greedy policy); while values of γ close to 1 lead to a "far-sighted" evolution. From a statistical point of view the discount factor is related to the probability that the process continues for another epoch. If the MDP is *episodic*, i.e., all trajectories reach an absorbing state, then $\gamma = 1$ can be used. Formally, the discount factor is a parameter of the MDP; however, many times it can be tuned to favor the convergence of the RL algorithms. Small values of γ improve the convergence rate; in the extreme case of $\gamma = 0$ the problem degenerates in a greedy choice. However, small γ could compromise the quality of the solution since the rewards collected in the far future become less important. Thus, the choice of γ, when possible, has to trade off the quality of the recovered solution and the convergence speed of the algorithms.

Value functions

The most trivial and inefficient way to determine the optimal behavior in an MDP is to list all possible behaviors and identify the one having the highest utility function, according to the cumulative reward function chosen. A better approach is based on computing value functions, which provide the utility function starting from each state e following a given policy. In this approach, one first computes the optimal value function and then determines an optimal behavior with relative easiness.

Given a policy π for an MDP $\mathcal{M}$, *policy evaluation* consists in computing the utility starting from each state and following π. Policy evaluation is based on the notion of *state value function* or simply *value function*.

Definition 2.4. *Given an MDP $\mathcal{M}$ and a policy π, the* state value function *in a state $s \in \mathcal{S}$ is the expected return starting from state s and following policy π:*

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim p(\cdot|s_0=s,\pi,\mathcal{P})} \left[\sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t}) \right], \qquad \forall s \in \mathcal{S}. \tag{2.11}$$

The state value function does not embed enough information to let the agent determine the greedy action in each state, i.e., the action yielding the maximum utility (this can be done if the agent knows the transition model $\mathcal{P}$). Thus, V^{π} is not suitable for model-free control. For control purposes the *action value function* or *Q-function* is typically used.

Definition 2.5. *Given an MDP $\mathcal{M}$ and a policy π, the* action value function *in a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ is the expected return starting from state s, taking action a and following policy π:*

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau \sim p(\cdot|s_0=s,a_0=a,\pi\,\mathcal{P})} \left[\sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t}) \right], \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{2.12}$$

Clearly, the value function is obtained by averaging the action value function over the action space:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} \left[Q^{\pi}(s,a) \right], \qquad \forall s \in \mathcal{S}. \tag{2.13}$$

Sometimes it is convenient to introduce the *advantage function* [8] defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s), \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}.$$

The advantage function represents how much an action a is convenient in a state s w.r.t. the average utility of the actions.

The performance of a policy on a MDP is typically measured taking into account the distribution of the initial state μ. It is possible to define the expected return in terms of the value function:

$$J^{\pi} = \mathop{\mathbb{E}}_{s \sim \mu} [V^{\pi}(s)].$$

Bellman Operators and Bellman Equations

We introduced the value functions by means of trajectory-based definitions; however, they can be computed in a recursive manner by leveraging on the *Bellman Expectation Operator*, defined as $\mathcal{T}_V^{\pi} : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$:

$$\mathcal{T}_V^{\pi}[V](s) = \mathop{\mathbb{E}}_{\substack{a \sim \pi(\cdot|s) \\ s' \sim P(\cdot|s,a)}} \left[R(s,a) + \gamma V(s') \right], \qquad \forall s \in \mathcal{S}. \tag{2.14}$$

It is possible to prove [96] that the state value function V^π is the unique fixed point of the operator $\mathcal{T}_V^\pi$, i.e., it satisfies $\mathcal{T}_V^\pi[V^\pi] = V^\pi$ (*Bellman Expectation Equation*). Analogously, for the Q-function the Bellman Expectation Operator, $\mathcal{T}_Q^\pi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, is defined as:

$$\mathcal{T}_Q^\pi[Q](s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}} \left[Q(s',a')\right], \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{2.15}$$

Like for the value function, Q^π is the unique fixed point of $\mathcal{T}_Q^\pi$, i.e., $\mathcal{T}_Q^\pi[Q^\pi] = Q^\pi$. It is worth to notice that both operators are linear. Furthermore, they satisfy the contraction property in L_∞-norm, i.e., $\|T_\star^\pi f_1 - T_\star^\pi f_2\|_\infty \leq \gamma\|f_1 - f_2\|_\infty$, thus the repeated application of $T_\star^\pi$ makes any function converge to the value function:

$$\lim_{t\to+\infty} \left(T_V^\pi\right)^t[f] = V^\pi, \qquad \forall f \in \mathbb{R}^{\mathcal{S}},$$
$$\lim_{t\to+\infty} \left(T_Q^\pi\right)^t[f] = Q^\pi, \qquad \forall f \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}.$$

This property is exploited by iterative policy evaluation (see 2.2.1). When the MDP is finite, the Bellman Equations become matrix equations and can be solved to find the value functions (the matrix form for finite MDPs in presented in Appendix. A). Under the assumption $\gamma \in [0,1)$, we have:

$$\begin{aligned} \mathbf{v}^\pi = \mathbf{r}^\pi + \gamma\mathbf{P}^\pi\mathbf{v}^\pi \quad &\Longrightarrow \quad \mathbf{v}^\pi = (\mathbf{I} - \gamma\mathbf{P}^\pi)^{-1}\mathbf{r}^\pi, \\ \mathbf{q}^\pi = \mathbf{r} + \gamma\mathbf{P}\boldsymbol{\pi}\mathbf{q}^\pi \quad &\Longrightarrow \quad \mathbf{q}^\pi = (\mathbf{I} - \gamma\mathbf{P}\boldsymbol{\pi})^{-1}\mathbf{r}. \end{aligned} \tag{2.16}$$

For the case of the state value function, this method however requires the inversion of matrix $(\mathbf{I} - \gamma\mathbf{P}^\pi)$, yielding a computational complexity of $O(|\mathcal{S}|^3)$[4], thus iterative policy evaluation, being more efficient, is typically preferred.

2.1.7 Optimality criteria

The standard approach to determine the *optimal policy*, is based on the notion of *optimal value function*, i.e., the value function that maximizes the utility for every state. From now on we will denote with Π the class of Markovian stationary policies.

Optimal value functions

We start by providing the definition of *optimal state value function.*

Definition 2.6. *Given an MDP $\mathcal{M}$, the* optimal state value function *in a state $s \in \mathcal{S}$ is given by:*

$$V^*(s) = \max_{\pi\in\Pi} V^\pi(s), \qquad \forall s \in \mathcal{S}.$$

[4]The complexity of inversion of a $n \times n$ matrix is $O(n^3)$ for non optimized algorithms. The exponent can be reduced up to 2.373 with Coppersmith-Winograd-like algorithm [25].

The optimal value function specifies the best possible performance for the given MDP. It is possible to prove that the optimal value function is the unique fixed point of the *Bellman Optimality Operator*, $\mathcal{T}_V^* : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$, defined as:

$$\mathcal{T}_V^*[V](s) = \max_{a \in \mathcal{A}} \Big\{ R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \big[V(s') \big] \Big\}, \qquad \forall s \in \mathcal{S}. \tag{2.17}$$

Differently from the Bellman Expectation Operator, due to the presence of the maximum function, the Bellman Optimality Operator is non-linear. Analogously, we can define the *optimal action value function.*

Definition 2.7. *Given an MDP $\mathcal{M}$, the* optimal action value function *in a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ is given by:*

$$Q^*(s,a) = \max_{\pi \in \Pi} Q^\pi(s,a), \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}.$$

The corresponding Bellman Optimality Operator, $\mathcal{T}_Q^* : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$, is:

$$\mathcal{T}_Q^*[Q](s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \big[\max_{a' \in \mathcal{A}} Q(s',a') \big], \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}. \tag{2.18}$$

It can be proved that V^* and Q^* are the fixed points of the corresponding Bellman Optimality Operators. Furthermore, the Bellman Optimality Operators satisfy the contraction property in L_∞-norm. Again, applying iteratively these operators any function converges to the optimal value function. However, being non-linear, no closed form solution can be found in general. Finally, the *expected optimal return* can be computed as:

$$J^* = \mathop{\mathbb{E}}_{s \sim \mu} \big[V^*(s) \big]. \tag{2.19}$$

Optimal policies

Given the optimal value function, the main question is whether there exists a policy π^* (that we will call *optimal policy*) such that the corresponding value function V^{π^*} coincides with the optimal value function V^*, i.e., a policy whose performance is the best possible performance for the given MDP. Let us start providing the definition of *optimal policy.*

Definition 2.8. *Given an MDP $\mathcal{M}$, a Markovian stationary policy $\pi \in \Pi$ is optimal if it satisfies:*

$$\mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} \big[Q^*(s,a) \big] = V^*(s), \qquad \forall s \in \mathcal{S}.$$

The intuitive meaning of the previous definition is that the optimal policy prescribes in each state the action(s) yielding the best performance possible. We can use the value functions to define a partial ordering over the space of Markovian stationary policies.

Definition 2.9. *Given two policies $\pi, \pi' \in \Pi$ for the same MDP $\mathcal{M}$ policy π is better than or equal to ($\succeq$) policy π' when the value function of π is greater than or equal to the value function of π' in all states:*

$$\pi \succeq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s), \quad \forall s \in \mathcal{S}.$$

Clearly, if an optimal policy exists it must be a maximal of the ordering relation $\succeq$. Indeed, the existence is proved by the following result [96].

Theorem 2.1. *For any Markov Decision Process $\mathcal{M}$ the following statements hold:*

- *there exists an optimal policy π^* that is better than or equal to all other policies $\pi \in \Pi$, i.e., $\pi^* \succeq \pi$, $\forall \pi \in \Pi$;*
- *all optimal policies π^* achieve the optimal value function, i.e., $V^{\pi^*}(s) = V^*(s)$, $\forall s \in \mathcal{S}$;*
- *there always exists a deterministic optimal policy.*

The theorem ensures the existence of an optimal policy, but there is no guarantee on uniqueness. Furthermore, among all optimal policies for a given MDP, there always exists a deterministic one.[5] Once the optimal action value function is computed it is possible to find a deterministic optimal policy by considering the *greedy policy*:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a). \tag{2.20}$$

The equality symbol in the previous equation might be inappropriate when there exist multiple optimal actions in the same state. This is the case when multiple optimal policies arise, since all policies that select a greedy action or any convex combination of them are optimal. It is worth noting that the knowledge of the optimal action value function Q^* is sufficient to compute an optimal policy in a model-free manner, while the optimal value function V^* requires the knowledge of the transition model $\mathcal{P}$. Model-free RL algorithms, that we will present in Section 2.3, aim to estimate Q^* from trajectories.

Literature proposed other optimality criteria. Instead of requiring that the optimal policy maximizes the value function in each state we can resort to a less demanding criterion, requiring that an optimal policy optimizes the expected return.

Definition 2.10. *Given an MDP $\mathcal{M}$ a Markovian stationary policy $\pi^* \in \Pi$ is optimal if it maximizes the expected return:*

$$\pi^* = \arg\max_{\pi \in \Pi} J^{\pi}.$$

[5]A simple rationale to get convinced that a deterministic optimal policy always exists is the following. Suppose that there exists an MDP for which only stochastic policies are optimal. For all states s in which there are (at least) two actions, a_1 and a_2, that are played with positive probability, it must be that $Q^*(s, a_1) = Q^*(s, a_2)$ otherwise the policy would not be optimal. Thus, also the deterministic policy that prescribes to play action a_1 only (or a_2 only) would be optimal.

Notice that an optimal policy according to definition (2.8) is always optimal in the sense of definition (2.10), while the vice versa does not hold. For instance, an optimal policy in the sense of definition (2.10) might select a suboptimal action in a region of the MDP that is not reachable according to the distribution of the initial state.

2.1.8 Reward Shaping

In sequential decision making problems, like MDPs, the intuitive notion of "task" is encoded by means of the reward function. The reward function, along with the transition model, determines the optimal policy. However, there exist multiple (infinite) reward functions inducing the same optimal policy, but not all of them share the same learning properties (e.g., learning speed). This observation induces the practice of *reward shaping* in RL, i.e., supplying additional rewards to a learning agent to guide the learning process, and possibly make it faster. In [81] this procedure is formalized by means of a *reward shaping function* $F(s,a,s')$ that is added to the original reward $R(s,a,s')$. The authors investigate the effect of reward shaping on the optimal policy and provide the following result.

Theorem 2.2. *Given an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \mathcal{T}, \mu, \gamma)$ and $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ a reward shaping function. We say F is* potential-based *shaping function if there exists a function $\chi : \mathcal{S} \rightarrow \mathbb{R}$ such that:*

$$F(s,a,s') = \gamma\chi(s') - \chi(s), \qquad \forall s, s' \in \mathcal{S},\ \forall a \in \mathcal{A}. \tag{2.21}$$

Then, π^* is an optimal policy for $\mathcal{M}$ if and only if π^* is an optimal policy for $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R+F, \mathcal{T}, \mu, \gamma)$.

The result is expressed for reward functions depending also on the next state, we can adapt the result for reward functions depending only on the current state and current action, by just averaging over the next state:

$$F(s,a) = \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[F(s,a,s') \right] = \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[\chi(s') \right] - \chi(s), \qquad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}.$$

The theorem provides a necessary and sufficient condition so that a reward shaping function does not change the optimal policy. Clearly, the transformation does not preserve the value functions. As a corollary, the optimal value functions are translated by the quantity $\chi(s)$:

$$\begin{aligned} V^*_{\mathcal{M}'}(s) &= V^*_{\mathcal{M}}(s) - \chi(s), \qquad \forall s \in \mathcal{S}, \\ Q^*_{\mathcal{M}'}(s,a) &= Q^*_{\mathcal{M}}(s,a) - \chi(s), \qquad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}. \end{aligned}$$

An interesting consequence is that the potential-based functions determine the class of the reward functions that are optimal under any policy. Indeed, they are indefferent to policies in the sense that they give us no reason to prefer a policy over another.

At an intuitive level this accounts for why they do not give us any reason to prefer any other policy than π^* when we switch from $\mathcal{M}$ to $\mathcal{M}'$ [81]. The main question is how to select the function $\chi(s)$. Clearly, $\chi(s)$ should be chosen using expert knowledge about the domain. A straightforward choice is $\chi(s) = V^*_{\mathcal{M}}(s)$, in this way the new value function $V^*_{\mathcal{M}'}(s)$ would be zero for all states and the new Q-function $Q^*_{\mathcal{M}'}(s, a)$ would become the advantage function $A^*_{\mathcal{M}}(s, a)$. Intuitively, this is a particularly easy function to learn even without the model of the environment, since we just need to lean the non-zero Q-values [81]. This is formalized in [80, Theorem 3] proving that using a shaping function sufficiently close to the value function we can run the same MDP with a smaller discount factor and still obtain a near-optimal policy. An analysis of the different aspects of reward shaping is provided in [27, 44, 120].

2.2 Dynamic Programming

Solving an MDP means finding an optimal policy.[6] The most naïve approach consists in enumerating all possible policies (*brute force*), evaluating their performance and returning the best one. This method, however, is inefficient for the finite MDPs since the number of policies is exponential $|\mathcal{A}|^{|\mathcal{S}|}$ and unapplicable to infinite MDPs. This section briefly outlines *Dynamic Programming* (DP), the most common approach to solve MDPs when the model of the environment is known. Dynamic programming [12] is a very general method to solve problems that can be decomposed in (overlapping) subproblems and satisfying the principle of optimality, i.e., once the subproblems are solved also the original problem is solved. MDPs comply to this requirement thanks to the recursive nature of Bellman Equation. The standard techniques are: *policy iteration* and *value iteration.*

2.2.1 Policy Iteration

Policy Iteration [48] alternates *policy evaluation* and *policy improvement* phases (Figure 2.2). The algorithm (Alg. 2.1) starts from an arbitrary policy (e.g., random policy), policy evaluation aims to recover the value function of the current policy. This phase can be performed in different ways: closed form solution (2.16) or iterative policy evaluation (iterative application of Bellman Expectation Operator). The first approach is computationally demanding, but allows recovering the exact value function, while the latter recovers just an approximation. In practice, when using iterative policy evaluation, we do not need to wait for convergence: a sufficiently accurate approximation is enough (*modified policy iteration* [97]). Policy improvement builds a new policy by taking in each state the action that maximizes the value function (greedy-action): $\pi^{(t+1)}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi^{(t)}}(s, a)$, $\forall s \in \mathcal{S}$. All generated policies are deterministic and the new policy is guaranteed to yield better

[6]Notice that we are interested in just one optimal policy not in the complete space of optimal policies.

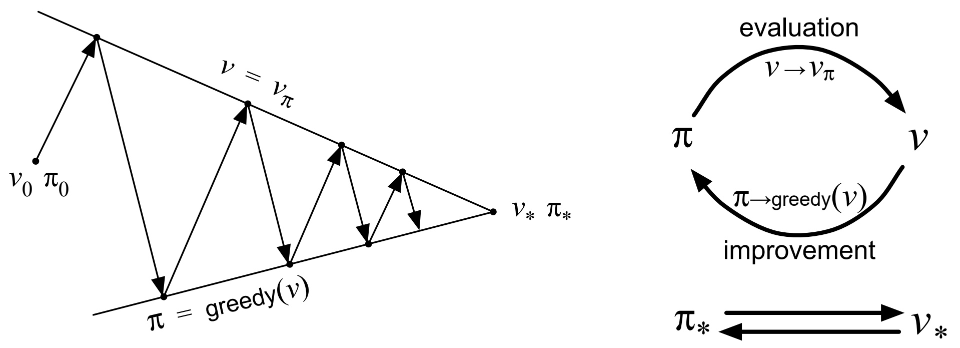

Figure 2.2: Policy iteration (from [109]).

Algorithm 2.1 Policy iteration.

```
Input: $\mathcal{S}$, $\mathcal{A}$, $P$, $R$, $T_{max}$
Output: $\pi^{(T_{max})}$
1: Initialize policy $\pi^{(0)}$ arbitrarily
2: for $t = 0, 1, ..., T_{max} - 1$ do
3:     Evaluate policy $\pi^{(t)}$ and get $V^{\pi^{(t)}}(s),\ \forall s \in \mathcal{S}$
4:     Perform policy improvement
           $\pi^{(t+1)}(s) = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[ V^{\pi^{(t)}}(s') \right] \right\}, \quad \forall s \in \mathcal{S}$
5: end for
6: return optimal policy $\pi^{(T_{max})}$
```

performance w.r.t. the old one. This is ensured by the following result.

Theorem 2.3. (Policy Improvement Theorem) *Given an MDP $\mathcal{M}$ and two deterministic policies π and π':*

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s),\ \forall s \in \mathcal{S} \quad \Longrightarrow \quad V^{\pi'}(s) \geq V^\pi(s),\ \forall s \in \mathcal{S}.$$

If we fix the maximum number of iterations to T_{max}, the following contraction property holds [96]:

$$\left\| V^{\pi^{(T_{max})}} - V^* \right\|_\infty \leq \gamma \left\| V^{\pi^{(T_{max}-1)}} - V^* \right\|_\infty \leq \gamma^{T_{max}} \left\| V^{\pi^{(0)}} - V^* \right\|_\infty.$$

The sequence of policies is guaranteed to converge to the optimal policy (in finite number of steps for finite MDPs) [96]. Similar algorithms can be derived for estimating the Q-function instead of the value function (Q-iteration).

2.2.2 Value Iteration

Value Iteration [12] computes the optimal value function without representing intermediate policies (Alg. 2.2). This approach is based on the iterative application

Algorithm 2.2 Value iteration.

```
Input: $\mathcal{S}$, $\mathcal{A}$, $P$, $R$, $T_{max}$
Output: $\pi^{(T_{max})}$
1: Initialize the policy $V^{(0)}$ arbitrarily
2: for $t = 0, 1, ..., T_{max} - 1$ do
3:     Apply Bellman Oprimality Operator
           $V^{(t+1)}(s) = T^*_V[V^{(t)}](s), \;\; \forall s \in \mathcal{S}$
4: end for
5: Compute the optimal policy
       $\pi^{(T_{max})} = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[ V^{(T_{max})}(s') \right] \right\}, \;\; \forall s \in \mathcal{S}$
6: return optimal policy $\pi^{(T_{max})}$
```

of the Bellman Optimality Operator. In the end, the optimal policy is obtained as the greedy policy induced by the recovered optimal value function. The convergence to the optimal value function is guaranteed. In particular, given two consecutive approximations of the optimal value function we can bound the error w.r.t. the true optimal value function:

$$\left\| V^{(t+1)} - V^{(t)} \right\|_\infty < \epsilon \quad \implies \quad \left\| V^{(t+1)} - V^{*} \right\|_\infty < \frac{2\epsilon\gamma}{1-\gamma}.$$

While policy iteration represents explicitly the policy, value iteration focuses on the value function only. This means that intermediate value functions may not correspond to any policy. Both are polynomial time algorithms for MDPs with fixed discount factor [71]. Considering the single iteration, policy iteration is more computationally demanding w.r.t. value iteration since it requires evaluating the policy and performing the greedy improvement, but it tends to converge in a smaller number of iterations. Besides DP, also Linear Programming (LP) can be employed to recover the optimal value function. However, LP becomes impractical at a much smaller number of states than DP methods do.

2.3 Reinforcement Learning Algorithms

Dynamic programming is a model-based approach since it assumes the knowledge of the transition model, which in most of real-world scenarios is unknown. Furthermore, DP becomes computationally infeasible for large state-action spaces and it is clearly inapplicable for infinite MDPs. The lack of the transition model, imposes to sample the environment in order to approximate, explicitly or implicitly, the underlying dynamics. Reinforcement learning focuses on this framework and provides a solution to the *model-free control problem*, i.e., finding the optimal policy without the knowledge of the MDP model. RL methods, essentially, reshape the DP algorithms in a sample-based version. Without claiming to be complete, in this section we provide an overview of the main classic model-free RL algorithms for finite state-action spaces.

2.3.1 RL Dichotomies

We can classify the RL algorithms according to several criteria. I) *Model-free* algorithms aim to learn the optimal policy directly from samples, while *model-based* algorithms first use samples to estimate the transition model and then apply standard DP to find the optimal policy. II) *On-policy* RL learns the value function of the policy used to collect samples, whereas *off-policy* RL learns another policy, typically the optimal policy, while the policy used to collect samples is suboptimal. III) *Online* methods learn while collecting data, instead *offline* methods perform learning when all the data have been collected. IV) *Tabular* algorithms represent the value function as a table (they are used for finite MDPs), while *function approximation* algorithms exploit approximators to estimate the value function (used in continuous MDPs). V) Finally, we distinguish between: *value-based* algorithms (they learn the optimal value function), *policy-based* (they learn the optimal policy) and *actor-critic* (they are composed of two components: the actor that demonstrates a policy and the critic that updates the value function).

2.3.2 The Prediction Problem

With the term *prediction*, we refer to the problem of estimating the value function of a given policy. We present here the two model-free approaches that we will discuss also for the control problem: *Monte Carlo* (MC) and *Temporal Difference* (TD) learning [109]. Both methods are able to learn the value function of the demonstrated policy directly from episodes of experience, with no need to know the transition model. The estimation of the value function is done iteratively, as soon as a state s_t is encountered, according to the exponential average update rule:

$$V^{(t+1)}(s_t) = V^{(t)}(s_t) + \alpha^{(t)}\big(v_t^{\star} - V^{(t)}(s_t)\big),$$

where $v_t^{\star}$ is an estimator of the value function $V(s_t)$ and $\alpha^{(t)} \in \mathbb{R}^{+}$ is the learning rate.

MC approximates the value function with $v_t^{\mathrm{MC}} = \sum_{i=t}^{T(\tau)-1} \gamma^i r_{i+1}$, i.e., the empirical return. MC requires considering a complete episode in order to compute the empirical return, therefore it works only for episodic MDPs. When a state is encountered multiple times on the same episode we can either perform the update only for the first occurrence (*first-visit MC*) or for all the occurrences (*every-visit MC*).

TD, on the other hand, leverages on bootstrapping in order to exploit the current approximation of the value function. The value function is updated using the estimated return (not the empirical return), named *temporal difference target*, defined as $v_t^{\mathrm{TD}(0)} = r_{t+1} + \gamma V^{(t)}(s_{t+1})$. Thus TD, learns online and does not require that the MDP is episodic. The bootstrapping is performed at the first step (TD(0)); the algorithm can be extended delaying the bootstrapping operation and considering the λ-return, $v_t^{\mathrm{TD}(\lambda)}$ (for details, refer to [109]).

It can be proved [109] that first-visit MC provides an unbiased estimator for the value function, while TD estimator is biased, but with lower variance. In practice, TD(λ) is preferred over MC. This is justified by the fact that TD exploits the Markov property using bootstrapping, thus it is more suitable for MDPs, while MC can be used even in non-Markovian environments.

2.3.3 The Control Problem

The most interesting applications of RL require learning the optimal policy in an unknown environment: this problem is known as *model-free control problem.* The RL algorithms, suited for control, can be derived by applying dynamic programming in a sample-based fashion and using the algorithms for prediction presented in the previous section. Essentially, we can apply Monte Carlo or Temporal Difference straightforwardly to perform policy evaluation from samples. However, some adjustments to DP are necessary. First, since the transition model unknown, we need to estimate the Q-function instead of the state value function. Second, the policy improvement step has to be performed with care. Since the Q-function is approximated from samples we cannot directly choose the greedy action, because the estimation could be poor. Indeed, we need to explore the available actions before choosing which is the optimal one. This is an instance of the classical *exploration vs exploitation* dilemma. We would like to make the best decision given current information (exploitation), but we are not confident enough on the current information so we need to gather more (exploration).

More technically, exploration can be enforced using ϵ-greedy policies, i.e., stochastic policies in which a random action is selected with probability ϵ (other approaches use a Boltzmann policy). Still the new policy is an improvement of the old one, as proved by the ϵ-greedy Policy Improvement Theorem [109]. Clearly, as the learning proceeds, we are more confident on the estimated Q-function so we can progressively reduce the exploration level ϵ. The update rule for the Q-function is the following:

$$Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha^{(t)}\big(v_t^{\star} - Q^{(t)}(s_t, a_t)\big).$$

For *Monte Carlo Control* the estimator is given by $v_t^{\mathrm{MC}} = \sum_{i=t}^{T(\tau)-1} \gamma^i r_{i+1}$ while for Temporal Difference Control, named *SARSA* [105], we have $v_t^{\mathrm{TD}(0)} = r_{t+1} + \gamma Q^{\pi(t)}(s_{t+1}, a_{t+1})$. The convergence of both methods is guaranteed provided that the learning rate $\alpha^{(t)}$ fulfills the Robbins-Moore conditions[7] and all the state-action

[7]The Robbins-Moore conditions prescribe that: $\sum_{t=0}^{+\infty} \alpha^{(t)} = +\infty$ and $\sum_{t=0}^{+\infty} \big(\alpha^{(t)}\big)^2 < +\infty$. A simple choice to satisfy the conditions is $\alpha^{(t)} = 1/t$.

Algorithm 2.3 SARSA with ϵ-greedy policy improvement.

```
Input: $\gamma$, $\epsilon^{(0)}$, $\alpha^{(0)}$
Output: $Q^{(T_{max})}$
 1: Initialize $Q^{(0)}$ arbitrarily (only for the first episode)
 2: Initialize $s_0$
 3: Sample an action $a_0$ from the $\epsilon$-greedy policy $\pi^{(0)}(\cdot|s_0)$
        $$\pi^{(0)}(a_0|s_0) = \begin{cases} 1-(1-1/|\mathcal{A}|)\epsilon^{(0)} & \text{if } a_0 = \arg\max_{a'\in\mathcal{A}} Q^{(0)}(s_0,a'), \\ \epsilon^{(0)}/|\mathcal{A}| & \text{otherwise} \end{cases}$$
 4: for $t = 0, 1, ..., T_{max}-1$ do
 5:     Take $a_t$ and measure $s_{t+1}$ and $r_{t+1}$.
 6:     Sample an action $a_{t+1}$ from the $\epsilon$-greedy policy $\pi^{(t+1)}(\cdot|s_{t+1})$
        $$\pi^{(t+1)}(a_{t+1}|s_{t+1}) = \begin{cases} 1-(1-1/|\mathcal{A}|)\epsilon^{(t)} & \text{if } a_{t+1} = \arg\max_{a'\in\mathcal{A}} Q^{(t)}(s_{t+1},a'), \\ \epsilon^{(t)}/|\mathcal{A}| & \text{otherwise} \end{cases}$$
 7:     Update the Q-function
        $$Q^{(t+1)}(s_t,a_t) = Q^{(t)}(s_t,a_t) + \alpha^{(t)}\big(r_{t+1} + \gamma Q^{(t)}(s_{t+1},a_{t+1}) - Q^{(t)}(s_t,a_t)\big)$$
 8:     Update the exploration rate $\epsilon^{(t)}$ to $\epsilon^{(t+1)}$
 9:     Update the learning rate $\alpha^{(t)}$ to $\alpha^{(t+1)}$
10: end for
11: return Q-function $Q^{(T_{max})}$
```

pairs are asymptotically visited infinitely many times [109, 51].

The algorithms we presented for control so far are on-policy, i.e., they aim to learn the value function of the policy used to collect samples. The extension to the off-policy approach can be done by performing Importance Sampling, we will not discuss this aspect in details (see [109]). However, we present briefly the popular *Q-learning* [118]. Q-learning is an off-policy RL algorithm, that can be considered the model-free extension of value iteration. The idea is to apply a sample-based version of the Bellman Optimality Operator that allows learning the optimal Q-function even if the agent is playing a suboptimal policy. The update rule is given by:

$$Q^{(t+1)}(s_t,a_t) = Q^{(t)}(s_t,a_t) + \alpha\big(r_{t+1} + \gamma \max_{a'\in\mathcal{A}} Q^{(t)}(s_{t+1},a') - Q^{(t)}(s_t,a_t)\big).$$

The pseudo-code of a single episode of SARSA and Q-learning, both with ϵ-greedy policy improvement, are reported in Alg. 2.3 and Alg. 2.4 respectively.

Algorithm 2.4 Q-learning with ϵ-greedy policy improvement.

```
Input: $\gamma$, $\epsilon^{(0)}$, $\alpha^{(0)}$
Output: $Q^{(T_{max})}$
1: Initialize $Q^{(0)}$ arbitrarily (only for the first episode)
2: Initialize $s_0$
3: for $t = 0, 1, ..., T_{max} - 1$ do
4:     Sample an action $a_t$ from the $\epsilon$-greedy policy $\pi^{(t)}(\cdot|s_t)$
           $\pi^{(t)}(a_t|s_t) = \begin{cases} 1 - (1 - 1/|\mathcal{A}|)\epsilon^{(t)} & \text{if } a_t = \arg\max_{a' \in \mathcal{A}} Q^{(t)}(s_t, a'), \\ \epsilon^{(t)}/|\mathcal{A}| & \text{otherwise} \end{cases}$
5:     Take $a_t$ and measure $s_{t+1}$ and $r_{t+1}$.
6:     Update the Q-function
           $Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha^{(t)}\big(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^{(t)}(s_{t+1}, a') - Q^{(t)}(s_t, a_t)\big)$
7:     Update the exploration rate $\epsilon^{(t)}$ to $\epsilon^{(t+1)}$
8:     Update the learning rate $\alpha^{(t)}$ to $\alpha^{(t+1)}$
9: end for
10: return Q-function $Q^{(T_{max})}$
```

2.4 Function Approximation

SARSA and Q-learning leverage on a tabular representation for the Q-function which is inapplicable when the state-action space is large, due to memory and time restrictions. Obviously, when the MDP is continuous a tabular representation is impossible. The solution proposed in literature is to estimate value function with function approximation, i.e., $V^\pi(s) \approx \hat{V}(s)$. This approach tends to enforce generalization capabilities of the model and to speed up the computation w.r.t. tabular representation with no significant performance degradation when the approximators are carefully chosen. We can distinguish between *parametric* and *non-parametric* approximators: the former have a set of parameters known a priori, data are used to tune the values of the parameters (e.g., neural networks, radial basis functions); while for the latter the number of parameters is not fixed (e.g., decision trees, nearest neighbors).

2.4.1 Basics of Function Approximation

In the context of function approximation, learning can be defined as the process of selecting a function $\hat{V}$ in a functional space $\mathcal{F}$ in order to minimize a loss function that encodes the fact that we aim to use $\hat{V}$ to approximate the value function V^π. Following the notation presented in [93], we aim to solve the problem:

$$\hat{V} = \arg\min_{f \in \mathcal{F}} \|V^\pi - f\|_q. \tag{2.22}$$

Clearly, like in supervised-learning the choice of the approximation space $\mathcal{F}$ is a fundamental issue [16] which has to account for the well-known *bias-variance trade off.* In the followings, we will focus on parametric function approximation, in which $\mathcal{F} = \mathcal{F}_\Theta = \{f_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^k\}$ is a parametric function space. The problem (2.22) becomes equivalent to finding the optimal parameter:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \|V^\pi - f_{\boldsymbol{\theta}}\|_q. \tag{2.23}$$

Furthermore, we assume that function $f_{\boldsymbol{\theta}}$ is constructed starting from a vector of *features* (or *basis functions*) $\boldsymbol{\phi} = \big(\phi_1, \phi_2, ..., \phi_p\big)$, i.e., functions that map each state to real numbers $\boldsymbol{\phi}(s)$ (or each state-action pair to real numbers $\boldsymbol{\phi}(s, a)$). The functional form of $f_{\boldsymbol{\theta}}$ can vary a lot: from linear models ($f_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$) to complex non linear approximators, like neural networks.

Under certain regularity conditions, prediction in this scenario can be carried out with gradient descent methods employing as loss function equation (2.23) in which we set $q = 2$ and the unknown value function is replaced with the corresponding Monte Carlo or Temporal Difference estimator. The parameter update for *approximate prediction* is given by:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \big(v_t^\star - f_{\boldsymbol{\theta}^{(t)}}(s_t)\big) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{(t)}}(s_t).$$

In case of linear approximators the gradient is simply the feature $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{(t)}}(s) = \boldsymbol{\phi}(s)$. The convergence, however, becomes more critical: while MC approximate prediction is guaranteed to converge, TD might diverge with non linear approximators since bootstrapping becomes unsafe. The control problem is solved by performing approximate prediction for the Q-function and employing an ϵ-greedy policy improvement:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)} \big(v_t^\star - f_{\boldsymbol{\theta}^{(t)}}(s_t, a_t)\big) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}^{(t)}}(s_t, a_t).$$

The convergence guarantees in this case are even worse: SARSA may display a chattering behavior around near-optimal value function, whereas for Q-learning there is no guarantee of convergence even for linear parametrizations. Other approches to overcome this problem have been proposed, such as Gradient Temporal Difference [66], that exploit different update directions to enforce convergence.

The algorithms presented so far perform updates as soon as a sample is drawn (*incremental methods*). This has at least two drawbacks: first, it is computationally inefficient; second, two subsequent samples are strongly correlated. On the contrary, *batch methods* seek to find the best fitting value function once all the data have been collected. The loss function remains the same, but learning is performed over the whole (static) dataset. This allows recovering the optimal parameters even in closed form for some simple approximators, like linear parametrizations. According to which estimator is used to approximate the value function in equation (2.23) we distinguish between: Least Squares Monte Carlo (LSMC), Least Squares Temporal Difference (LSTD). Those algorithms are guaranteed to converge at least for linear patametrizations. The corresponding algorithms for the Q-function are named

Algorithm 2.5 Fitted Q-iteration.

```
Input: $\gamma$, $\mathcal{D}^{(0)} = \{(s_i, a_i, r_i^{(0)}, s_{i+1})\}_{i=1}^N$
Output: $Q^{(T_{max})}$
1: Initialize $Q^{(0)}(s,a) = 0,\ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$
2: for $t = 1, 2, ..., T_{max}$ do
3:     Build the new training set
           $\mathcal{D}^{(t)} = \{(s_i, a_i, r_i^{(t-1)} + \gamma \arg\max_{a \in \mathcal{A}} Q^{(t-1)}(s_{i+1}, a), s_{i+1})\}_{i=1}^N$
4:     Use the regression algorithm on $\mathcal{D}^{(t)}$ to build $Q^{(t)}$
5: end for
6: return Q-function $Q^{(T_{max})}$
```

LSMCQ and LSTDQ. The control problem can be solved by exploiting approximate dynamic programming, e.g., Least Square Policy Iteration (LSPI). A more extensive discussion is reported in [109].

Another class of algorithms that deserve a mention are the *fitted value iteration* methods [41]. We present the fitted Q-iteration that is able to learn the optimal Q-function by solving a sequence of regression problems. Starting from a dataset $\mathcal{D}^{(0)} = \{(s_i, a_i, r_i^{(0)}, s_{i+1})\}_{i=1}^N$, at each iteration t, it builds an approximation of the optimal Q-function $Q^{(t)}$ using a regressor over $\mathcal{D}^{(t)}$. Then $Q^{(t)}$ is exploited to build new the dataset $\mathcal{D}^{(t+1)}$ in which the return in each tuple is replaced with $r_i^{(t)} + \gamma \arg\max_{a \in \mathcal{A}} Q^{(t)}(s_{i+1}, a)$. This method allows to use a large range of regressors, such as kernel averaging, regression trees, fuzzy regression, also neural networks attain good results. The algorithm is reported in Alg. 2.5.

2.4.2 Feature construction

The choice of the basis functions is substantial for the success of any function approximation architecture. Following the presentation in [17], we can distinguish between different classes of basis functions:

- *handcrafted features*: features derived from experience (engineered features). They embed the expert's knowledge of the environment, but they are generally difficult to design. Furthermore, a bad choice has dramatic effects on the accuracy of the approximator [e.g., 108].

- *General function approximators*: in most of the cases are universal approximators, however they do not embed knowledge of the environment and they are prone to the curse of dimensionality. Some classic examples are polynomial basis, radial basis functions (RBF) and Fourier basis [63].

- *Automatic basis construction*: they build the basis functions from the knowledge or the estimation of some properties of the MDP. Some remarkable examples are: Bellman Error Basis Functions (BEBF) [87], Krylov Basis [86], Proto-Value Functions (PVF) [68].

Automatic basis construction has received a lot of attention in literature [e.g., 56, 17, 28, 34]. We will focus just on Proto-Value functions that we will use as a comparison term in the experimental evaluation.

Proto Value Functions

Proto-Value Functions have been introduced in [67, 68] as an automatic feature construction method that is able to build features that represent the underlying structure of the environment. The framework is presented in the context of *Representation Policy Iteration*, an algorithm that simultaneously builds the features and the optimal policy.

The assumption that justifies the method is the following. In many real-world cases the value function is smoother than the reward function, thus it makes sense to build features that assign similar values to states that are reachable one from the other. For this purpose a *connection graph* is constructed out of a Markov chain containing only the states visited by the agent. The graph connects two states s and s' if the agent has experienced a transition from s to s'. The corresponding connection matrix $\mathbf{W}$ is given by:

$$W_{ss'} = \begin{cases} 1 & \text{if agent experimented transition } s \rightarrow s' \\ 0 & \text{otherwise} \end{cases}, \qquad \forall s, s' \in \mathcal{S}. \tag{2.24}$$

The resulting graph is unweighted; other approaches to build the connection matrix, yielding weighed graphs, are possible [68]. Instead of assigning just one if the transition occurred, we can use the number of occurrences or the discounted number of occurrences when dealing with a discounted MDP.

From the connection graph we would like to apply the theory of diffusion maps [24] to extract the features. As diffusion maps require undirected graphs, this matrix $\mathbf{W}$ must be symmetrized by setting $\mathbf{W} \leftarrow \frac{1}{2}(\mathbf{W} + \mathbf{W}^T)$ (other symmerization schemes are possible, like $\mathbf{W}\mathbf{W}^T$). Let $\mathbf{d}$ be the vector of the row sums of matrix $\mathbf{W}$, i.e., $d_s = \sum_{s' \in \mathcal{S}} W_{ss'}$, $\forall s \in \mathcal{S}$, and $\mathbf{D} = \text{diag}(\mathbf{d})$; we can now define the *Random Walk* [23] on the symmetrized graph:

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}. \tag{2.25}$$

The random walk describes completely the Markov chain underlying the graph. We can observe that $\mathbf{P}$ is similar in an intuitive sense to the transition kernel however, as a consequence of symmetrization, it represents the "transition possibilities" rather than the probabilities. Following this spirit, we would like to find a set of basis functions that is able to reconstruct $\mathbf{P}$. This can be done by diagonalizing $\mathbf{P}$ and considering the eigenvectors associated to the largest eigenvalues of $\mathbf{P}$, named ρ_i, in absolute value.

This approach, also known as spectral encoding [11], yields approximation spaces in which Euclidean distances are equivalent to diffusion distances on the connection graph. Essentially, starting from a state space with possibly no metric we have been able to build a distance between states.

However, $\mathbf{P}$ is not symmetric in general and diagonalization might result costly. The approach proposed in [67] is to use a surrogate diffusion operator guaranteed to be symmetric: the Laplacian [23]. We consider the two versions: the *Combinatorial Laplacian* $\mathbf{L}$ and the *Normalized Laplacian* $\boldsymbol{\mathcal{L}}$, defined respectively as:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}, \qquad \boldsymbol{\mathcal{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}. \tag{2.26}$$

It can be proved that $\mathbf{I} - \boldsymbol{\mathcal{L}}$ is similar (in the sense of matrices) to the random walk $\mathbf{P}$ [68]. Thus the eigenvalues of the random walk ρ_i are related to the eigenvalues of the normalized Laplacian λ_i according to: $\rho_i = 1 - \lambda_i$. We define as Proto-Value functions the eigenvectors associated to the smallest eigenvalues of the normalized Laplacian.

So far PVFs were only constructed on graphs defined over the state space. If we are interested in approximating the Q-function we need a set of basis functions defined over the state-action space. In [84] the algorithm of PVFs is extended to recover state-action basis functions. The nodes of the graph are no longer associated to a visited state s, but to a visited state-action pair (s, a). Two approaches are proposed for the construction of state-action graphs: *on-policy* and *off-policy*. The on-policy graph connects two state-action pairs (s, a) and (s', a') if the agent is in state s, performs action a, gets to state s and then selects action a' from there. The off-policy graph instead, connects (s, a) to (s', a') if the agent is in state s, performs action a, gets to state s' and $a' \in \mathcal{A}$, independently on the action taken in s'. The on-policy graph encodes information on the policy used to explore the MDP, whereas the off-policy approach models the random walk, i.e., the environment that lies behind the MDP.

For infinite observation spaces PVFs are also defined by connection graphs. However, differently from the finite case, the construction of this graph is not straightforward. In [68] a symmetrized k-nearest neighbors graph is proposed. Restricting the case to Euclidean state spaces, each visited state $\mathbf{s}$ is connected with the k nearest visited states w.r.t. the Euclidean distance only. After symmetrization, the PVFs ϕ at the visited states are computed. A Nyström extension approximates the PVFs for all unvisited states by computing the mean over the weighted PVFs of the k states:

$$\hat{\phi}(\mathbf{s}) = \frac{\sum_{\mathbf{s}' \in \mathrm{KNN}(k,\mathbf{s})} \mathcal{K}(\mathbf{s}, \mathbf{s}')\phi(\mathbf{s}')}{\sum_{\mathbf{s}' \in \mathrm{KNN}(k,\mathbf{s})} \mathcal{K}(\mathbf{s}, \mathbf{s}')}, \qquad \forall \mathbf{s}' \in \mathcal{S} \subseteq \mathbb{R}^h,$$

where $\mathrm{KNN}(k, \mathbf{s})$ is the set of the k closest visited states to unvisited state $\mathbf{s}$ according to the Euclidean distance. The weights are determined by a Gaussian kernel with bandwidth σ:

$$\mathcal{K}(\mathbf{s}, \mathbf{s}') = \exp\bigg(- \frac{1}{2\sigma^2} \|\mathbf{s} - \mathbf{s}'\|_2^2 \bigg), \qquad \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \subseteq \mathbb{R}^h.$$

2.5 Policy Search

Approximate dynamic programming focuses on the estimation of the optimal value function and derives the optimal policy, no explicit policy representation is considered. In contrast, Policy Search (PS) aims to recover directly the optimal policy, given a predetermined class of approximating policies $\hat{\Pi}$. PS can be seen as the problem of finding the policy in $\hat{\Pi}$ that induces a value function as close as possible to the optimal value function:[8]

$$\hat{\pi} = \arg\min_{\pi \in \hat{\Pi}} \|V^* - V^\pi\|_q. \tag{2.27}$$

When $q = 2$, the optimization problem (2.27) can be interpreted as a projection problem in the space of Markovian stationary policies Π where $\hat{\pi}$ is the orthogonal projection of an optimal policy π^* onto the subspace of policies $\hat{\Pi}$.

PS methods have received a lot of attention in literature in the last decade as an alternative to standard value function methods. Notice, by the way, the analogy between the function approximation problem (2.22) and the policy search problem (2.27); they are both search problems optimizing the same objective function but formulated in different search spaces. The first main advantage of PS over ADP lies in the representational complexity, it is typically simpler to represent the optimal policy rather than the optimal value function [81]. Secondly, in many real-world problems (e.g., robotics) the functional form of the optimal policy is known, up to the value of a vector of parameters [26]. Furthermore, a parametric representation of the policy space allows encoding in the policy prior information on the optimal parameters [26].

Brief overview on Policy Search methods

A variety of methods have been proposed in literature to solve the problem (2.27). The earlier techniques were mainly related to evolutionary computation [78] and gradient methods [121]. In [26] an attempt of classification of PS methods, restricted essentially to parametric methods, is proposed (Figure 2.3). The authors distinguish between *model-based* and *model-free* policy search. The former exploits trajectories to learn a forward model of the dynamics of the environment and the agent behavior; while the latter uses trajectories to directly learn the optimal policy parameters. Among the most popular approaches, for *parametric policy search*, we find the standard gradient optimization [89, 38, 54]. We will focus on them in the followings. Other approaches exploit Monte Carlo Expectation-Maximization techniques [62], variational inference policy search approaches [37] and entropy-based methods, like Relative Entropy Policy Search [88].

[8]We can also resort to a less restrictive demand formulated on the return rather than on the value function: $\hat{\pi} = \arg\min_{\pi \in \hat{\Pi}} \|J^* - J^\pi\|_q$.

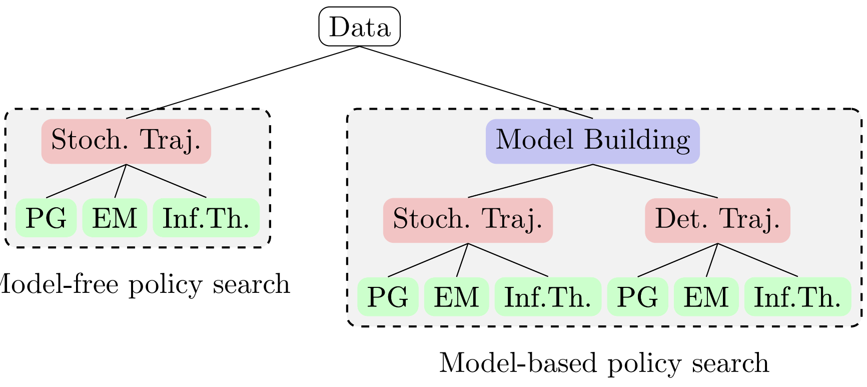

Figure 2.3: A taxonomy of policy search methods (from [26]).

2.5.1 Policy Gradient Methods

Policy Gradient Methods (PGM), introduced by [121], tackle the PS problem as a standard gradient optimization. Clearly, in order to use gradient-based optimization techniques several assumptions on the regularity of the loss function are required. More precisely, PGM assume a predefined *parametric space* of *stochastic* policies:[9]

$$\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subset \mathbb{R}^k\}.$$

Furthermore, we assume that all policies $\pi_{\boldsymbol{\theta}}$ are differentiable w.r.t. $\boldsymbol{\theta}$ for all state-action pairs. PGM search for the optimal policy in the parametric space Π_Θ, i.e., the policy that maximizes the exptected return, which in turn corresponds to finding the optimal policy parameters:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta}). \tag{2.28}$$

The typical approach to optimization problem (2.28) is based on the usage of gradient-based optimization methods. Under regularity conditions, the expression of the gradient of $J(\boldsymbol{\theta})$, named *policy gradient*, is derived in [111].

Theorem 2.4. (Policy Gradient Theorem) *Given an MDP $\mathcal{M}$ and a Markovian stationary stochastic parametric policy $\pi_{\boldsymbol{\theta}} \in \Pi_\Theta$ differentialble w.r.t. to $\boldsymbol{\theta}$ for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$, the gradient of the expected return is given by:*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s,a)\right]. \tag{2.29}$$

[9]The requirement for the policy to be stochastic is necessary in order to compute the policy gradient in a model-free way.

$d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}$ is the γ-discounted state occupancy, as defined in equation (2.8). Clearly, when $\boldsymbol{\theta}$ is an optimal parametrization for policy $\pi_{\boldsymbol{\theta}}$ the policy gradient vanishes. Equation (2.29) can be rephrased into a trajectory-based expression:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \Big[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau) \Big], \tag{2.30}$$

where $p_{\boldsymbol{\theta}}$ is an abbreviation for $p_{\boldsymbol{\theta}}(\cdot|\mu, \pi, \mathcal{P})$, i.e., the trajectory probability density function. This second formulation is typically preferred in practical applications since it does not require to compute explicitly the Q-function. Furthermore, the term $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)$ can be computed without the knowledge of the transition model when the policy is stochastic:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) &= \nabla_{\boldsymbol{\theta}} \log \left(\mu(s_{\tau,0}) \prod_{t=0}^{T(\tau)-1} \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}) P(s_{\tau,t+1}|s_{\tau,t}, a_{\tau,t}) \right) = \\ &= \sum_{t=0}^{T(\tau)-1} \nabla_{\theta} \log \pi_{\theta}(a_{\tau,t}|s_{\tau,t}), \qquad \forall \tau \in \mathbb{T}. \end{aligned} \tag{2.31}$$

Thus we can rewrite equation (2.30) into:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}) \sum_{t=0}^{T(\tau)-1} \gamma^t R(s_{\tau,t}, a_{\tau,t}) \right]. \tag{2.32}$$

Since their introduction these methods have been the center of a large amount of reaserch mainly focused on: gradient estimation [10, 72], variance reduction techniques [119, 42], function approximation techniques [111, 90] and real world applications [62, 103]. The standard *steepest gradient ascent* (also called *vanilla gradient*) [89, 43], albeit simple and efficient, displays substantial issues, like slow convergence, that make it unattractive in practice. Various alternative methods have been proposed in literature, such as *natural gradient ascent* [54, 90] and second-order methods (e.g., *Newton method* [38, 70]). In [38] a comprehensive formulation of the gradient update rule is presented:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha^{(t)} \mathbf{G}(\boldsymbol{\theta}^{(t)})^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)}), \tag{2.33}$$

where $\alpha^{(t)} \in \mathbb{R}^+$ is the learning rate and $\mathbf{G}(\boldsymbol{\theta})$ is a positive definite matrix. The steepest gradient ascent is obtained by setting $\mathbf{G}(\boldsymbol{\theta}^{(t)}) = \mathbf{I}$. In the latter case the convergence is guaranteed almost surely for sufficiently regular loss functions and when the learning rate fulfills the Robbins-Moore conditions. The main issues with the update rule (2.33) are the estimation of the policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (Section 2.5.2) and the choice of matrix $\mathbf{G}(\boldsymbol{\theta})$ (Section 2.5.3). The general scheme of a PGM is reported in Alg. 2.6.

Algorithm 2.6 General setup for policy gradient methods.

```
1: Initialize the policy parameters $\boldsymbol{\theta}^{(0)}$ arbitrarily
2: for $t = 1, 2, ..., T_{max}$ do
3:     Obtain an estimate $\hat{\mathbf{G}}(\boldsymbol{\theta}^{(t)})$ of matrix $\mathbf{G}(\boldsymbol{\theta}^{(t)})$
4:     Obtain a policy gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
5:     Update the policy parameters $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha^{(t)} \hat{\mathbf{G}}(\boldsymbol{\theta}^{(t)})^{-1} \hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
6:     Update the learning rate $\alpha^{(t)}$ to $\alpha^{(t+1)}$.
7: end for
8: return optimal policy parameters $\boldsymbol{\theta}^{(T_{max})}$
```

Compatible function approximation

In order to estimate the gradient we can either exploit the trajectory-based formulation or approximate the Q-function and resort to the Policy Gradient Theorem. Estimating the Q-function might be a challenging task in particular for infinite state-action spaces. However, if the goal is just get a good gradient estimate we can resort to simple function approximators based on the class of *compatible* basis functions defined as $f_{\boldsymbol{\omega}}(s, a) = \boldsymbol{\omega}^T \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)$. In [111] the authors prove that by replacing in equation (2.29) the Q-function with the compatible function that best approximate the Q-function we get an unbiased estimation of the gradient. Clearly, the estimated compatible function might differ significantly from the true Q-function.

2.5.2 Policy Gradient Estimation

The literature on PGM proposed a variety of estimation methods for the policy gradient over the last years. The most prominent approaches, which have been successfully applied to robotics, are Finite-Difference and Likelihood Ratio Methods.

Finite-Difference methods

Finite-difference methods are among the oldest techniques to estimate the policy gradient, invented by the stochastic simulation community. The idea is to approximate the derivative as a quotient of finite increments. The policy parametrization is perturbed multiple times (N) by a small increment $\Delta\boldsymbol{\theta}_i$, $i = 1, 2, ..., N$. The expected return increment for the i-th sample is computed as $\Delta\hat{J}_i = J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i) - J(\boldsymbol{\theta}_i^{ref})$. There are multiple choices for $\boldsymbol{\theta}_i^{ref}$, that correspond to the finite-difference approximators for the derivative [99], like forward-finite-difference ($\boldsymbol{\theta}_i^{ref} = \boldsymbol{\theta}$) and central-finite-difference ($\boldsymbol{\theta}_i^{ref} = \boldsymbol{\theta} - \Delta\boldsymbol{\theta}_i$). The policy gradient estimate is obtained by solving a regression problem yielding:

$$\hat{\nabla}_{\boldsymbol{\theta}}^{\mathrm{FD}} J(\boldsymbol{\theta}) = \left(\boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\Theta}\right)^{-1} \boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta}\hat{\mathbf{J}}, \tag{2.34}$$

where $\boldsymbol{\Delta\Theta} = \left(\Delta\boldsymbol{\theta}_1, \Delta\boldsymbol{\theta}_2, ..., \Delta\boldsymbol{\theta}_N\right)$ and $\boldsymbol{\Delta}\hat{\mathbf{J}} = \left(\Delta\hat{J}_1, \Delta\hat{J}_2, ..., \Delta\hat{J}_N\right)^T$.

Finite-difference methods do not require knowledge of the policy and the differentiability of the policy w.r.t. the parameters, thus they work both with stochastic and deterministic policies without any change and they display good results in simulation environmnets. However, the choice of the perturbation of the parameters is a very difficult problem often with disastrous impact on the learning process when the system goes unstable. In the presence of noise on a real system, the gradient estimate error decreases much slower than for the likelyhood ratio methods. Finally, performance depends highly on the chosen policy parametrization [89].

Likelihood Ratio Methods

Likelihood Ratio Methods exploit the trajectory-based formulation (2.30) in a sample-based fashion to estimate the policy gradient. By observing that a constant reward yields a null policy gradient and exploiting linearity of the expected value we can rewrite equation (2.30) as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \Big[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \big(R(\tau) - b \big) \Big], \tag{2.35}$$

where b is a constant baseline (scalar or vectorial) that can be chosen arbitrarely [121], but typically used to reduce the variance in the estimation of the gradient. The most popular sample-based estimate of the gradient are REINFORCE [121], PGT [110] and G(PO)MDP [10]. REINFORCE just rephrases equation (2.35) in a sample-based form, replacing the expectation with the empirical average:

$$\hat{\nabla}^{RF}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t} | s_{\tau_i,t}) \right) \left(\sum_{t=0}^{T(\tau_i)-1} \gamma^t R(s_{\tau_i,t}, a_{\tau_i,t}) - b \right), \tag{2.36}$$

in which we considered N independent trajectories. The optimal baseline is obtained by minimizing the variance of the gradient estimate (2.36), we can seek for a scalar or component-wise (vectorial) baseline. We indicate with $\mathrm{Var}_\tau \big[\hat{\nabla}^{RF}_{\theta} J(\boldsymbol{\theta}, b) \big]$ the $k \times k$ coviariance matrix of the gradient estimate, where k is the number of parameters. If we minimize the trace of the covariance matrix we obtain a scalar baseline [124], while if we minimize each component of the diagonal we obtain a component-wise baseline made of k elements, one for each parameter [89]:

$$b^*_{\nabla,RF} = \frac{\mathbb{E}_\tau \big[\| \sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t} | s_{\tau,t}) \|_2^2 R(\tau) \big]}{\mathbb{E}_\tau \big[\| \sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t} | s_{\tau,t}) \|_2^2 \big]},$$

$$\big(b^*_{\nabla,RF} \big)_j = \frac{\mathbb{E}_\tau \big[\big(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t} | s_{\tau,t}) \big)^2 R(\tau) \big]}{\mathbb{E}_\tau \big[\big(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t} | s_{\tau,t}) \big)^2 \big]}, \qquad j = 1, 2, \ldots k.$$

Clearly, the baselines are estimated exploiting the straightforward sample-based estimators obtained replacing the expectation with the empirical average. The REINFORCE pseudo-code with optimal component-wise baseline is reported in Alg. 2.7.

Algorithm 2.7 Episodic REINFORCE with optimal component-wise baseline.

```
1: for $n = 1, 2, ..., N$ do
2:     Collect a trajectory $\tau_n$
3:     for all gradient component $j = 1, 2, ..., k$ do
4:         Estimate the optimal baseline
               $b_j^{(n)} = \frac{\sum_{i=1}^{n} \left(\sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})\right)^2 \left(\sum_{t=0}^{T(\tau_i)-1} \gamma^t R(s_{\tau_i,t}, a_{\tau_i,t})\right)}{\sum_{i=1}^{n} \left(\sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})\right)^2}$
5:         Estimate the gradient element
               $\hat{\nabla}_{\boldsymbol{\theta}_j}^{(n)} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=0}^{T(\tau_i)-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})\right) \left(\sum_{t=0}^{T(\tau_i)-1} \gamma^t R(s_{\tau_i,t}, a_{\tau_i,t}) - b_j^{(n)}\right)$
6:     end for
7: end for
8: return gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}}^{(N)} J(\boldsymbol{\theta})$
```

Likelihood ratio methods have several advantages over finite-difference methods. It can be proved [40] that the REINFORCE estimator achieves the fastest possible convergence rate. Furthermore, in practice they require less trajectories w.r.t. finite-difference methods to obtain a good unbiased estimator [10]. However, REINFORCE introduces a high variance in the gradient estimate. Such problem can be mitigated by observing the causal relation between actions and rewards. In particular the reward obtained at time t depends only on the actions performed up to time t, as future actions do not affect the past rewards. This consideration is exploited to define the G(PO)MDP [10] and PGT [110] gradient estimators:

$$\hat{\nabla}_{\theta}^{\mathrm{G(PO)MDP}} J(\boldsymbol{\theta}, b) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T(\tau_i)-1} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'}) \right) \left(\gamma^t R(s_{\tau_i,t}, a_{\tau_i,t}) - b_t \right) \right),$$

$$\hat{\nabla}_{\theta}^{\mathrm{PGT}} J(\boldsymbol{\theta}, b) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T(\tau_i)-1} \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}) \left(\sum_{t'=t}^{T(\tau_i)-1} \gamma^{t'-t} R(s_{\tau_i,t'}, a_{\tau_i,t'}) - b_t \right) \right).$$

In [89] the authors show that, while the algorithms look different, their gradient estimates are equal. The optimal baseline for G(PO)MDP can be obtained by modifying appropriately the sum in the optimal baseline of REINFORCE [89]. Furthermore, while the optimal baseline for REINFORCE is time-invariant, that of G(PO)MDP can also depend on the time step. The pseudo-code of G(PO)MDP with optimal time-variant baseline is reported in Alg. 2.8.

2.5.3 Search Direction Analysis

In the previous section we presented the methods to estimate the policy gradient and reduce the variance of the estimate. All these approaches assume to perform a

Algorithm 2.8 G(PO)MDP with optimal time-variant baseline.

```
1: for $n = 1, 2, ..., N$ do
2:     Collect a trajectory $\tau_n$
3:     for all gradient component $j = 1, 2, ..., k$ do
4:         for all time steps $t'$ do
5:             Estimate the optimal baseline for time step $t'$
                   $$b_{j,t'}^{(n)} = \frac{\sum_{i=1}^{n}\left(\sum_{t=0}^{t'} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})\right)^2 \gamma^{t'} R(s_{\tau_i,t'}, a_{\tau_i,t'})}{\sum_{i=1}^{n}\left(\sum_{t=0}^{t'} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})\right)^2}$$
6:         end for
7:         Estimate the gradient element
                   $$\hat{\nabla}_{\boldsymbol{\theta}_j}^{(n)} J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{t=0}^{T(\tau_i)-1}\left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)\left(\gamma^t R(s_{\tau_i,t}, a_{\tau_i,t}) - b_{j,t}^{(n)}\right)\right)$$
8:     end for
9: end for
10: return gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}}^{(N)} J(\boldsymbol{\theta})$
```

gradient update towards the steepest direction, i.e., matrix $\mathbf{G}(\boldsymbol{\theta})$ in equation (2.33) is the identity matrix. It is well known that exploring different directions, might favor the convergence properties of the gradient optimization methods. The choice of matrix $\mathbf{G}(\boldsymbol{\theta})$, thus, depends on several requirements. Numerical stability, computational complexity of the parameter update and rate of convergence are, in general, reasonable demands. We will focus on two specific choices of matrix $\mathbf{G}$ that lead to two well-known approaches: the *Newton Method* and the *Natural Gradient Ascent.*

Newton Method

In order to cast Newton Method in the form of equation (2.33) we have to set $\mathbf{G}(\boldsymbol{\theta}) = -\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, where $\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the *policy Hessian.* Like for the policy gradient, the policy Hessian admits a state-based formulation and a trajectory-based formulation. The former is shown in [54, 53]:

$$\begin{aligned}\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}\\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}} \Bigg[&\Big(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T + \mathcal{H}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\Big) Q^{\pi_{\boldsymbol{\theta}}}(s,a) + \\ &+ \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} Q^{\pi_{\boldsymbol{\theta}}}(s,a)^T + Q^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T\Bigg].\end{aligned} \tag{2.37}$$

Unfortunately, the computation of the Hessian with equation (2.37) requires the knowledge of the derivative of the Q-function that is difficult to estimate since it requires the transition model. In [38] a trajectory-based formulation of the Hessian, that overcomes this limitation, is provided:

$$\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\Big(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)^T + \mathcal{H}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \Big) R(\tau) \right]. \tag{2.38}$$

This equation is suitable to derive model-free estimators for the policy Hessian. By following the same reasoning used for the policy gradient, it can be proved that also $\mathcal{H}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}$ does not depend on the transition model. For brevity we introduce the symbol $\mathbf{C}_{\boldsymbol{\theta}}(\tau)$:

$$\begin{aligned}\mathbf{C}_{\boldsymbol{\theta}}(\tau) = & \left(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}) \right) \left(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}) \right)^T + \\ & + \sum_{t=0}^{T(\tau)-1} \mathcal{H}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}).\end{aligned}$$

Thus we can rewrite equation (2.38) as:

$$\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\mathbf{C}_{\boldsymbol{\theta}}(\tau) \big(R(\tau) - b \big) \right].$$

The usage of a baseline b might become substantial since the estimate of the policy Hessian results more noisy w.r.t. that of the gradient. REINFORCE-like and G(PO)MDP-like estimators for the policy Hessian can be easily found. We report only the REINFORCE-like policy Hessian estimator:

$$\hat{\mathcal{H}}_{\boldsymbol{\theta}}^{\mathrm{RF}} J(\boldsymbol{\theta}, b) = \frac{1}{N} \sum_{i=1}^{N} \Big(\mathbf{C}_{\boldsymbol{\theta}}(\tau_i) \big(R(\tau_i) - b \big) \Big). \tag{2.39}$$

The optimal baselines, scalar and component-wise, have been derived respectively in [70] and [93]. In those cases the variance of the Hessian matrix is defined as the covariance matrix of its vectorization:[10]

$$\begin{aligned} b^*_{\mathcal{H},\mathrm{RF}} &= \frac{\mathbb{E}_\tau \left[\|\mathrm{vec}\big(\mathbf{C}_{\boldsymbol{\theta}}(\tau)\big)\|_2^2 R(\tau) \right]}{\mathbb{E}_\tau \left[\|\mathrm{vec}\big(\mathbf{C}_{\boldsymbol{\theta}}(\tau)\big)\|_2^2 \right]}, \\ \big(B^*_{\mathcal{H},\mathrm{RF}}\big)_{ij} &= \frac{\mathbb{E}_\tau \left[\big(C_{\boldsymbol{\theta}}(\tau)\big)_{ij}^2 R(\tau) \right]}{\mathbb{E}_\tau \left[\big(C_{\boldsymbol{\theta}}(\tau)\big)_{ij}^2 \right]}, \qquad i,j = 1,2,...k. \end{aligned}$$

Compared to steepest gradient ascent, Newton Method has the highly desirable property of having a quadratic rate of convergence in the neighborhood of a local optimum. However, it is well-known that it suffers from problems that make it either infeasible or unattractive in practice, first of all the inversion of the Hessian matrix.

[10] Let $\mathbf{A}$ be a $n \times m$ matrix, the vectorization of $\mathbf{A}$, indicated as $\mathrm{vec}(\mathbf{A})$ is the vector of nm components obtained by stacking the columns of $\mathbf{A}$, i.e., $A_{ij} = \big(\mathrm{vec}(\mathbf{A})\big)_{i+(j-1)n}$.

Approximate Newton Methods have been introduced to overcome these limitations: they replace the Hessian with a more stable and computationally convenient surrogate (refer to [38] for details). A step towards the search of a more suitable updating direction is done by Natural Gradient methods.

Natural Gradient Ascent

Although variance reduction techniques allow estimating the policy gradient accurately, these methods still tend to perform surprisingly poorly [89]. The underlying reason cannot solely be the variance in the gradient estimate, but rather must be caused by the large plateaus in the expected return landscape where the gradients are small and often do not point directly towards the optimal solution. Furthermore, vanilla gradient methods might result ineffective when the loss function presents multiple maxima or the gradient is not isotropic in magnitude w.r.t. any direction away from its maximum [3]. In order to overcome these problems, which are very frequent, *Natural Gradient* approaches have been introduced [54]. Natural gradient ascent techniques originated in the neural network literature [2] and consider the parameter space as a Riemannian manifold equipped with a proper norm: $\|\boldsymbol{\theta}\|^2_{\mathbf{G}(\boldsymbol{\theta})} = \boldsymbol{\theta}^T \mathbf{G}(\boldsymbol{\theta})\boldsymbol{\theta}$ (where $\mathbf{G}$ is the local Riemann metric tensor) which replaces the Euclidean norm, $\|\boldsymbol{\theta}\|^2_{\mathbf{I}} = \boldsymbol{\theta}^T\boldsymbol{\theta}$. The steepest ascent direction w.r.t. the Riemannian manifold results in the natural gradient direction. For the case of MDPs and in the context of probabilistic models, the most commonly used local metric is given by the *Fisher Information Matrix* (FIM), i.e., $\mathbf{G}(\boldsymbol{\theta}) = \mathbf{F}(\boldsymbol{\theta})$:

$$\begin{aligned}\mathbf{F}(\boldsymbol{\theta}) &= \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}\\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T\right] = \\ &= \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)^T\right],\end{aligned}$$

where we have reported both the state-based and the trajectory-based formulations. Recalling that $p_{\boldsymbol{\theta}}$ is independent from the transition model, we can derive the straightforward estimator:

$$\hat{\mathbf{F}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t})\right)\left(\sum_{t=0}^{T(\tau)-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t})\right)^T.$$

The main advantages of natural gradient ascent over steepest gradient ascent have been summerized by [89]: convergence to a local maximum is guaranteed [2]; by choosing a more direct path to the optimal solution in parameter space, the natural gradient has, from empirical observations, faster convergence and avoids premature convergence of steepest ascent gradient; the natural policy gradient can be shown to be covariant, i.e., independent of the coordinate frame chosen for expressing the policy parameters; as it analytically averages out the influence of the stochastic policy, it requires fewer data points for a good gradient estimate.

General adaptive methods

The convergence properties of gradient ascent can also benefit from general adaptive methods, such as Momentum [98], Adagrad [31], Adadelta [123] and Adam [58], just to mention some of the most popular. Those algorithms, albeit not specifically suited for PGM, can improve and accelerate the convergence of the gradient ascent. We just describe Adam (Adaptive Moment Estimation), that we will use in the experimental evaluation. The idea is to adapt the learning rate to each parameter and according to the local shape of the function. More precisely, Adam keeps an exponentially decaying average of past gradients, similar to momentum:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1-\beta_1)\mathbf{g}^{(t)},$$
$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1-\beta_2)(\mathbf{g}^{(t)})^2,$$

where $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$ are respectively the estimate of the mean and the second momentum of the gradient and $\mathbf{g}^{(t)} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$ in the case of PGM. $(\mathbf{g}^{(t)})^2$ indicates the element-wise square. Since both are initialized to $\mathbf{0}$, to remove the bias toward zeros we need to correct the estimates:

$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1-\beta_1^t}, \qquad \hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1-\beta_2^t}.$$

The update rule is given by:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \frac{\alpha}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon}\hat{\mathbf{m}}^{(t)},$$

where all operations are element-wise. The authors propose default values of 0.9 for β_1, 0.999 for β_2 and $1e-8$ for ϵ. For a synthetic presentation of the various adaptive gradient methods refer to [104].

Chapter 3

Inverse Reinforcement Learning

Reinforcement learning assumes that the implicit notion of "task" (or goal) is encoded in a Markov decision process by means of the reward function. Provided that the environment is equipped with a reward function, reinforcement learning proposes a wide range of algorithms to estimate the optimal policy when the dynamics of the environment is unknown. The MDP formalism can be effectively applied in all the problems in which the reward function is available or easy to specify. However, there are several real-world cases in which manually devising a reward function is a difficult task. The classic example, proposed by Abbeel and Ng in [1], is the *car driving problem*. Specifying a suitable reward function means, in this case, formalizing the intuitive notion of "driving well". Driving is a complex task: a good driver trades off many conflicting desiderata, such as keeping a reasonable speed, staying far from any pedestrian, maintaining a safe following distance. The authors claim that, rather than specifying a reward function, it is easier to generate demonstrations of good drivers, inducing the agent to learn by observation. Learning from expert's demonstrations is called Apprenticeship Learning (AL), also named learning by watching, imitation learning, or learning from demonstration.

The main approaches solving this problem are Behavioral Cloning (BC) [4] and Inverse Reinforcement Learning (IRL) [82]. The former recovers the demonstrated policy by learning the state-action mapping in a supervised learning way, while inverse reinforcement learning aims to learn the reward function that makes the expert optimal. Behavioral cloning is simple, but its main limitation is the intrinsic goal, i.e., to replicate the observed policy. This task has several limitations: it requires a huge amount of data when the environment (or the expert) is stochastic [46]; it does not provide good generalization or a description of the expert's goal. On the contrary, inverse reinforcement learning accounts for generalization and transferability by directly learning the reward function. This information can be transferred to any new environment in which the features are well defined. As a consequence, IRL allows recovering the optimal policy a posteriori, even under variations of the environment. Furthermore, the expert's policy might be very far from the class of policies that the learner can play (e.g., a human demonstrator and a robotic learner).

Therefore, the best policy recovered by behavioral cloning could result worse w.r.t. a policy learned by optimizing the reward function of the problem. IRL has received a lot of attention in the literature and has succeeded in several applications [e.g., 1, 112, 125, 102, 47].

It is important to stress that the reward function is a more powerful and compact information w.r.t. the optimal policy. The transition model can change over time, due to external factors, as a consequence also the optimal policy might change. However, the reward encodes the notion of "task" and thus will remain the same. Knowing the reward, we can recompute the optimal policy associated to the new MDP. Thus, the reward function is a transferable information, i.e., it can be plugged into new problems allowing learning the optimal policy.

However, BC and IRL are tightly related by the intrinsic relationship between reward and optimal policy. The reward function defines the space of optimal policies and to recover the reward it is required to observe/recover the optimal policy.

This chapter is devoted to an exhaustive presentation of the state-of-the-art algorithms for apprenticeship learning. We will mention some of the behavioral cloning approaches, but we will mainly focus on inverse reinforcement learning algorithms. This overview aims to compare the characteristics of the different algorithms and underline their points of strength and weakness, along with the possible extensions and improvements. It is not easy to provide a satisfying classification of AL and IRL algorithms due to the variety of the approaches proposed. We will present them trying to group together algorithms that share similar approaches. Not all the algorithms are going to be discussed with the same level of detail. Within each category we will mainly focus on the most representative algorithm, typically the one that was proposed first, and we will give a high-level overview of the extensions and improvements.

The chapter is organized as follows. We start in Section 3.1 with an overview of the main apprenticeship learning algorithms, focusing on behavioral cloning. The subsequent sections are fully devoted to inverse reinforcement learning. In Section 3.2 we introduce IRL presenting the first algorithm proposed in the literature. In Section 3.3 we focus on the feature expectation algorithms, whereas in Section 3.4 we describe the entropy-based methods. Section 3.5 is devoted to supervised approaches to IRL. In Section 3.6 we illustrate the IRL algorithms based on the policy gradient. Section 3.7 gives an overview of the techniques to build approximation spaces for the reward function. Finally, in Section 3.8, we highlight the most relevant pros and cons of the considered algorithms.

3.1 Apprenticeship Learning and Behavioral Cloning

Learning a mapping between states and actions in order to maximize a given utility function is the basic problem of reinforcement learning. When the utility function is missing, the problem, of course, becomes ill-posed. However, in real-life, a reward

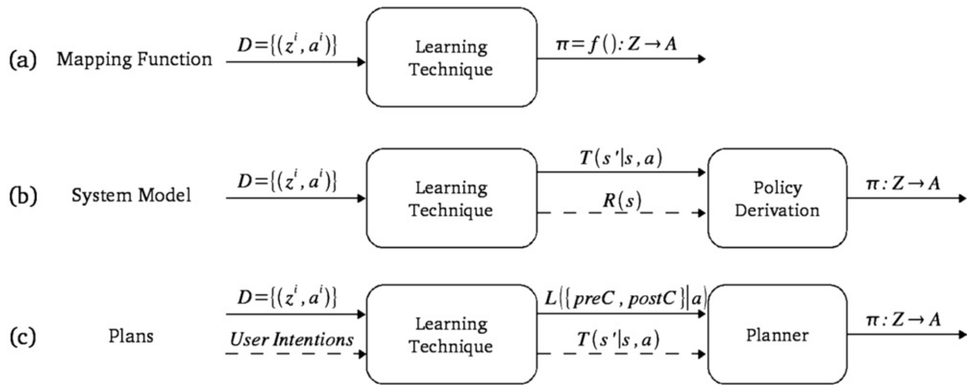

Figure 3.1: Policy derivation using the generalization approach of determining (a) an approximation to the state-action mapping function, (b) a dynamics model of the system and (c) a plan of sequenced actions (from [4]).

function is not typically given explicitly, but living learning agents (e.g., human beings and animals) are able to improve progressively their performance by observing other agents. The key point of this learning framework is imitation. When examples of trajectories generated by experts (or teachers) are available an agent can exploit them to learn the optimal behavior.

In this section, we provide an overview of the different aspects of apprenticeship learning and behavioral cloning algorithms. We mainly follow the presentation proposed in [4]. The apprenticeship learning problem can be stated as follows. Given a dataset of expert's trajectories $\mathcal{D} = \{(s_{\tau_i,0}, a_{\tau_i,0}), ..., (s_{\tau_i,T(\tau_i)}, a_{\tau_i,T(\tau_i)})\}_{i=1}^{N}$ we want to derive, directly or indirectly, a policy that reproduces the demonstrated behavior. We can roughly classify the apprenticeship learning approaches into three categories (Figure 3.1):

- *mapping function*: the demonstrated trajectories are used to derive a policy that reproduces the expert's behavior (this is the case of behavioral cloning);

- *system model*: the data are exploited to recover a model of the environment dynamics and possibly the underlying reward function (this is the case of inverse reinforcement learning); the policy is then computed using classic RL methods;

- *plans*: demonstration data, and often additional user intention information, is used to learn rules that associate a set of pre and post-conditions with each action and possibly a sparsified state dynamics; a sequence of actions is then planned using this information. We will not address this approach in our presentation.

3.1.1 Mapping function

The goal of behavioral cloning via mapping function is to recover a policy that explains the expert's behavior. The policy model has to reproduce the underlying teacher's policy, which is unknown, and to generalize over the set of available training examples such that valid solutions are also acquired for similar states that may not have been encountered during the demonstration [4]. From the supervised learning point of view, we can see the problem of finding a mapping function as either a classification, regression or probability estimation problem.

If our goal is to recover a deterministic policy $\pi(s)$ that for every state $s \in \mathcal{S}$ produces as output the expert's action, we resort to classification and regression. Classification approaches map (possibly) continuous input to a set of classes, thus they are suitable when the action space is finite. Classic approaches make use of Gaussian Mixture Models (GMM) [21], Decision Trees [106], Bayesian Networks [50] and k-Nearest Neighbors [107]. Regression methods, on the contrary, are employed when the action space is continuous. A first notable distinction is whether the mapping function is made run time or prior to run time. The former approach is employed in Lazy Learning [5], where function approximation does not occur until the current observation requires generating the mapping. At the other extreme, there are the methods where the approximation is fully computed before the execution, like Neural Networks (NN) [95].

A different approach to BC is based on the estimation of the parameters of a policy in a given policy space, possibly stochastic. This problem can be formulated as a probability estimation problem. Formally, given a class of parametric policies $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^k\}$ we aim to find the policy that best represents the unknown expert's policy π^E. Let d be a score function, quantifying the proximity of two policies, we want to solve the optimization problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} d(\pi^E, \pi_{\boldsymbol{\theta}}). \tag{3.1}$$

Notice the analogies and differences between this optimization problem (3.1) and the one presented in the context of policy gradient methods (2.28). In both cases we are looking for the parameters of the policy, but the objective functions look different. In policy gradient methods we seek for a policy that maximizes the expected return, i.e., a policy having a performance as close as possible to that of the expert assuming that the expert is optimal. Here, on the contrary, we seek for the parameters making the agent policy as close as possible to the expert's policy. The goal is intuitively the same, but the optimization is carried out in two different spaces.

There are multiple choices for score function d. We observe that for every state $s \in \mathcal{S}$, $\pi(\cdot|s)$ is a probability distribution, thus, concerning the single state we can adopt a distance score between distributions, such as *Total Variation distance* (TV):

$$d_{\text{TV}}\big(\pi_1(\cdot|s), \pi_2(\cdot|s)\big) = \|\pi_1(\cdot|s) - \pi_2(\cdot|s)\|_{TV} = \frac{1}{2}\int_{\mathcal{A}} |\pi_1(a|s) - \pi_2(a|s)| \mathrm{d}a, \quad \forall s \in \mathcal{S},$$

or *Kullback-Leibler divergence* (KL-div):

$$d_{\mathrm{KL}}\big(\pi_1(\cdot|s), \pi_2(\cdot|s)\big) = \mathop{\mathbb{E}}_{a\sim\pi_1(\cdot|s)}\left[\log\frac{\pi_1(a|s)}{\pi_2(a|s)}\right] = \int_{\mathcal{A}} \pi_1(a|s)\log\frac{\pi_1(a|s)}{\pi_2(a|s)}\mathrm{d}a, \quad \forall s \in \mathcal{S}.$$

While total variation is a proper metric, the Kullback-Leibler divergence is not being non-symmetric. However, contrary to total variation, KL-div yields a differentiable loss function and its optimization is related to the problem of finding the maximum likelihood policy parameters. Clearly, the previous score functions can be extended over the whole state space.

We can look at the same probability estimation problem by considering the classic *Maximum Likelihood* (ML) and *Maximum A Posteriori* (MAP) approaches. In the former case, we look for the parameters $\boldsymbol{\theta}^{\mathrm{ML}}$ that maximize the likelihood function:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\tau_1, \tau_2, ..., \tau_N|\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\tau_i), \tag{3.2}$$

where $p_{\boldsymbol{\theta}}$ is the probability density function of the trajectories. It can be proved (see Appendix B.1) that maximizing equation (3.2) is equivalent to minimizing the KL-div between expert's policy and the approximating policy $\pi_{\boldsymbol{\theta}}$. On the contrary, if some knowledge on the expert's parameters is available a Bayesian approach is more suitable. Let $p(\boldsymbol{\theta})$ be a prior, the Maximum A Posteriori estimation aims to maximize the parameter posterior probability:

$$p(\boldsymbol{\theta}|\tau_1, \tau_2, ..., \tau_N) = \frac{p(\tau_1, \tau_2, ..., \tau_N|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\tau_1, \tau_2, ..., \tau_N)} \propto p(\tau_1, \tau_2, ..., \tau_N|\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

3.2 Ng-Russell IRL

Ng and Russell in [82] propose an algorithm (NR-IRL[1]) that is able to recover a reward function assuming to know $\mathcal{M} \setminus \mathcal{R}$, i.e., an MDP without the reward function and an optimal deterministic policy π^*. The method is presented for finite state-action spaces, but can be extended to the continuous MDPs. We assume the transition model $\mathbf{P}$ to be a $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ tensor, $\mathbf{P}_{:a:}$ is the transition kernel ($|\mathcal{S}| \times |\mathcal{S}|$ matrix) associated to action $a \in \mathcal{A}$, $\mathbf{P}_{sa:}$ is the $|\mathcal{S}|$-dimensional vector of the probability distribution of the next state and the reward $\mathbf{r}$ a vector of $|\mathcal{S}|$ components.[2] Given $\mathcal{A} = \{a_1, a_2, ..., a_{|\mathcal{A}|}\}$, without loss of generality, we assume $\pi^*(s) = a_1$, $\forall s \in \mathcal{S}$. The algorithm originates from the following result.

Theorem 3.1. *Given an MDP $\mathcal{M}$ with $\gamma < 1$ and a policy π such that $\pi(s) = a_1$, $\forall s \in \mathcal{S}$. π is an optimal policy if and only if it holds:*

$$\big(\mathbf{P}_{:a_1:} - \mathbf{P}_{:a:}\big)\big(\mathbf{I} - \gamma\mathbf{P}_{:a_1:}\big)^{-1}\mathbf{r} \geq \mathbf{0}, \qquad \forall a \in \mathcal{A} \setminus \{a_1\}. \tag{3.3}$$

[1]The authors did not give a name to their algorithm, we will call it NR-IRL from the initials of the authors' surnames.

[2]The approach can be extended to state-action reward function straightforwardly, by considering $\mathbf{r}^{\pi} = \boldsymbol{\pi}\mathbf{r}$.

Moreover, it is possible to prove that the condition $\big(\mathbf{P}_{:a_1:}-\mathbf{P}_{:a:}\big)\big(\mathbf{I}-\gamma\mathbf{P}_{:a_1:}\big)^{-1}\mathbf{r} > \mathbf{0}$ imposes that π is the only optimal policy. If we observe that $\mathbf{v}^\pi = \big(\mathbf{I}-\gamma\mathbf{P}_{:a_1:}\big)^{-1}\mathbf{r}$, the theorem reduces to the inequality $\mathbf{P}_{:a_1:}\mathbf{v}^\pi \geq \mathbf{P}_{:a:}\mathbf{v}^\pi$, which says that a_1 is better then all the other actions a, i.e., π is optimal (the formal proof can be found in [82]).

For finite MDPs, this theorem provides a characterization of the space of reward functions that make the policy π^* optimal. We first observe here the problem of *reward ambiguity*: given $\mathbf{r}_1$ and $\mathbf{r}_2$ satisfying Theorem 3.1 any linear combination with non-negative coefficients would still be a reward optimized by π. Moreover, this space includes trivial reward functions, like constant functions or even the zero function. To mitigate this problem we could impose that π is the *only* optimal policy, however, rewards arbitrarily close to the constant ones would still be valid solutions. Even if we remove all trivial solutions, it seems that Theorem 3.1 is fulfilled by a large class of functions. How can we select a single reward function? The authors propose to prefer reward functions that make deviations from the optimal policy as costly as possible. They choose to maximize the difference between the expected return of the optimal action and the expected return of the second best action:

$$\sum_{s\in\mathcal{S}}\bigg(Q^{\pi^*}(s,a_1) - \max_{a\in\mathcal{A}\setminus\{a_1\}} Q^{\pi^*}(s,a)\bigg). \tag{3.4}$$

Moreover, they add a regularization term (L1-norm) to favor "simpler" solutions, i.e., reward functions with a small number of non-zero entries. Finally, the problem can be formulated using linear programming and a further constraint to bound the magnitude the recovered reward is added:

$$\begin{aligned}
\underset{\mathbf{r}\in\mathbb{R}^{|\mathcal{S}|}}{\text{maximize}} \quad & \sum_{s\in\mathcal{S}} \min_{a\in\mathcal{A}\setminus\{a_1\}} \left\{\big(\mathbf{P}_{sa_1:}-\mathbf{P}_{sa:}\big)\big(\mathbf{I}-\gamma\mathbf{P}_{:a_1:}\big)^{-1}\mathbf{r}\right\} - \lambda\|\mathbf{r}\|_1 \\
\text{subject to} \quad & \big(\mathbf{P}_{:a_1:}-\mathbf{P}_{:a:}\big)\big(\mathbf{I}-\gamma\mathbf{P}_{:a_1:}\big)^{-1}\mathbf{r} \geq \mathbf{0}, \quad \forall a\in\mathcal{A}\setminus\{a_1\}, \\
& |r_s| \leq R_{max}, \quad \forall s\in\mathcal{S}.
\end{aligned}$$

The algorithm extends, not without issues, to the case of infinite state space, but finite action space. To avoid the usage of calculus of variations the authors propose to approximate the reward function with a linear combination of known and bounded state features $\{\psi_i\}_{i=1}^p$:

$$R(s) = \sum_{i=1}^{p}\omega_i\psi_i(s) = \boldsymbol{\omega}^T\boldsymbol{\psi}(s), \qquad \forall s\in\mathcal{S}. \tag{3.5}$$

By linearity of expectation, the value function attained by policy π is given by $V^\pi(s) = \sum_{i=1}^p \alpha_i V_i^\pi(s)$, where V_i^π is the value function of policy π when the reward is ψ_i. Theorem 3.1 can be easily generalized resulting in the set of linear constraints:

$$\mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a_1)}\big[V^\pi(s')\big] \geq \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}\big[V^\pi(s')\big], \qquad \forall s\in\mathcal{S},\ \forall a\in\mathcal{A}\setminus\{a_1\}. \tag{3.6}$$

However, this set of constraints is infinite, being the state space infinite; the authors suggest circumventing this problem by enforcing the constraint only for a finite subset of states $\mathcal{S}_0 \subset \mathcal{S}$. The second problem is representational and originates from the choice of the function approximator: since the reward is now linearly approximated, it may be no longer possible to find a non-trivial reward function that makes the observed policy optimal. Nevertheless, to avoid falling into trivial solutions we might allow constraints (3.6) being violated and paying a penalty. The final optimization problem can be written as:

$$\begin{aligned} &\underset{\boldsymbol{\alpha}\in\mathbb{R}^p}{\text{maximize}} && \sum_{s\in\mathcal{S}_0} \min_{a\in\mathcal{A}\setminus\{a_1\}} \left\{ c\Big(\mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a_1)} \left[V^{\pi^*}(s')\right] - \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)} \left[V^{\pi^*}(s')\right] \Big) \right\} \\ &\text{subject to} && |\alpha_i| \le 1, \qquad i = 1, 2, ..., p. \end{aligned}$$

c is a penalty function, that the authors suggest to define as: $c(x) = x$ if $x \ge 0$ else $2x$.

NR-IRL represents the first attempt to build an IRL algorithm. The main limitations of this approach are the requirement of the transition model and the laborious extension to continuous domains. The paper [82] proposes an extension to the setting in which the model of the environment in unknown, but it is possible to simulate trajectories on the MDP. We will not present this method, but we will focus on more popular IRL algorithms defined assuming the same setting [1].

3.3 Feature Expectation Algorithms

In this section, we present a class of algorithms that have in common the notion of *feature expectation*, which was introduced in [1]. The main assumption is that the unknown reward function can be expressed as a linear combination of known state (or state-action) feature vectors $\{\psi_i\}_{i=1}^p$, like equation (3.5). Given a policy π, it is possible to compute the expectation of the features with respect to the trajectory distribution that corresponds to the discounted accumulated feature vector $\boldsymbol{\mu}(\pi)$, also called *feature expectations*:

$$\boldsymbol{\mu}(\pi) = \mathbb{E}_\tau \left[\sum_{t=0}^{T(\tau)} \gamma^t \boldsymbol{\psi}(s_{\tau,t}) \right]. \tag{3.7}$$

Since the reward can be expressed as a linear combination of ψ_i, the expected return can be written in terms of the feature expectations:

$$J^\pi = \mathbb{E}_\tau \left[\sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}) \right] = \mathbb{E}_\tau \left[\sum_{t=0}^{T(\tau)} \gamma^t \boldsymbol{\omega}^T \boldsymbol{\psi}(s_{\tau,t}) \right] = \boldsymbol{\omega}^T \boldsymbol{\mu}(\pi). \tag{3.8}$$

Thus, the feature expectations for a given policy π completely characterize the policy in the linear approximation space for the reward function [1]. Clearly, we can

Algorithm 3.1 General scheme for max-margin and projection algorithms.

```
Input: $\epsilon$, $T_{max}$, $\boldsymbol{\psi}$
Output: $\{\boldsymbol{\omega}^{(t')}\}_{t'=1}^{T_{max}}$, $\{\pi^{(t')}\}_{t'=0}^{T_{max}}$
 1: Initialize the policy $\pi^{(0)}$ arbitrarily
 2: Compute (or estimate) $\boldsymbol{\mu}^{(0)} = \boldsymbol{\mu}(\pi^{(0)})$.
 3: for $t = 1, 2, ..., T_{max}$ do
 4:     Compute the margin $m^{(t)}$ and the weight vector $\boldsymbol{\omega}^{(t)}$ using max-margin or projection algorithm.
 5:     if $m^{(t)} < \epsilon$ then
 6:         return weight vectors $\{\boldsymbol{\omega}^{(t')}\}_{t'=1}^{t}$ and the policies $\{\pi^{(t')}\}_{t'=0}^{t-1}$
 7:     end if
 8:     Compute the optimal policy $\pi^{(t)}$ using rewards $R = (\boldsymbol{\omega}^{(t)})^T\boldsymbol{\psi}$.
 9:     Compute (or estimate) $\boldsymbol{\mu}^{(t)} = \boldsymbol{\mu}(\pi^{(t)})$.
10: end for
11: return weight vectors $\{\boldsymbol{\omega}^{(t')}\}_{t'=1}^{T_{max}}$ and the policies $\{\pi^{(t')}\}_{t'=0}^{T_{max}}$
```

compute the feature expectations also for the expert's policy π^E (*expert feature expectation*), i.e., $\boldsymbol{\mu}^E = \boldsymbol{\mu}(\pi^E)$.

Let us consider a weight vector $\boldsymbol{\omega}$, we indicate with $\pi^{\boldsymbol{\omega}}$ the optimal policy induced by the reward function $\boldsymbol{\omega}^T\boldsymbol{\psi}$. We aim to find a reward function inducing an optimal policy as close as possible to the expert's policy. This corresponds to finding the weights $\boldsymbol{\omega}$ that minimize the distance (e.g., Euclidean distance) between the expert feature expectation $\boldsymbol{\mu}^E$ and the feature expectation $\boldsymbol{\mu}(\pi^{\boldsymbol{\omega}})$, i.e., the feature expectation of the optimal policy when the reward is given by $\boldsymbol{\omega}^T\boldsymbol{\psi}$:

$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}\in\mathbb{R}^p} \|\boldsymbol{\mu}^E - \boldsymbol{\mu}(\pi^{\boldsymbol{\omega}})\|_2 \tag{3.9}$$

If the reward R^E optimized by the expert's policy π^E belongs to such space, there exists a weight vector $\boldsymbol{\omega}^E$, such that $R^E = (\boldsymbol{\omega}^E)^T\boldsymbol{\psi}$. We assume to have access to a set of N demonstrations provided by an expert playing the policy π^E. From the available demonstrations we can estimate the expert feature expectation as: $\hat{\boldsymbol{\mu}}^E = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T(\tau)}\gamma^t\boldsymbol{\psi}(s_{\tau_i,t})$ [1].

3.3.1 Max-margin and projection algorithms

Abbeel and Ng [1] propose two iterative algorithms to single out the weight vector $\hat{\boldsymbol{\omega}}$: the *max-margin algorithm* and the *projection algorithm.* Both require having access to an RL algorithm that is able to find the optimal policy under the current reward function (like the ones presented in Chapter 2). Both algorithms share the same steps, except for step 4; the general scheme is reported in Alg. 3.1.

Max-margin algorithm

The *max-margin algorithm* is an iterative algorithm that, at each iteration, computes a weight vector $\boldsymbol{\omega}^{(t)}$ and a policy $\pi^{(t)}$ starting from an initial randomly chosen policy $\pi^{(0)}$. The weight vector $\boldsymbol{\omega}^{(t)}$ is computed by solving the quadratic optimization problem (step 4 in Alg. 3.1):

$$\begin{aligned} \underset{m\in\mathbb{R},\boldsymbol{\omega}\in\mathbb{R}^p}{\text{maximize}} \quad & m \\ \text{subject to} \quad & \boldsymbol{\omega}^T\boldsymbol{\mu}^E \geq \boldsymbol{\omega}^T\boldsymbol{\mu}^{(t')} + m, \quad t' = 0,\dots,t-1. \\ & \|\boldsymbol{\omega}\|_2 \leq 1 \end{aligned}$$

where $\boldsymbol{\mu}^{(t')}$ is the feature expectation computed with policy $\pi^{(t')}$. Thus the algorithm at each iteration finds a weight vector $\boldsymbol{\omega}^{(t)}$ that maximizes the margin $m^{(t)}$ between the expected return of the expert's policy and the expected return of the approximating policies $\pi^{(t')}$. This approach is very similar to the optimization problem posed for Support Vector Machines [16]. The new policy $\pi^{(t)}$ is obtained by using a standard RL algorithms assuming as reward $R^{(t)} = \left(\boldsymbol{\omega}^{(t)}\right)^T\boldsymbol{\psi}$. The algorithm stops when the maximum margin is smaller than a user tuned threshold ϵ.

Projection algorithm

The projection algorithm overcomes one of the main limitations of the max-margin algorithm, i.e., the need of solving a quadratic programming problem. In this case the weight vector is computed as $\boldsymbol{\omega}^{(t)} = \boldsymbol{\mu}^E - \bar{\boldsymbol{\mu}}^{(t-1)}$ where $\bar{\boldsymbol{\mu}}^{(t-1)}$ is the orthogonal projection of $\boldsymbol{\mu}^E$ onto the line passing through the points $\bar{\boldsymbol{\mu}}^{(t-2)}$ and $\boldsymbol{\mu}^{(t-1)}$:

$$\bar{\boldsymbol{\mu}}^{(t-1)} = \bar{\boldsymbol{\mu}}^{(t-2)} + \frac{\left(\boldsymbol{\mu}^{(t-1)} - \bar{\boldsymbol{\mu}}^{(t-2)}\right)^T\left(\boldsymbol{\mu}^E - \bar{\boldsymbol{\mu}}^{(t-2)}\right)}{\left(\boldsymbol{\mu}^{(t-1)} - \bar{\boldsymbol{\mu}}^{(t-2)}\right)^T\left(\boldsymbol{\mu}^{(t-1)} - \bar{\boldsymbol{\mu}}^{(t-2)}\right)}\left(\boldsymbol{\mu}^{(t-1)} - \bar{\boldsymbol{\mu}}^{(t-2)}\right),\ t = 2, 3, ..., T_{max}.$$

The margin is now computed as $m^{(t)} = \|\boldsymbol{\mu}_E - \bar{\boldsymbol{\mu}}^{(t-1)}\|_2$. In the first iteration we set $\bar{\boldsymbol{\mu}}^{(0)} = \boldsymbol{\mu}^{(0)}$ and the stopping condition is preserved.

When the algorithm terminates (the proof of termination and the complexity analysis can be found in [1]) there is at least one policy from the set returned by the algorithm, whose performance under R^E (the reward optimized by the expert) is at least as good as the expert's performance minus ϵ. At this stage we might ask the agent designer to examine the recovered policies and select the one having decent performance. To avoid human intervention, we can select a convex combination of the recovered policies, solving the the problem:

$$\begin{aligned} \underset{\boldsymbol{\lambda}\in\mathbb{R}^{t+1}}{\text{minimize}} \quad & \|\boldsymbol{\mu}^E - \boldsymbol{\mu}\|_2 \\ \text{subject to} \quad & \boldsymbol{\mu} = \sum_{t'=0}^{t} \lambda_{t'} \boldsymbol{\mu}^{(t')}, \\ & \lambda_{t'} \geq 0 \quad t' = 0, 1, ..., t, \\ & \sum_{t'=0}^{t} \lambda_{t'} = 1. \end{aligned}$$

Thus, we look for the mixture of policies whose feature expectation is the closest to the expert feature expectations.

This algorithm does not require the knowledge of the transition model. However, it assumes to have access to an RL algorithm to solve the forward problem as a part of an inner loop. Moreover, an agent sampling the environment is necessary in order to compute the optimal policy at each iteration, making the algorithm inefficient in terms of required samples. The ambiguity problem, in this case, is not treated explicitly. Indeed, the quality of the recovered rewards is tested at each iteration since it is used to recover the optimal policy; this would reasonably discard trivial solutions.

3.3.2 Multiplicative Weights for Apprenticeship Learning

The projection algorithm aims to find a reward function that induces an optimal policy similar to the expert's behavior. This notion of similarity is encoded in the feature expectations. This means that if the expert is imperfect, also the recovered reward function will induce a suboptimal policy. Essentially, the projection algorithm prevents from recovering a reward function that allows learning policies that outperform the expert.

An interesting game-theoretic view of the IRL problem is provided by [113], which leverages on the multiplicative weights algorithm for solving two-player zero-sum games [36]. The authors design an algorithm, named *Multiplicative Weights for Apprenticeship Learning* (MWAL), able to find a policy that performs at least as good as the expert. They propose to solve the minimax problem:

$$v^* = \max_{\pi\in\Pi} \min_{\boldsymbol{\omega}\in\mathbb{R}^p} \left\{ \boldsymbol{\omega}^T \boldsymbol{\mu}(\pi) - \boldsymbol{\omega}^T \boldsymbol{\mu}^E \right\}. \tag{3.10}$$

Since the performance of the optimal policy $\pi^{\boldsymbol{\omega}}$ under parametrization $\boldsymbol{\omega}$ is $J^{\pi^{\boldsymbol{\omega}}} = (\boldsymbol{\omega}^E)^T \boldsymbol{\mu}(\pi^{\boldsymbol{\omega}})$ and $\boldsymbol{\omega}^E$ is unknown, π^* is the policy that maximizes the difference in performance between $\pi^{\boldsymbol{\omega}}$ and the expert's policy with respect to the worst-case possibility for $\boldsymbol{\omega}$. The authors showed that, under some constraints on $\boldsymbol{\omega}$, finding the optimal weights and the optimal apprenticeship policy corresponds to finding the optimal strategy of a suitable zero-sum game. Moreover, by using von Neumann's minimax theorem, the game value v^* is always positive or zero at the minimum, i.e., the policy found by the algorithm will be, in the worst case, as good as that of the expert.

A more detailed analysis of the computational cost and some strategies to solve the problem exploiting linear programming are reported in [113, 112].

3.3.3 Linear Programming Apprenticeship Learning

Linear Programming Apprenticeship Learning (LPAL) [112] is able to find directly a good apprenticeship policy with no need to repeatedly find the optimal policy, via classic iterative techniques (like for the case of MWAL). It does not output a reward function, therefore it is not an IRL algorithm in the strict sense. The idea is to exploit *Bellman flow constraints* in order to identify a policy explaining the expert's behavior. Let us consider the linear program:

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}}{\text{maximize}} \quad & \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} r_{sa}x_{sa} \\ \text{subject to} \quad & \sum_{a\in\mathcal{A}} x_{sa} = \mu_s + \gamma \sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} x_{s'a}P_{sas'}, \quad \forall s \in \mathcal{S}, \\ & x_{sa} \geq 0, \quad \forall s \in \mathcal{S},\, \forall a \in \mathcal{A}. \end{aligned}$$

It is well-known [96] that if $\mathbf{x}^*$ is a solution to this problem, then the policy:

$$\pi(a|s) = \frac{x^*_{sa}}{\sum_{a'\in\mathcal{A}} x^*_{sa'}}, \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A},$$

is an optimal policy, and $\mathbf{x}^*$ is the occupancy measure of π. The constraints of the linear program are often called the Bellman flow constraints.

Let us consider a set of reward features $\{\psi_i\}_{i=1}^p$ the IRL problem is formulated as the problem of finding the occupancy measure $\mathbf{x}$ that satisfies the inequality $J^{\pi_A} \geq J^E + \epsilon$ where J^E is expert's return, J^{π_A} the return of the apprenticeship policy and ϵ is a variable to be optimized. The algorithm is reported in Alg. 3.2.

3.4 Entropy-based algorithms

While feature expectation algorithms share the purpose of finding a reward function inducing a behavior similar to that of the expert, the algorithms we present in this section exploit an entropy-based score to deal with the problem of reward ambiguity. We will discuss in details the original Maximum Entropy IRL [125] and we will outline some extensions.

3.4.1 Maximum Entropy Inverse Reinforcement Learning

As we have already pointed out, the IRL problem is in general ill-posed, since there exist an infinite number of reward functions for the same MDP that make the expert's

Algorithm 3.2 Linear Programming Apprenticeship Learning.

```
Input: $\mathcal{S}$, $\mathcal{A}$, $\mathbf{P}$, $\boldsymbol{\mu}$, $\gamma$, $\boldsymbol{\Psi}$
Output: $\pi^A$
1: Compute or estimate the expert feature expectations $\boldsymbol{\mu}^E$
2: Find the solution $(\epsilon^*, \mathbf{x}^*)$ of the problem:
   $$\begin{aligned} \underset{\epsilon\in\mathbb{R},\mathbf{x}\in\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}}{\text{maximize}} \quad & \epsilon \\ \text{subject to} \quad & \epsilon \le \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \psi_{i,sa} x_{sa}, \qquad i = 1,2,...,p, \\ & \sum_{a\in\mathcal{A}} x_{sa} = \mu_s + \gamma \sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} x_{s'a} P_{sas'}, \quad \forall s \in \mathcal{S}, \\ & x_{sa} \ge 0, \quad \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}. \end{aligned}$$
3: return the apprenticeship policy defined as:
   $$\pi^A(a|s) = \frac{x^*_{sa}}{\sum_{a'\in\mathcal{A}} x^*_{sa'}}, \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}.$$
```

policy optimal. Ziebert et al in [125] solve this *ambiguity* problem by leveraging on the *maximum entropy principle* [52]. The maximum entropy principle applies to a constrained probability estimation problem from data (like estimating the expert's trajectory distribution subject to matching expert's feature expectations). It states that the probability distribution that best represents the available data is the one with the largest entropy. In other words, with no constraint we would select a uniform distribution that is the distribution with maximum possible entropy, whereas under some constraint it can be proved [52] that we need to resort to the maximum likelihood Boltzmann distribution.

Like [1], we assume that the true reward function can be expressed as a linear combination of state features $\boldsymbol{\psi}$. Given a trajectory $\tau \in \mathbb{T}$ we can define the *feature counts* [125]:[3]

$$\boldsymbol{\mu}(\tau) = \sum_{t=0}^{T(\tau)} \gamma^t \boldsymbol{\psi}(s_{\tau,t}), \qquad \forall \tau \in \mathbb{T}. \tag{3.11}$$

The return of trajectory τ can be computed, given a vector of reward weights $\boldsymbol{\omega}$, as $R(\tau) = \boldsymbol{\omega}^T \boldsymbol{\mu}(\tau)$. Given a set of N expert's trajectories, we want to estimate the probability distribution of the demonstrated trajectories, named $p_{\boldsymbol{\omega}}$, constrained to

[3]We use the symbol $\boldsymbol{\mu}$ both for the feature counts and the feature expectations. The reader will realize that the strong connection between the two concepts. Given a distribution of the trajectories, the feature expectation is the expected value of the feature counts.

the fact that the feature expectation matches the average feature count:

$$\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\omega}}} [\boldsymbol{\mu}(\tau)] = \bar{\boldsymbol{\mu}}, \tag{3.12}$$

where $\bar{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{\mu}(\tau)$. According to the maximum entropy principle this is equivalent to finding the maximum likelihood Boltzmann distribution from the sampled trajectories that matches the average feature count $\bar{\boldsymbol{\mu}}$:

$$p_{\boldsymbol{\omega}}(\tau_i) = \frac{e^{\boldsymbol{\omega}^T \boldsymbol{\mu}(\tau_i)}}{\sum_{j=1}^{N} e^{\boldsymbol{\omega}^T \boldsymbol{\mu}(\tau_j)}}, \qquad i = 1, 2, ..., N. \tag{3.13}$$

Looking at the equation (3.13), we notice that trajectories with the same return have the same probability while trajectories with larger return are exponentially preferred. One key benefit of this probabilistic approach is that we implicitly handle the uncertainty and noise in the observed trajectories, potentially leading to obtaining clearer or more robust reward functions.

The equation (3.13) holds only for deterministic MDPs. If we admit stochastic environments we need to resort to a more complex distribution, reported in [125], which is in general intractable. Here we report the approximation that holds when the transition randomness has a limited effect on the expert's behavior:

$$p_{\boldsymbol{\omega}}(\tau_i) = \frac{e^{\boldsymbol{\omega}^T \boldsymbol{\mu}(\tau_i)}}{\sum_{j=1}^{N} e^{\boldsymbol{\omega}^T \boldsymbol{\mu}(\tau_j)}} \mu(s_{\tau_i,0}) \prod_{t=0}^{T(\tau_i)-1} P(s_{\tau_i,t+1} | s_{\tau_i,t}, a_{\tau_i,t}), \qquad i = 1, 2, ..., N. \tag{3.14}$$

As a consequence, $p_{\boldsymbol{\omega}}$ induces a stochastic policy in which the probability of an action is weighted by the expected exponentiated rewards of all trajectories that begin with that action:

$$\pi_{\boldsymbol{\omega}}(a|s) = \sum_{\substack{i:(s_{\tau_i,0}, a_{\tau_i,0})=(s,a), \\ i=1,2,...,N}} p_{\boldsymbol{\omega}}(\tau_i). \tag{3.15}$$

Clearly, the parameters $\boldsymbol{\omega}$ are found in order to enforce constraint (3.12) which can be posed for the available expert's trajectories as a maximum likelihood estimation problem:

$$\underset{\boldsymbol{\omega} \in \mathbb{R}^p}{\text{maximize}} \quad \sum_{i=1}^{N} \log p_{\boldsymbol{\omega}}(\tau_i). \tag{3.16}$$

The problem is convex for deterministic MDPs and the optimum can be obtained using gradient-based optimization methods. The authors suggest adopting exponentiated gradient ascent [59]. The gradient of the objective function is given by:

$$\nabla_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log p_{\boldsymbol{\omega}}(\tau_i) = \bar{\boldsymbol{\mu}} - \sum_{i=1}^{N} p_{\boldsymbol{\omega}}(\tau_i) \boldsymbol{\mu}(\tau_i) = \bar{\boldsymbol{\mu}} - \sum_{s \in \mathcal{S}} d_{\mu}^{\pi_{\boldsymbol{\omega}}}(s) \boldsymbol{\psi}(s), \tag{3.17}$$

where $d_{\mu}^{\pi_{\boldsymbol{\omega}}}(s)$ is the *expected state visitation frequency* and represents the probability of being in a given state under policy $\pi_{\boldsymbol{\omega}}$ (it is equivalent to the undiscounted future

Algorithm 3.3 Dynamic programming algorithm for finding $d_\mu^{\pi_{\boldsymbol{\omega}}}$.

```
Input: $\boldsymbol{\omega}$, $\mathcal{S}$, $\mathcal{A}$, $P$, $\mu$, $T_{max}$
Output: $d_\mu^{\pi_{\boldsymbol{\omega}}}$
 1: $Z(s) = 1 \ \forall s \in \mathcal{S}$                                ▷ Backward Pass
 2: for $t = 1, 2, ..., T_{max}$ do
 3:     for all $s \in \mathcal{S}$ do
 4:         for all $a \in \mathcal{A}$ do
 5:             $Z(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) Z(s') e^{\boldsymbol{\omega}^T \boldsymbol{\mu}(s)}$
 6:         end for
 7:     end for
 8:     for all $s \in \mathcal{S}$ do
 9:         $Z(s) = \sum_{a \in \mathcal{A}} Z(s,a)$
10:     end for
11: end for
12: $\pi_{\boldsymbol{\omega}}(a|s) = Z(s,a)/Z(s) \ \ \forall s \in \mathcal{S},\ \forall a \in \mathcal{A}$ ▷ Local action probability computation
13: $D(s,t) = \mu(s) \ \ \forall t = 1, 2, ..., N$                    ▷ Forward pass
14: for $t = 1, 2, ..., T_{max} - 1$ do
15:     for all $s \in \mathcal{S}$ do
16:         $D(s, t+1) = \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} D(s', t) \pi_{\boldsymbol{\omega}}(a|s) P(s'|s,a)$
17:     end for
18: end for
19: $d_\mu^{\pi_{\boldsymbol{\omega}}}(s) = \sum_{t=1}^{N} D(s,t) \ \ \forall s \in \mathcal{S}$            ▷ Summing frequencies
20: return expected state visitation frequency $d_\mu^{\pi_{\boldsymbol{\omega}}}(s) \ \forall s \in \mathcal{S}$
```

state distribution as defined in equation (2.7)). $d_\mu^{\pi_{\boldsymbol{\omega}}}(s)$ can be efficiently computed using a dynamic programming algorithm that takes transition probabilities and a reward function (Alg. 3.3). The transition probabilities are ideally given, but can be approximated from the observed trajectories.

Maximum Entropy IRL represents a successful example of how to tackle explicitly the problem of reward ambiguity. The maximum entropy principle allows to recover reward functions that result to be resilient to noise and imperfect experts. The major limitation is the need to approximate, at each gradient descent iteration, the state visitation frequency that requires the knowledge of the model or the availability of an agent collecting trajectories. The method provided good results in the context of locomotion, for modeling driver route choices [125].

3.4.2 Deep Maximum Entropy Inverse Reinforcement Learning

The choice of the approximation model has a dramatic impact on the ability of the algorithm to capture the relationship between the feature expectation and the reward. Feature expectation algorithms and Maximum Entropy IRL assume that the mapping from state to reward is simply a weighted linear combination of feature values. This choice, simple and convenient from a computational point of view, is inappropriate when the characteristics of the true reward function can be explained only by complex non-linear models. *Deep Maximum Entropy Inverse Reinforcement Learning* (DeepIRL) [122] employs the principle of maximum entropy using non-linear approximators based on deep neural networks (Fully Convolutional Neural Networks). The approximation space for the reward function is given by a nested non-linear function:

$$R_{\boldsymbol{\omega}_1,\boldsymbol{\omega}_2,\ldots,\boldsymbol{\omega}_n}(s) = g_{1,\boldsymbol{\omega}_1}(g_{2,\boldsymbol{\omega}_2}(\ldots(g_{n,\boldsymbol{\omega}_n}(\boldsymbol{\psi}(s)))\ldots)), \qquad \forall s \in \mathcal{S}$$

The network structure is not predefined and can be adapted to the specific features of the considered domain. The training is performed by back-propagation employing a loss function similar to the one adopted by [125]. It is worth to notice that the Maximum Entropy IRL is a particular case of DeepIRL obtained when the architecture is made of a single layer network with a linear output connected to all the inputs. Furthermore, the algorithmic complexity is independent from the number of demonstrations samples.

3.4.3 Relative Entropy Inverse Reinforcement Learning

Estimating the state visitation frequencies requires the knowledge of the dynamics of the environment. In [18] a model-free IRL algorithm, named *Relative Entropy Inverse Reinforcement Learning* (RE-IRL), is proposed, inspired by the Relative Entropy Policy Search (REPS) approach [88] and based on importance sampling to overcome the need of the transition model. The algorithm minimizes the relative entropy (Kullback-Leibler divergence) between the empirical distribution of the trajectories under a baseline policy and the distribution of the trajectories under a policy that matches the reward feature counts of the demonstrator. The resulting distribution is given by (see [18] for the optimization problem and the derivation of the solution):

$$p_{\boldsymbol{\omega}}(\tau_i) = q(\tau_i)\frac{e^{\boldsymbol{\omega}^T\boldsymbol{\mu}(\tau_i)}}{\sum_{j=1}^{N} e^{\boldsymbol{\omega}^T\boldsymbol{\mu}(\tau_j)}}\mu(s_{\tau_i,0})\prod_{t=0}^{T(\tau_i)-1} P(s_{\tau_i,t+1}|s_{\tau_i,t},a_{\tau_i,t}), \qquad i = 1,2,\ldots,N, \tag{3.18}$$

where $q(\tau)$ is the probability density function of the trajectory distribution under the baseline policy, which depends on the transition model. It is possible to prove that, exploiting importance sampling, we can determine the parameters $\boldsymbol{\omega}$ with no need to estimate the transition model. In [18] $q(\tau)$ is assumed to be a uniform distribution, while other works, like [55], consider $q(\tau)$ as the distribution of the trajectories induced by a near-optimal policy.

3.4.4 Maximum Entropy Semi-Supervised Inverse Reinforcement Learning

A semi-supervised learning approach [20] to IRL, named *Maximum Entropy Semi-Supervised Inverse Reinforcement Learning* (MESS-IRL), is introduced in [6] defining a similarity score between trajectories. The authors extend the original ME-IRL approach adding a pairwise regularization term in the loss function penalizing parametrizations giving different rewards to similar trajectories. The method shares the same limitations of ME-IRL, i.e., the need of the transition model to compute the state visitation frequencies.

3.4.5 Maximum Likelihood Inverse Reinforcement Learning

Maximum Likelihood Inverse Reinforcement Learning (MLIRL) [117, 7]. Similarly to ME-IRL it is based on a probabilistic model, that is exploited to represent the expert's policy instead of the distribution of the trajectories. The reward function is represented by means of a linear combination of known features $\boldsymbol{\psi}$ and the parameters $\boldsymbol{\omega}$ are learned by means of maximum likelihood estimation. The policy is modeled by means of a Boltzmann distribution:

$$\pi_{\boldsymbol{\omega}}(a|s) = \frac{e^{\frac{Q_{\boldsymbol{\omega}}(s,a)}{\beta}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q_{\boldsymbol{\omega}}(s,a')}{\beta}}}, \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}, \tag{3.19}$$

where β is the temperature parameter tuned to manage the exploration/exploitation trade off. Under this policy, the log-likelihood of the N expert's trajectories is given by:

$$\log \mathcal{L}(\boldsymbol{\omega}) = \log p(\tau_1, \tau_2, ..., \tau_N|\boldsymbol{\omega}) = \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \pi_{\boldsymbol{\omega}}(a_{\tau_i,t}|s_{\tau_i,t}). \tag{3.20}$$

The likelihood can be maximized by means of gradient ascent approaches, thanks to the differentiability of the Boltzmann policy. Once the maximum likelihood weights $\boldsymbol{\omega}^{\text{ML}}$ are found, the algorithm outputs the reward $(\boldsymbol{\omega}^{\text{ML}})^T\boldsymbol{\psi}$.

3.5 Planning and Supervised approaches to IRL

Another view of the IRL problem consists in looking at the reward function as a mapping from the state space (or state-action space) to the real numbers. This way, we can formulate the problem of recovering a reward function as a supervised learning problem. This class of methods originates from the *Maximum Margin Planning* (MMP) [101]. We will present MMP and the *Structured Classification based IRL* (SCIRL) proposed in [60, 61]. The most relevant difference w.r.t. the previous approaches is that the set of expert's trajectories $\{\tau_i\}_{i=1}^{N}$ is seen in an "unfolded" form as set of state-action pairs $\{(s_i, a_i)\}_{i=1}^{M}$, where each pair (s_i, a_i) means that the expert in state s_i performed action a_i.

3.5.1 Maximum Margin Planning

Maximum Margin Planning [101] exploits the learning to plan framework in order to make the agent mimicking the expert. Given a set of expert's demonstrations $\{\tau_i\}_{i=1}^{N}$

the learner's goal is to predict the same sequence of actions. This problem can be tackled as a supervised learning problem, assuming that the reward is represented again as a linear combination of given features $\{\psi_i\}_{i=1}^{p}$. The principal innovation w.r.t. feature expectation IRL is the introduction of a loss function $\mathcal{L}$ measuring the distance between the actions taken by the expert and the actions induced by a candidate policy, determined by a specific reward parametrization. In other words, such loss function quantifies the distance between the expert's policy and the optimal policy under a specific reward. Among all feasible solutions, the maximum margin principle looks for a reward function that makes the candidate policy significantly better than alternative policies. Such reward can be found solving the following quadratic programming problem [115]:

$$\begin{aligned} &\underset{\boldsymbol{\omega}\in\mathbb{R}^p,\boldsymbol{\xi}\in\mathbb{R}^N}{\text{minimize}} && \frac{1}{2}\|\boldsymbol{\omega}\|_2^2 + \frac{C}{N}\sum_{i=1}^{N}\beta_i\xi_i^q \\ &\text{subject to} && \boldsymbol{\omega}^T\boldsymbol{\mu}(\tau_i) + \xi_i \geq \max_{j=1,2,\dots,N}\left\{\boldsymbol{\omega}^T\boldsymbol{\mu}(\tau_j) + \mathcal{L}(\tau_j)\right\} \qquad i = 1,2,\dots,N. \end{aligned}$$

The intuition behind these constraints is that we allow only weight vectors $\boldsymbol{\omega}$ for which the candidate policies have higher expected reward than all other policies by a margin that scales with the loss. C is a hyperparameter, $q \in \{1,2\}$ is used to distinguish between L1 and L2 penalty and the coefficients β_i allow weighting differently the available examples. The authors suggest selecting a linear loss function proportional to the state visitation frequencies in order to increment the contribution of the highly visited states. Under linearity of the loss function, the problem can be reformulated as a convex quadratic program [101]. The algorithm is extended to non-linear parametrizations of the reward function in [100] and contextualized in the framework of *learning to search* in [102].

3.5.2 Structured Classification based IRL

While MMP adopts a regression approach to determine the parameters of the reward function, Structured Classification IRL [60] casts the problem of recovering the reward function into a multi-class classification problem. Starting from a linear reward parametrization $R(s,a) = \boldsymbol{\omega}^T\boldsymbol{\psi}(s,a)$, we can derive the Q-function using feature expectations:

$$Q^{\pi}_{\boldsymbol{\omega}}(s,a) = \mathop{\mathbb{E}}_{\tau\sim p(\cdot|s_0=s,a_0=a,\mu,\mathcal{P})}\left[\sum_{t=0}^{T(\tau)}\gamma^t\boldsymbol{\omega}^T\boldsymbol{\psi}(s,a)\right] = \boldsymbol{\omega}^T\boldsymbol{\mu}_{sa}(\pi), \quad \forall s\in\mathcal{S},\ \forall a\in\mathcal{A},$$

where $\boldsymbol{\mu}_{sa}(\pi)$ is the feature expectation restricted to all trajectories starting with state s and action a. The inputs of the classifier are the states, the labels are the actions and the decision rule of the classifier is greedy policy w.r.t. the Q-function, which corresponds to the apprenticeship policy $\pi^C_{\boldsymbol{\omega}}$:

$$\pi^C_{\boldsymbol{\omega}}(s) = \arg\max_{a\in\mathcal{A}} Q^{\pi}_{\boldsymbol{\omega}}(s,a), \qquad \forall s\in\mathcal{S}. \tag{3.21}$$

Thus, the Q-function acts as a classification score function. Clearly, the weights $\boldsymbol{\omega}$ are obtained by training the multi-class classifier.

The algorithm was extended in [61] considering a multi-stage architecture, the approach is named *Cascaded Supervised IRL* (CSI). It consists in two subsequent classification problems. The first aims to recover an approximation of the Q-function with no assumption on the representational model (linear or non-linear). The second consists in recovering the reward function, again with linear or non-linear classifiers. This latter stage requires reversing the Bellman Equation in order to relate the reward with the Q-function.

This class of IRL techniques displayed good results when the expert feature expectation are accurately computed. However, the main assumption is that the expert is optimal and plays a deterministic policy, the case of imperfect expert is not addressed.

3.6 Policy Gradient-based algorithms

This section is devoted to the presentation of a class of IRL algorithms that exploit policy gradient to recover the reward function. First-order necessary conditions have already been used in the field of control to determine a cost function. The *Inverse-KKT optimization* [32] assumes a quadratic cost model $c_{\boldsymbol{\omega}}(s,a) = \boldsymbol{\psi}(s,a)^T \text{diag}(\boldsymbol{\omega})\boldsymbol{\psi}(s,a)$ in which $\boldsymbol{\omega}$ are the parameters to determine. The authors claim that, when the control is optimal, the Karush-Kuhn-Tucker conditions [19] must be fulfilled and derive from this an optimization problem.

The policy gradient has been used in imitation learning [47] to directly learn the parameters of an apprenticeship policy starting from an optimality criterion very similar to the one proposed in [113]. We now focus on *Gradient Inverse Reinforcement Learning* (GIRL) [94], an algorithm that recovers the reward function by minimizing the policy gradient.

3.6.1 Gradient Inverse Reinforcement Learning

The main bottleneck of feature expectation algorithms is the requirement of solving multiple forward RL problems. GIRL considers a (possibly non-linear) parametrization of the reward function $R_{\boldsymbol{\omega}}(s,a)$ depending on the vector of parameters $\boldsymbol{\omega}$. To overcome the ambiguity problem the authors suggest to restrict the domain of the parameters to the fundamental simplex. We assume to have access to a parametric representation of the expert's policy $\pi_{\boldsymbol{\theta}^E}$.[4] When the expert's policy is deterministic the model must be available or the policy must be forced to be stochastic. For continuous state-action domains, the latter approach can be easily implemented by adding zero-mean Gaussian noise. Instead, the Boltzmann model is suited for dis-

[4]If the expert is playing a non-parametric policy or only trajectories are available the policy $\pi_{\boldsymbol{\theta}^E}$ can be estimated using behavioral cloning.

crete actions, because the stochasticity can be regulated by varying the temperature parameter. Under regularity assumptions on the policy, we can write the policy gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) = \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}^E}} \\ a \sim \pi_{\boldsymbol{\theta}^E}(\cdot|s)}} \Big[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}^E}(a|s) R_{\boldsymbol{\omega}}(s,a) \Big]. \tag{3.22}$$

Clearly, the reward parametrization optimized by the expert $\boldsymbol{\omega}^E$ is the one making the expert's policy $\pi_{\boldsymbol{\theta}^E}$ a stationary point of $J(\boldsymbol{\theta}, \boldsymbol{\omega}^E)$. However, it might be the case the chosen reward parametrization is not powerful enough to represent the reward optimized by the expert. Thus, the authors suggest minimizing the norm of the policy gradient:

$$\boldsymbol{\omega}^A = \mathop{\arg\min}_{\boldsymbol{\omega} \in \mathbb{R}^p} \big\| \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) \big\|_q. \tag{3.23}$$

An important property of GIRL is that the objective function is convex whenever the parametric reward model is convex w.r.t. $\boldsymbol{\omega}$ [94]. From an analytic point of view, $\boldsymbol{\omega}^A$ represents the minimum norm gradient, i.e., the reward that induces the minimum interest in changing in the policy parameters.

The authors also provide a multi-objective interpretation of the problem when the reward model is linear, i.e., $R_{\boldsymbol{\omega}}(s,a) = \boldsymbol{\omega}^T \boldsymbol{\psi}(s,a)$. In this case, the expected return can be decomposed as:

$$J(\boldsymbol{\theta}, \boldsymbol{\omega}) = \sum_{i=1}^{p} \boldsymbol{\omega}_i \mathbb{E}_\tau \Bigg[\sum_{t=0}^{T(\tau)-1} \gamma^t \boldsymbol{\psi}_i(s_{\tau,t}, a_{\tau,t}) \Bigg] = \sum_{i=1}^{p} \boldsymbol{\omega}_i J_i(\boldsymbol{\theta}) = \boldsymbol{\omega}^T \mathbf{j}(\boldsymbol{\theta}). \tag{3.24}$$

This equation can be interpreted as a weighted sum of the components of the objective vector $\mathbf{j}(\boldsymbol{\theta}) = \big(J_1(\boldsymbol{\theta}), J_2(\boldsymbol{\theta}), ..., J_p(\boldsymbol{\theta})\big)^T$. Thus, maximizing $J(\boldsymbol{\theta}, \boldsymbol{\omega})$ corresponds to finding the parameters that make the expert Pareto optimal.

The main innovation proposed by this work is the complete removal of the need of solving the direct RL problem as an internal step of the IRL algorithm. The approach yields good results when compared with structured IRL approaches on the benchmark problems Linear Quadratic Gaussian Regulator [30] and Mountain Car [77]. Nevertheless, in more complex problems it might be the case that a first-order condition is not sufficient to ensure that $\boldsymbol{\omega}^A$ is a maximum of J. Therefore, we should resort to second-order criteria in order to discard minima and saddle points.

3.7 Feature Construction in IRL

Almost all IRL approaches presented so far share the necessity to define a priori a set of handcrafted features, spanning the approximation space of the reward functions. This may affect significantly the performance of the algorithm, since a wrong choice of the feature space compromises the quality of the recovered reward. The importance of incorporating feature construction in IRL has been known in literature for a while. A wide range of approaches have focused on exploiting the capability of neural networks and in particular those of deep learning to automatically construct features out of the provided data [35, 47, 45]. However, by exploiting a “black box” approach, these methods sometimes do not take advantage from the structure of the underlying MDP. The problem of feature construction has been explicitly addressed only by FIRL (*Feature Construction for Inverse Reinforcement Learning*) [64] in which features are iteratively generated in order to better describe regions where the old features were too coarse.

3.7.1 Feature Construction for Inverse Reinforcement Learning

FIRL [64] iteratively constructs both the feature set and the reward function, as a linear projection in the feature space. The algorithm alternates between *optimization* and *fitting* phases. The optimization phase aims to recover a reward function, from the current feature set as a linear projection, such that the associated optimal policy is consistent with the demonstrations. In the fitting phase new features are created (using a regression tree) in order to better explain regions where the old features were too coarse. The features ψ correspond to the leaf nodes of the regression tree and are represented as a set of states. Thus, the current set of features $\Psi^{(t)}$ is a set of sets of states.

Optimization step

The t-th optimization step consists in computing the reward function $\mathbf{r}^{(t)}$ using expert's trajectories and the current set of features $\Psi^{(t-1)}$. This reward function is chosen so that the optimal policy under the reward is consistent with the expert's trajectories and so that it minimizes the sum of squared errors between $\mathbf{r}^{(t)}$ and its projection onto the linear basis of features $\Psi^{(t)}$. This is formalized introducing the matrices:

$$\mathbf{T}_{R\to\Psi}(\psi, s) = \begin{cases} |\psi|^{-1} & \text{if } s \in \psi \\ 0 & \text{otherwise} \end{cases}, \mathbf{T}_{\Psi\to R}(s, \psi) = \begin{cases} 1 & \text{if } s \in \psi \\ 0 & \text{otherwise} \end{cases}, \quad \forall s \in \mathcal{S},\ \forall \psi \in \Psi.$$

Thus, $\mathbf{T}_{R\to\Psi}\mathbf{T}_{\Psi\to R}\mathbf{r}$ is a vector where the reward in each state is the average over all rewards in the feature that state belongs to. The optimization problem is the following:

$$\begin{aligned} \underset{\mathbf{r}\in\mathbb{R}^{|\mathcal{S}|}}{\text{minimize}} \quad & \|\mathbf{r} - \mathbf{T}_{R\to\Psi}\mathbf{T}_{\Psi\to R}\mathbf{r}\|_2^2 \\ \text{subject to} \quad & \pi^E(s) = a, \qquad \forall (s,a) \in \mathcal{D}. \end{aligned}$$

However, the constraint is not convex thus we must resort to a relaxation of the optimal Bellman equation (similarly to the linear programming problem [96]). These constraints operate both on observed and unobserved state-action pairs and require the knowledge of the transition kernel.

Fitting step

Once the reward function $\mathbf{r}^{(t)}$ for the current feature set $\Psi^{(t-1)}$ is computed, we formulate a new feature hypothesis $\Psi^{(t)}$ that is able to better represent this reward function. The objective of this step is to construct a set of features that gives greater resolution in regions where the old features are too coarse, and lower resolution in regions where the old features are unnecessarily fine. $\Psi^{(t)}$ are obtained by building a regression tree for $\mathbf{r}^{(t)}$ over the state space $\mathcal{S}$. This step requires computing the optimal policy in order to evaluate the consistency with the demonstrations. The authors suggest solving this forward problem using value iteration.

The detailed description of the algorithm can be found in [64]. One of the most interesting properties of the reward functions produced by FIRL is *transferability*. A reward function is transferable when it can be plugged into a new environment and allows recovering the optimal policy. In real-world applications of IRL the issue of transferability becomes substantial. In the highway driving, for instance, we expect the agent to behave well even if the traffic pattern has never been experimented by the expert. This problem has been addressed only in FIRL for the Grid World problem and in [65] for the simulated highway driving. These works demonstrate that a non-accurate choice of the reward features might make the agent perform poorly on transfer environments (obtained in these cases by placing distractors on the original one) even if it performs well in the training environment.

The issue of transferability is analysed also in *Gaussian Process Inverse Reinforcement Learning* (GPIRL) [65], a Bayesian approach for learning non-linear rewards. The authors leverage on Gaussian processes to learn the reward as a non-linear function, while also determining the relevance of each feature to the expert's policy.

3.7.2 Deep Learning approaches

We already mentioned in Section 3.4.2 an IRL algorithm based on a deep architecture, DeepIRL [122]. However, this method requires specifying a set of basis functions that combined non-linearly define the approximation space for the reward function. Here we focus on the approaches that aim, through deep learning architectures, to construct both features and the reward function. We will mainly discuss *Guided Cost Learning* [35] and mention *Deep Q-learning from Demonstrations* (DQfD) [45].

Guided Cost Learning [35] addresses two key challenges: first, the need for informative features and effective regularization to impose structure on the cost, and second, the difficulty of learning the cost function under unknown dynamics for high-dimensional continuous systems. For the former goal, the authors exploit the neural networks capabilities to represent non-linear models, whereas for the latter they resort to a sample-based approximation of the Maximum Entropy IRL. The neural network has the double objective to represent the cost function and suggest the optimal action. At each backpropagation step the parameters of the (non-linear) cost function are updated while optimizing the maximum entropy cost function (3.16). The recovered cost function, however, is hardly transferable since, as hypothesized by the authors, training the policy on a new instance of the task provides the algorithm with additional information about task variation, producing a better cost function and reducing overfitting. While the expressive power of nonlinear cost functions brings a range of benefits, it introduces significant model complexity requiring to apply regularization techniques. The approach was successful on several robotic simulators and on real-world robotic applications.

Deep Q-learning from Demonstrations (DQfD) [45] is a hybrid algorithm that uses expert's demonstration to speed up the forward learning. The method is built on top of Deep Q-Networks [75] and combines classic temporal difference learning with large-margin classification of the demonstrated actions. The architecture showed better performance w.r.t. Deep Q-Networks in terms of training time, when tested on Atari games.

3.7.3 Bayesian approaches

In the context of feature construction for IRL other methods have been proposed. We mention the *Bayesian Nonparametric Feature Construction for Inverce Reinforcement Learning* BNP-FIRL [22]. BNP-FIRL works under the assumption that the features are binary; it builds composite features as logical conjunctions of the predefined atomic features. The reward function is then represented as a linear combination of the composite features. Being the logical conjunction a non-linear operation, BNP-FIRL can improve the representational power of the linearly parametrized IRL methods.

3.8 Discussion

In this chapter, we presented several approaches to the problem of imitation learning, focusing on the inverse reinforcement learning algorithms. In this section, we perform a comparative analysis highlighting the most relevant strengths and weaknesses of the presented algorithms.

A first notable dichotomy is between model-based and model-free algorithms. The former, such as NR-IRL or LPAL, are hardly deployable to real-world applications, since they require to have access to the environment transition model. The estimation of the transition model is particularly challenging when the state (or action) space is continuous, but it is even more difficult if we have only expert's demonstrations at our disposal. An expert is typically an optimal (or near-optimal) agent, thus it is going to explore a very limited region of the state space, resulting in a poor estimation of the transition model. As an alternative we could collect trajectories using an explorative (e.g., random) policy, but this would potentially increase the execution cost. Other methods, like feature expectation algorithms, do not require explicitly the transition model even if they need to sample the environment in order to solve the direct reinforcement learning problem. These algorithms belong to a gray zone in between model-based and model-free.

Few algorithms, among which we illustrated, can be defined really model-free. Besides deep learning approaches, only GIRL is able to recover a reward function from a set of expert's trajectories only. Furthermore, several algorithms cannot be extended straightforwardly to infinite state spaces. Clearly, general purpose discretization techniques can be applied whenever the method does not consider explicitly this possibility.

Another relevant issue underlined in this chapter is reward ambiguity. The choice of a reward function among all feasible ones is, of course, related to the notion of optimality we decide to employ. Indeed, literature has not treated this issue in a comprehensive way yet. Intuitively, we aim to find a reward with *good* learning properties, i.e., a reward able to learn the optimal policy quickly. However, the learning speed is an algorithm-dependent property. Only ME-IRL dealt with this issue explicitly, proposing an optimality criterion based on the maximum entropy principle.

The choice of the reward approximation space is a further issue. Early IRL methods had a preference for linear parametrizations, most of them work with a linear parametrizations only. Resorting to linear models is risky for, essentially, two reasons. First, a linear model might not be able to learn the possibly complex dynamics going on in the features. Second, a bad choice of the features might harm the process of reward recovery. Research tried to tackle both problems. Algorithms able to generate automatically the feature space, like FIRL, overcome the latter problem. Indeed, with a good feature construction mechanism, even a linear model is effective. Instead, linear approximators are replaced with expressive non-linear neural networks in deep learning architectures which are able to deal with both problems at once.

Finally, we need to think about the usage of the reward function once it has been recovered. In real-world applications we have to be able to plug the reward function in new environments and still manage to recover the optimal policy. Therefore, the transferability of the reward is a required property. Nevertheless, only FIRL and GPIRL, focused on this aspect.

Chapter 4

Compatible Reward Inverse Reinforcement Learning

Apprenticeship learning aims to learn to perform a task by observing only expert's demonstrations. We consider the settings where only expert's demonstrations are given, no information about the dynamics and the objective of the problem is provided (e.g., reward) or ability to query for additional samples. In Chapter 2, we presented extensively several algorithms that roughly fall into two classes: behavioral cloning and inverse reinforcement learning. We also pointed out few issues of both classes not completely solved yet. First, several IRL methods require solving the forward problem as part of an inner loop [e.g., 1, 112]. Literature has extensively focused on removing this limitation [61, 92, 94] in order to scale IRL to real-world applications [35, 46, 45]. Second, IRL methods generally require designing the function space by providing *features* that capture the structure of the reward function [e.g., 1, 101, 112, 61, 6, 94]. This information, provided in addition to expert's demonstrations, is critical for the success of the IRL approach. The issue of designing the function space is a well-known problem in supervised learning, but it is even more critical in IRL since a wrong choice might prevent from finding good solutions to the IRL problem [82, 79], especially when linear reward models are considered. The importance of incorporating feature construction in IRL has been known in literature for a while [1], but, as far as we know, it has been explicitly addressed only in [64]. Recently, IRL literature, by mimicking supervised learning one, has focused on exploiting neural network capability of automatically constructing relevant features out of the provided data [35, 47, 45]. By exploiting a "black-box" approach, these methods do not take advantage of the structure of the underlying Markov decision process (in the phase of feature construction).

We present an IRL algorithm that constructs reward features directly from expert's demonstrations. The proposed algorithm is model-free and does not require solving the forward problem (i.e., finding an optimal policy given a candidate reward function) as an inner step. The *Compatible Reward Inverse Reinforcement Learning*

(CR-IRL) algorithm builds a reward function that is *compatible* with the expert's policy. It mixes BC and IRL in order to recover the "optimal" and most "informative" reward function in the space spanned by the recovered features. Inspired by the gradient-minimization IRL approach proposed in [94], we focus on the space of reward functions that makes the policy gradient of the expert vanish. Since a zero gradient is only a necessary condition for optimality, we consider a second order optimality criterion based on the policy Hessian to rank the reward functions and finally select the best one (i.e., the one that penalizes the most a deviation from the expert's policy).

The chapter is organized as follows. In section 4.1 we provide a high level description of CR-IRL. In Section 4.2 we discuss the problem of recovering a parametric representation of the expert's policy. Section 4.2 recalls some concepts about policy gradient and describes the process of feature extraction for the Q-function, particularizing the procedure for the discrete and continuous domains. Section 4.3 illustrates how to derive the space of reward features from the Q-function features. Section 4.5 is devoted to the second-order criteria used to single out a reward function from the set of reward features. Finally, in Section 4.6 we perform the computational analysis of the algorithm.

4.1 Algorithm Overview

CR-IRL takes as input a parametric policy space Π_Θ and a set of rewardless trajectories from the expert policy π^E, denoted by $\mathcal{D}$. CR-IRL is a non-iterative algorithm that recovers a reward function for which the expert is optimal without requiring to specify a reward function space. It starts building the features $\{\phi_i\}$ of the value function that are compatible with policy π^E, i.e., that make the policy gradient vanish. This step requires a parametric representation $\pi_{\boldsymbol{\theta}^E} \in \Pi_\Theta$ of the expert's policy which can be obtained through behavioral cloning.[1] The choice of the policy space Π_Θ influences the size of the functional space used by CR-IRL for representing the value function (and the reward function) associated with the expert's policy. In order to formalize this notion, we introduce the *policy rank*, a quantity that represents the ability of a parametric policy to reduce the dimensions of the approximation space for the value function of the expert's policy. Once these value features have been built, they can be transformed into reward features $\{\psi_i\}$ by means of the Bellman equation [96] (model-based) or reward shaping [81] (model-free). All the rewards spanned by the features $\{\psi_i\}$ satisfy the first-order necessary optimality condition [83], but we are not sure about their nature (minima, maxima or saddle points). The final step is thus to recover a reward function that is maximized by the expert's policy. This is achieved by considering a second-order optimality condition, with the idea

[1]We want to stress that our primal objective is to recover the reward function since we aim to explain the motivations that guide the expert and to transfer it, not just to replicate the behavior. We aim to exploit the synergy between BC and IRL.

that we want the reward function that penalizes the most a deviation from the parameters of the expert's policy $\pi_{\boldsymbol{\theta}^E}$. This criterion is similar in spirit to what done in [82, 1, 101], where the goal is to identify the reward function that makes the expert's policy better than any other policy by a margin. The algorithmic structure is reported in Alg. 4.1.

IRL literature usually considers two different settings: optimal or sub-optimal expert. This distinction is necessary when a fixed reward space is provided. In fact, the demonstrated behavior may be not optimal under the considered reward space. In this case, the problem becomes somehow not well defined and additional "optimality" criteria are required [79]. This is not the case for CR-IRL that is able to automatically generate the space of reward functions that make the policy gradient vanish, containing also reward functions under which the recovered expert's policy $\pi_{\boldsymbol{\theta}^E}$ is optimal.

4.2 Expert's policy estimation

In general the full expert's policy is not available but it can be estimated from samples exploiting the available expert's trajectories. In this section, we discuss the approach we adopted to determine a parametric representation of the expert's policy. Given a set of expert's demonstrations $\mathcal{D} = \left\{(s_{\tau_i,0}, a_{\tau_i,0}, \dots, s_{\tau_i,T(\tau_i)}, a_{\tau_i,T(\tau_i)})\right\}_{i=1}^{N}$, and a parametric policy space $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^k\}$ we need to estimate the parameters of a policy that approximates the expert's policy. As we have discussed in Section 3.1, this is a probability estimation problem and we resort to the minimization of the Kullback-Leibler divergence (KL-div). Given two policies $\pi_1(\cdot|s)$ and $\pi_2(\cdot|s)$ in a state $s \in \mathcal{S}$, the KL-div is given by:

$$d_{\mathrm{KL}}(\pi_1(\cdot|s), \pi_2(\cdot|s)) = \mathop{\mathbb{E}}_{a\sim\pi_1(\cdot|s)}\left[\log \frac{\pi_1(a|s)}{\pi_2(a|s)}\right]. \tag{4.1}$$

However, we would like to obtain a score function for the overall policy, not just in a single state. Thus, we can alternatively integrate over the state space or compute the KL-div over the trajectory space:

$$d_{\mathrm{KL}}(\pi_1, \pi_2) = \mathop{\mathbb{E}}_{\tau\sim p_1}\left[\sum_{t=0}^{T(\tau)-1} \log \frac{\pi_1(a_{\tau,t}|s_{\tau,t})}{\pi_2(a_{\tau,t}|s_{\tau,t})}\right], \tag{4.2}$$

where $p_1(\tau)$ is the probability density function of trajectory $\tau \in \mathbb{T}$ when policy π_1 is played. The latter can be estimated from samples via the Monte Carlo estimator:

$$\hat{d}_{\mathrm{KL}}(\pi_1, \pi_2) = \frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{T(\tau_i)-1} \log\left(\frac{\pi_1(a_{\tau_i,t}|s_{\tau_i,t})}{\pi_2(a_{\tau_i,t}|s_{\tau_i,t})}\right), \tag{4.3}$$

where the M independent trajectories are collected with the policy π_1.

Thus, we can determine the parameters of policy $\pi_{\boldsymbol{\theta}}$ by minimizing $\hat{d}_{KL}(\pi^E, \pi_{\boldsymbol{\theta}})$. Notice that the gradient of equation (4.3) does not depend on the expert's policy π^E which is usually unknown, therefore the minimization can be carried out with standard gradient descent techniques. Minimizing the estimator $\hat{d}_{KL}(\pi^E, \pi_{\boldsymbol{\theta}})$ corresponds to finding the Maximum Likelihood (ML) parameters, given the set of independent expert's trajectories (see Appendix B.1 for the derivation):

$$\boldsymbol{\theta}^{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{M} \sum_{t=0}^{T(\tau_i)-1} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}). \tag{4.4}$$

Clearly, the choice of the parametric policy space has an impact at least for what concerns the ability to represent the expert's policy. It is important to point out that policy $\pi_{\boldsymbol{\theta}^{\mathrm{ML}}}$ will not be used to collect samples, as we are assuming to have the provided trajectories only, but just to extract the features for the reward function, as explained in the following sections. Therefore, even an imperfect choice of the policy space has a limited effect on the performance of the algorithm. Nevertheless, a highly inaccurate choice of the policy space might produce a non-optimal approximation space, compromising the effectiveness of the algorithm.

It is worth noting that the ML policy $\pi_{\boldsymbol{\theta}^{\mathrm{ML}}}$ is not necessarily the optimal policy $\pi_{\boldsymbol{\theta}^*}$ in the policy space Π_Θ, i.e., the policy that maximizes the expected return:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta}).$$

When the policy space Π_Θ is sufficiently expressive to contain the expert's policy, i.e., there exists $\boldsymbol{\theta}^E \in \Theta$ such that $\pi^E = \pi_{\boldsymbol{\theta}^E}$, ML policy converges almost surely to the expert's policy. On the contrary, when the expert's policy falls outside Π_Θ, $\pi_{\boldsymbol{\theta}^{\mathrm{ML}}}$ converges to the minimum KL-div policy. For brevity in the followings, we will indicate with $\pi_{\boldsymbol{\theta}}$ the parametric representation of the expert's policy.

4.3 Expert's Compatible Value Features

In this section, we present the procedure to obtain the set $\{\phi_i\}_{i=1}^p$ of *Expert's COmpatible Q-features* (ECO-Q) that make the policy gradient vanish.[2] We recall from Chapter 2 the policy gradient and the associated first-order optimality condition. We will indicate with $\mathbb{T}$ the set of all possible trajectories, $p_{\boldsymbol{\theta}}(\tau)$ the probability density of trajectory τ and $R(\tau)$ the γ-discounted trajectory reward defined as $R(\tau) = \sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t})$ that, in our settings, is obtained as a linear combination of reward features. Given a policy $\pi_{\boldsymbol{\theta}}$, the expected γ-discounted return for an infinite horizon MDP is:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(s,a)\sim\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}} \big[R(s,a)\big] = \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}} \big[R(\tau)\big],$$

[2]Notice that any linear combination of the ECO-Q also satisfies the first-order optimality condition.

$\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a) = d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s)\pi_{\boldsymbol{\theta}}(a|s)$ is the γ-discounted future state-action occupancy, which represents the expected discounted number of times action a is executed in state s given μ as initial state distribution and following policy $\pi_{\boldsymbol{\theta}}$; $d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s)$ is the γ-discounted future state occupancy [110]. If $\pi_{\boldsymbol{\theta}}$ is differentiable w.r.t. the parameter $\boldsymbol{\theta}$, the gradient of the expected reward (*policy gradient*) [110, 121] is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(s,a)\sim\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}} \Big[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s,a)\Big] = \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}} \Big[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\Big]. \tag{4.5}$$

When $\pi_{\boldsymbol{\theta}}$ is an optimal policy in the class of policies $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \boldsymbol{\Theta} \subseteq \mathbb{R}^k\}$ then $\boldsymbol{\theta}$ is a *stationary point* of the expected return and thus $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{0}$ (*first-order necessary conditions for optimality* [83]).

We assume the space $\mathcal{S} \times \mathcal{A}$ is a Hilbert space [17] equipped with the weighted inner product:[3]

$$\langle f, g\rangle_{\mu,\pi_{\boldsymbol{\theta}}} = \mathop{\mathbb{E}}_{(s,a)\sim\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}} \Big[f(s,a)g(s,a)\Big] = \int_{\mathcal{S}}\int_{\mathcal{A}} f(s,a)\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a)g(s,a)\mathrm{d}s\mathrm{d}a. \tag{4.6}$$

When $\pi_{\boldsymbol{\theta}}$ is optimal for the MDP, $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$ and $Q^{\pi_{\boldsymbol{\theta}}}$ are *orthogonal* w.r.t. the inner product (4.6). We can exploit the orthogonality property to build an approximation space for the Q-function. Let $G_{\pi_{\boldsymbol{\theta}}} = \{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} \boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\}$ the subspace spanned by the gradient of the log-policy $\pi_{\boldsymbol{\theta}}$. From equation (4.5) finding an approximation space for the Q-function is equivalent to finding the orthogonal complement of the subspace $G_{\pi_{\boldsymbol{\theta}}}$, which in turn corresponds to finding the null space of the functional:

$$\mathcal{G}_{\pi_{\boldsymbol{\theta}}}[\phi] = \langle \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}, \phi\rangle_{\mu,\pi_{\boldsymbol{\theta}}}. \tag{4.7}$$

We define an Expert's COmpatible Q-feature as any function ϕ making the functional (4.7) null. This space $G_{\pi_{\boldsymbol{\theta}}}^{\perp} = \mathrm{null}(\mathcal{G}_{\pi_{\boldsymbol{\theta}}})$ represents the Hilbert subspace of the features for the Q-function that are compatible with the policy $\pi_{\boldsymbol{\theta}}$ in the sense that any Q-function optimized by policy $\pi_{\boldsymbol{\theta}}$ can be expressed as a linear combination of those features. Section 4.3.2 and 4.3.3 describe how to compute the ECO-Q from samples in finite and continuous MDPs, respectively. The dimension of $G_{\pi_{\boldsymbol{\theta}}}^{\perp}$ is typically very large since the number k of policy parameters is significantly smaller than the number of state-action pairs. A formal discussion of this issue for finite MDPs is presented in the next section.

4.3.1 Policy rank

The parametrization of the expert's policy influences the size of $G_{\pi_{\boldsymbol{\theta}}}^{\perp}$. Intuition suggests that the larger the number k of the parameters the more the policy is

[3]The inner product as defined is clearly symmetric, positive definite and linear, but there could be state-action pairs never visited, i.e., $\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a) = 0$, making $\langle f, f\rangle_{\mu,\pi_{\boldsymbol{\theta}}} = 0$ for non-zero f. To ensure the properties of the inner product, we assume to compute it only on visited state-action pairs.

informative to infer the Q-function and so the reward function. This is motivated by the following rationale. Consider representing the expert's policy using two different policy models such that one model is a superclass of the other one (for instance, assume to use linear models where the features used in the simpler model are a subset of the features used by policies in the other model). All the reward functions that make the policy gradient vanish with the rich policy model, do the same with the simpler model, while the vice versa does not hold. This suggests that complex policy models are able to reduce more the space of optimal reward function w.r.t. simpler models. This notion plays an important role for finite MDPs, i.e., MDPs where the state-action space is finite. We formalize the ability of a policy to infer the characteristics of the MDP with the concept of *policy rank*.

Definition 4.1. *Let $\pi_{\boldsymbol{\theta}}$ a policy with k parameters belonging to the class Π_Θ and differentiable in $\boldsymbol{\theta}$. The policy rank is the dimension of the space of the linear combinations of the partial derivatives of $\pi_{\boldsymbol{\theta}}$ w.r.t. $\boldsymbol{\theta}$:*

$$\mathrm{rank}(\pi_{\boldsymbol{\theta}}) = \dim(\Gamma_{\pi_{\boldsymbol{\theta}}}), \quad \Gamma_{\pi_{\boldsymbol{\theta}}} = \{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\}.$$

A first important note is that the policy rank depends not only on the policy model Π_Θ, but also on the *value* of the parameters of the policy $\pi_{\boldsymbol{\theta}}$. So the policy rank is a property of the *policy* not of the *policy model*. The following bound on the policy rank holds.

Proposition 4.1. *Given a finite MDP $\mathcal{M}$, let $\pi_{\boldsymbol{\theta}}$ a policy with k parameters belonging to the class Π_Θ and differentiable in $\boldsymbol{\theta}$, then:*

$$\mathrm{rank}(\pi_{\boldsymbol{\theta}}) \le \min\{k, |\mathcal{S}||\mathcal{A}| - |\mathcal{S}|\}.$$

Proof. $\Gamma_{\pi_{\boldsymbol{\theta}}} = \{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\}$ is a subspace of $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ generated by k vectors, thus its dimension cannot be larger than k. For each state $s \in \mathcal{S}$ it holds that $\pi_{\boldsymbol{\theta}}(\cdot|s)$ is a probability density function thus:

$$\sum_{a\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) = 1, \qquad \forall s \in \mathcal{S}.$$

For all parameters θ_j, $j = 1, ..., k$ the partial derivatives of the previous equation results in:

$$\sum_{a\in\mathcal{A}} \frac{\partial \pi_{\boldsymbol{\theta}}}{\partial \theta_j}(a|s) = 0, \qquad \forall s \in \mathcal{S}, \quad j = 1, 2, ..., k.$$

Let us consider $h \in \Gamma_{\pi_{\boldsymbol{\theta}}}$, by definition of $\Gamma_{\pi_{\boldsymbol{\theta}}}$, it can be written as a linear combination of the partial derivatives of π_θ:

$$h = \sum_{j=1}^{k} \alpha_j \frac{\partial \pi_{\boldsymbol{\theta}}}{\partial \theta_j},$$

so for all the states s:

$$\sum_{a \in A} h(a|s) = 0, \qquad \forall s \in \mathcal{S}.$$

These are $|\mathcal{S}|$ linearly independent equations, thus the dimension of $\Gamma_{\pi_{\boldsymbol{\theta}}}$ cannot be larger then $|\mathcal{S}||\mathcal{A}| - |\mathcal{S}|$. $\square$

From an intuitive point of view this is justified by the fact that $\pi_{\boldsymbol{\theta}}(\cdot|s)$ is a probability distribution. As a consequence, for all $s \in \mathcal{S}$ the probabilities $\pi_{\boldsymbol{\theta}}(a|s)$ must sum up to one, removing $|\mathcal{S}|$ degrees of freedom. This has a relevant impact on the algorithm since it induces a lower bound on the dimension of the orthogonal complement $\dim(G_{\pi_{\boldsymbol{\theta}}}^{\perp}) \geq \max\{|\mathcal{S}||\mathcal{A}| - k, |\mathcal{S}|\}$, thus even the most flexible policy (i.e., a policy model with a parameter for each state-action pair) cannot determine a unique reward function that makes the expert's policy optimal, leaving $|\mathcal{S}|$ degrees of freedom. It follows that it makes no sense to consider a policy with more than $|\mathcal{S}||\mathcal{A}| - |\mathcal{S}|$ parameters.

The bound on the policy rank does not assume any information about the optimality of the policy within the class Π_Θ. $\Gamma_{\pi_{\boldsymbol{\theta}}}$ might lose further dimensions when it is obtained as the maximum likelihood policy from the set of expert's trajectories.

4.3.2 Construction of ECO-Q in Finite MDPs

We now develop in details the algorithm to generate ECO-Q in the case of finite MDPs. From now on we will indicate with $\mathcal{D}$ the set of state-action pairs visited by the expert along the available trajectories. Moreover, we will denote with $\mathcal{S}_{\mathcal{D}} \subseteq \mathcal{S}$ and $\mathcal{A}_{\mathcal{D}} \subseteq \mathcal{A}$ the subsets of states and actions respectively visited along the available trajectories $\mathcal{D}$. When the state-action space is finite the inner product (4.6) can be written in matrix notation as:

$$\langle \mathbf{f}, \mathbf{g} \rangle_{\mu, \pi_{\boldsymbol{\theta}}} = \mathbf{f}^T \mathbf{D}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}} \mathbf{g},$$

where $\mathbf{f}$, $\mathbf{g}$ and $\boldsymbol{\delta}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}$ are real vectors with $|\mathcal{D}|$ components and $\mathbf{D}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}} = \operatorname{diag}(\boldsymbol{\delta}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}})$. The term $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$ is a $|\mathcal{D}| \times k$ real matrix, thus finding the null space of the functional (4.7) is equivalent to finding the null space of the matrix $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}^T \mathbf{D}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}$. This can be done for instance through SVD which allows to obtain a set of orthogonal basis functions $\boldsymbol{\Phi}$. Given that the weight vector $\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a)$ is usually unknown, it needs to be estimated. A simple Monte Carlo estimate exploiting the expert's demonstrations in $\mathcal{D}$ is:

$$\hat{\delta}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \gamma^t \mathbb{1}(s_{\tau_i,t} = s, a_{\tau_i,t} = a), \qquad \forall (s,a) \in \mathcal{D}. \tag{4.8}$$

However, since the policy $\pi_{\boldsymbol{\theta}}$ is known and recalling that $\delta_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s,a) = d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s) \pi_{\boldsymbol{\theta}}(a|s)$, we need to estimate just $d_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s)$:

$$\hat{d}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)} \gamma^t \mathbb{1}(s_{\tau_i,t} = s), \qquad \forall s \in \mathcal{S}_{\mathcal{D}}. \tag{4.9}$$

In Appendix B.2 we derive mean and variance of the latter estimator.

4.3.3 Construction of ECO-Q in Continuous MDPs

To extend the previous approach to the continuous domain we assume that the state-action space is equipped with the Euclidean distance. Now we can adopt an approach similar to the one exploited to extend Proto-Value Functions (PVF) [67, 68] to infinite observation spaces [69]. The problem is treated as a discrete one considering only the state-action pairs visited along the collected trajectories. A Nyström interpolation method is used to approximate the value of a feature in a non-visited state-action pair as a weighted mean of the values of the closest k features. The weight of each feature is computed by means of a Gaussian kernel placed over the Euclidean space $\mathcal{S} \times \mathcal{A}$:

$$\mathcal{K}((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) = \exp\left(-\frac{1}{2\sigma_{\mathcal{S}}^2}\|\mathbf{s}-\mathbf{s}'\|_2^2 - \frac{1}{2\sigma_{\mathcal{A}}^2}\|\mathbf{a}-\mathbf{a}'\|_2^2\right), \qquad \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}, \quad \forall \mathbf{a}, \mathbf{a}' \in \mathcal{A},$$

where $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{A}}$ are respectively the state and action bandwidth. In our setting this approach is fully equivalent to a kernel k-Nearest Neighbors regression:

$$\phi(\mathbf{s}, \mathbf{a}) = \frac{\sum_{(\mathbf{s}',\mathbf{a}') \in \mathrm{KNN}((\mathbf{s},\mathbf{a}),k,\mathcal{D}))} \mathcal{K}((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}'))\phi(\mathbf{s}', \mathbf{a}')}{\sum_{(\mathbf{s}',\mathbf{a}') \in \mathrm{KNN}((\mathbf{s},\mathbf{a}),k,\mathcal{D}))} \mathcal{K}((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}'))}, \qquad \forall \mathbf{s} \in \mathcal{S}, \quad \forall \mathbf{a} \in \mathcal{A},$$

where $\mathrm{KNN}((\mathbf{s}, \mathbf{a}), k, \mathcal{D})$ is the set of the k closest state-action pairs according to the Euclidean distance and given samples in the dataset $\mathcal{D}$. The main difference with the PVF extension is that we need to interpolate over the state-action space (not only over the state space) since we aim to recover a state-action reward function.

4.4 Expert's Compatible Reward Features

The set of ECO-Q basis functions allow representing the optimal value function under the policy $\pi_{\boldsymbol{\theta}}$. In this section, we will show how it is possible to exploit ECO-Q functions to generate basis functions for the reward representation.

We propose two approaches: the first is based on explicit application of Bellman equation and requires the knowledge of the transition model; the second is model-free and exploits optimality-invariant reward transformations. The obtained features, named *Expert's COmpatible Reward features* (ECO-R), still satisfy the first-order optimality condition.

4.4.1 Model-Based Construction of Reward Features

The relation between the reward and the Q-function is given by the Bellman equation:

$$Q^{\pi_{\boldsymbol{\theta}}}(s, a) = R(s, a) + \gamma \mathop{\mathbb{E}}_{\substack{s' \sim P(\cdot|s,a) \\ a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)}} \left[Q^{\pi_{\boldsymbol{\theta}}}(s', a')\right]. \tag{4.10}$$

Considering only the state-action pairs visited along the available expert's trajectories $\mathcal{D}$, we can express the Bellman equation in matrix form:

$$\mathbf{q}^{\pi_{\boldsymbol{\theta}}} = \mathbf{r} + \gamma \mathbf{P} \boldsymbol{\pi}_{\boldsymbol{\theta}} \mathbf{q}^{\pi_{\boldsymbol{\theta}}}. \tag{4.11}$$

Similarly to [61], we can obtain the reward function by reversing the equation $\mathbf{r} = (\mathbf{I} - \gamma \mathbf{P} \boldsymbol{\pi}_{\boldsymbol{\theta}}) \mathbf{q}^{\pi_{\boldsymbol{\theta}}}$. Thus the set of ECO-R $\{\psi_i\}_{i=1}^{p}$ are obtained by means of the linear mapping of the ECO-Q $\{\phi_i\}_{i=1}^{p}$:

$$\boldsymbol{\Psi} = (\mathbf{I} - \gamma \mathbf{P} \boldsymbol{\pi}_{\boldsymbol{\theta}}) \boldsymbol{\Phi}. \tag{4.12}$$

This transformation however requires the knowledge of the transition model $\mathcal{P}$ which in most of the real cases cannot be assumed. As shown in [61], in order to exploit only expert's demonstrations it is necessary to resort to heuristics that may be hard to compute in practice or to additional samples for exploration of the system dynamics.

4.4.2 Model-Free Construction of Reward Features

Reversing the Bellman equation allows finding the reward space that generates the estimated Q-function. However, IRL is interested in finding just a reward space under which the expert's policy is optimal. This problem can be seen as an instance of reward shaping [81] where the authors show that the space of all the reward functions sharing the same optimal policy is given by:

$$R'(s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[\chi(s')\right] - \chi(s),$$

where $\chi(s)$ is a state-dependent potential function. A smart choice [81] is to set $\chi = V^{\pi_{\boldsymbol{\theta}}}$ under which the new reward space is given by the advantage function: $R'(s,a) = Q^{\pi_{\boldsymbol{\theta}}}(s,a) - V^{\pi_{\boldsymbol{\theta}}}(s) = A^{\pi_{\boldsymbol{\theta}}}(s,a)$. Thus the expert's advantage function is an admissible reward optimized by the expert's policy itself. This choice is, of course, related to using $Q^{\pi_{\boldsymbol{\theta}}}$ as reward. However, the advantage function encodes a more local and more transferable information w.r.t. the Q-function. In general, the advantage function can be expressed in matrix form as:

$$\mathbf{a}^{\pi_{\boldsymbol{\theta}}} = \mathbf{q}^{\pi_{\boldsymbol{\theta}}} - \tilde{\mathbf{v}}^{\pi_{\boldsymbol{\theta}}} = \big(\mathbf{I} - \tilde{\boldsymbol{\pi}}_{\boldsymbol{\theta}}\big) \mathbf{q}^{\pi_{\boldsymbol{\theta}}}, \tag{4.13}$$

where $\tilde{\mathbf{v}}^{\pi_{\boldsymbol{\theta}}}$ is a $|\mathcal{S}||\mathcal{A}|$-dimensional vector obtained from the $|\mathcal{S}|$-dimensional vector $\mathbf{v}^{\pi_{\boldsymbol{\theta}}}$ repeating each component $|\mathcal{A}|$ times. Analogously $\tilde{\boldsymbol{\pi}}_{\boldsymbol{\theta}}$ is a $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|$ matrix obtained from the $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ matrix $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ repeating each row $|\mathcal{A}|$ times. This holds for the full state-action space. When the expert visited a limited number of state-action pairs, the transformation requires repeating only the rows of matrix $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ associated to visited states for a number of times equal to the number of distinct actions performed by the expert in that state. Thus, the space of reward features can be recovered through matrix equality:

$$\boldsymbol{\Psi} = (\mathbf{I} - \tilde{\boldsymbol{\pi}}_{\boldsymbol{\theta}}) \boldsymbol{\Phi}. \tag{4.14}$$

Notice that this is a simple linear transformation through the expert's policy. However, the transformation does not preserve the orthogonality of features $\mathbf{\Phi}$, thus it is necessary to apply SVD again to get an orthogonal basis of the recovered approximation space. Furthermore, since $\mathbf{I} - \tilde{\boldsymbol{\pi}}_{\boldsymbol{\theta}}$ is not full-rank, therefore the recovered space has a smaller dimensionality. The specific choice of the state-potential function has the advantage to improve the learning capabilities of any RL algorithm [81]. This is not the only choice of the potential function possible, but it has the advantage of allowing model-free estimation.

Once the ECO-R basis functions have been generated, they can be used to feed any IRL algorithm that represents the expert's reward through a linear combination of basis functions. In the next section, we propose a new method based on the optimization of a second-order criterion that favors reward functions that significantly penalize deviations from the expert's policy.

4.5 Reward Selection via Second-Order Criterion

Any linear combination of the ECO-R $\{\psi_i\}_{i=1}^{p}$ makes the gradient vanish, however in general this is not sufficient to ensure that the policy parameter $\boldsymbol{\theta}$ is a maximum of $J(\boldsymbol{\theta})$. Combinations that lead to minima or saddle points should be discarded. Furthermore, provided that a subset of ECO-R leading to maxima has been selected, we should identify a single reward function in the space spanned by this subset of features. Both these requirements can be enforced by imposing a second-order optimality criterion based on the policy Hessian that is given by [54, 38]:

$$\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) = \mathbb{E}_{\tau}\left[\Big(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)^T + \mathcal{H}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)\Big) R(\tau, \boldsymbol{\omega})\right],$$

where $\boldsymbol{\omega}$ is the reward weight and the reward is obtained as a linear combination of the ECO-R:

$$R(\tau, \boldsymbol{\omega}) = \sum_{i=1}^{p} \omega_i \sum_{t=0}^{T(\tau)} \gamma^t \psi_i(s_{\tau,t}, a_{\tau,t}). \tag{4.15}$$

It is worth to point out that the linear parametrization, in this case, does not represent a limitation of the algorithm. Indeed, we defined the approximation space of the reward function as the Hilbert space whose functions make the policy gradient vanish. This space will contain *all* the reward functions optimized by the expert's policy and each function in that space can be expressed as a linear combination of the elements in the basis, like ECO-R.

4.5.1 Maximum eigenvalue optimality criterion

To retain only maxima we need to impose that the Hessian is negative definite which is equivalent to require that the maximum eigenvalue $\lambda_{\max}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}))$ is negative. The requirement $\lambda_{\max}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega})) < 0$ is only a constraint, in order to single out a reward function we need an optimality criterion over the Hessian. We aim to find the reward function that best represents the optimal policy parametrization in the sense that even a slight alteration of the policy parameters induces a dramatic degradation of the performance. Geometrically this corresponds to finding the reward function for which the expected return locally represents the sharpest hyper-paraboloid. Analytically this condition can be enforced by minimizing the maximum eigenvalue of the Hessian whose eigenvector corresponds to the direction of minimum curvature (*maximum eigenvalue optimality criterion*, ME-opt). The problem can be formalized using Semi-Definite Programming (SDP):

$$\begin{aligned} \underset{\boldsymbol{\omega}\in\mathbb{R}^p}{\text{minimize}} \quad & \lambda_{\max}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega})) \\ \text{subject to} \quad & \mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega}) + \epsilon\mathbf{I} \preceq 0 \\ & \|\boldsymbol{\omega}\|_2 \le 1, \end{aligned}$$

where $\epsilon \geq 0$ is a threshold to ensure the Hessian is (strictly) negative definite. The constraint $\|\boldsymbol{\omega}\|_2 \le 1$ on the reward parameters is necessary to ensure that the problem is bounded, otherwise, if the problem is feasible, the maximum eigenvalue can be made arbitrarily negative. Clearly the constraint on the weights will be satisfied with equality, i.e., $\|\boldsymbol{\omega}\|_2 = 1$, when the problem admits a solution.

In our framework, the reward function can be expressed as a linear combination of the ECO-R so we can rewrite the Hessian as:

$$\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega}) = \sum_{i=1}^{p} \omega_i \mathcal{H}_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}), \tag{4.16}$$

where $J_i(\boldsymbol{\theta})$ is the expected return considering as reward function ψ_i.

Being the Hessian symmetric and defined through an affine function of the variables $\boldsymbol{\omega}$, the maximum eigenvalue is convex [85]. This is due to the fact that we can define $\lambda_{\max}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega})) = \max_{\|\mathbf{x}\|_2\le 1} \mathbf{x}^T \mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega})\mathbf{x}$, which is the maximum of an affine function in $\boldsymbol{\omega}$ over a convex set $\|\mathbf{x}\|_2 \le 1$. Anyway, we can rephrase the problem by removing the $\lambda_{\max}$ function:

$$\begin{aligned} \underset{\boldsymbol{\omega}\in\mathbb{R}^p}{\text{minimize}} \quad & t \\ \text{subject to} \quad & \mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta},\boldsymbol{\omega}) - t\mathbf{I} \preceq 0 \\ & t \le -\epsilon \\ & \|\boldsymbol{\omega}\|_2 \le 1. \end{aligned}$$

This optimization problem is for sure feasible (for sufficiently small ϵ) since, as already seen, the true reward function and the advantage function make the expert's policy parametrization optimal. However, in most of the cases, it is impractical to solve, at least for two reasons. First, the computational effort is enormous even for few policy parameters. Second, it might be the case that the strict negative definiteness constraint is never satisfied due to blocked-to-zero eigenvalues. This problem arises quite often and is related to the presence of "useless" policy parameters that even if modified do not affect the policy performance (for instance, a subset of parameters which are linearly dependent). In those cases $\lambda_{\max}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}))$ would be zero no matter which reward weights $\boldsymbol{\omega}$ are selected.

4.5.2 Trace optimality criterion

In order to account for the possible presence of blocked-to-zero eigenvalues we need to consider all the eigenvalues of the Hessian matrix instead of the maximum one only. When in the direction of minimum curvature the curve is flat (like in case of blocked-to-zero eigenvalues) we aim to maximize an index of the overall concavity. The trace of the Hessian, being the sum of the eigenvalues, can be used for this purpose. Minimizing the trace of the Hessian (*trace optimality criterion*, TR-opt) can be formulated in the following semi-definite programming problem:

$$\begin{aligned} \underset{\boldsymbol{\omega} \in \mathbb{R}^p}{\text{minimize}} \quad & \operatorname{tr}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega})) \\ \text{subject to} \quad & \mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) + \epsilon \mathbf{I} \preceq 0 \\ & \|\boldsymbol{\omega}\|_2 \leq 1 \end{aligned} \tag{4.17}$$

The trace is a linear function, i.e., $\operatorname{tr}(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega})) = \sum_{i=1}^{p} \omega_i \operatorname{tr}(\mathcal{H}_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}))$, thus the objective function is for sure convex. Like for the maximum eigenvalue optimality criterion a normalization constraint on the reward parameters is necessary to ensure boundedness. The negative definiteness constraint becomes substantial to ensure the optimality, since there might be ECO-R with negative trace but positive eigenvalues. Moreover, in presence of blocked-to-zero eigenvalues ϵ should be set to zero to ensure feasibility. This formulation, although less demanding w.r.t. the previous one, still displays performance degradation as the number of basis functions increases due to the negative definiteness constraint.

4.5.3 Trace heuristic criterion

Solving the semidefinite programming problem of one of the presented optimality criteria is unfeasible for almost all the real world problems. We are interested in formulating a non-SDP problem, which is a surrogate of the trace optimality criterion, that can be solved more efficiently (*trace heuristic criterion*, TR-heu). Essentially, we need to get rid of the semi-definite constraint.

We assume that the ECO-R are orthonormal in order to compare them.[4] The main challenge is how to select the weight $\boldsymbol{\omega}$ in order to get a (near-)optimal trace minimizer that preserves the negative semidefinite constraint.

Given two symmetric matrices A and B, it holds from Weyl's inequality that $\lambda_{\max}(A) + \lambda_{\max}(B) \leq \lambda_{\max}(A + B)$, thus any linear combination of semidefinite ma-

[4]A normalization condition is necessary since the magnitude of the trace of a matrix can be arbitrarily changed by multiplying the matrix by a constant.

trices with positive coefficients for the negative semidefinite ones and negative coefficients for the positive semidefinite ones is guaranteed to yield a negative semidefinite matrix. In principle, nothing can be ensured for the indefinite matrices, without looking to the magnitude of the eigenvalues. Therefore, we get a feasible solution by retaining only the ECO-R yielding a semidefinite Hessian and switching sign to those with positive semidefinite Hessian. Notice that, in this way, we can lose the optimal solution since the trace minimizer might assign a non-zero weight to a ECO-R with indefinite Hessian. For brevity, we will indicate with $tr_i = \mathrm{tr}(\mathcal{H}_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}))$ and $\mathbf{tr}$ the vector whose components are tr_i, $i = 1, 2, ..., p$. SDP is no longer needed:

$$\begin{aligned} \underset{\boldsymbol{\omega} \in \mathbb{R}^p}{\text{minimize}} \quad & \boldsymbol{\omega}^T \mathbf{tr} \\ \text{subject to} \quad & \|\boldsymbol{\omega}\|_2 = 1. \end{aligned} \tag{4.18}$$

The constraint $\|\boldsymbol{\omega}\|_2 = 1$ ensures that, when the ECO-R are orthonormal, the resulting ECO-R has Euclidean norm one. This is a convex programming problem with linear objective function and quadratic constraint, the closed form solution can be found with Lagrange multipliers:

$$\omega_i = -\frac{tr_i}{\|\mathbf{tr}\|_2}, \qquad i = 1, 2, ..., p. \tag{4.19}$$

Proof. We assume to consider a set of ECO-R, yielding negative semidefinite Hessian. The Lagragian function is:

$$\mathcal{L}(\boldsymbol{\omega}, \alpha) = \sum_{i=1}^{p} \omega_i tr_i + \alpha \bigg(\sum_{i=1}^{p} \omega_i^2 - 1 \bigg),$$

where α is the Lagrange multiplier. We impose that partial derivatives of $\mathcal{L}$ w.r.t. ω_j and α vanish, thus:

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \omega_j} = tr_j + 2\alpha\omega_j = 0, \qquad j = 1, 2, ..., p, \\ &\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{i=1}^{p} \omega_i^2 - 1 = 0. \end{aligned}$$

From the first equation, for $\alpha \neq 0$, we obtain an expression of ω_j as a function of α:

$$\omega_j = -\frac{tr_j}{2\alpha}, \qquad j = 1, 2, ..., p.$$

The case $\alpha = 0$ is uninteresting since it is feasible only if all traces are null. By substitution we obtain an expression for α:

$$\alpha = \pm \frac{1}{2} \sqrt{\sum_{i=1}^{p} tr_i^2} = \pm \frac{1}{2} \|\mathbf{tr}\|_2.$$

Since we are looking for a minimum, provided that all traces are non positive, the objective function is minimized for non negative weights:

$$\omega_j = -\frac{tr_j}{\sqrt{\sum_{i=1}^{p} tr_i^2}} = -\frac{tr_j}{\|\mathbf{tr}\|_2}, \qquad j = 1, 2, ..., p.$$

$\square$

Algorithm 4.1 CR-IRL with trace heuristic.

Input: $\mathcal{D} = \left\{ (s_{\tau_i,0}, a_{\tau_i,0}, \dots, s_{\tau_i,T(\tau_i)}, a_{\tau_i,T(\tau_i)}) \right\}_{i=1}^{N}$ a set of expert's trajectories collected with the exptert's policy π^E and a parametric representation of the expert's policy $\pi_{\boldsymbol{\theta}}$

Output: trace heuristic solution $R^{\text{TR-heu}}$

```
1: Estimate $\delta^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s,a) = d^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s)\pi_{\boldsymbol{\theta}}(a|s)$ for the visited state-action pairs using equation (4.9):
   $$\hat{d}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T(\tau_i)} \gamma^t \mathbb{1}(s_{\tau_i,t} = s), \qquad \forall s \in \mathcal{S}_{\mathcal{D}}.$$
2: Collect $\delta^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s,a)$ in the $|\mathcal{D}| \times |\mathcal{D}|$ diagonal matrix $\mathbf{D}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}$ and $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a)$ in the $|\mathcal{D}| \times k$ matrix $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$
3: Get the set of ECO-Q by computing the null space of matrix $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}{}^T \mathbf{D}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}$ through SVD:
   $$\boldsymbol{\Phi} = \text{null}\big(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}{}^T \mathbf{D}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}\big)$$
4: Get the set of ECO-R by applying reward shaping to the set of ECO-Q:
   $$\boldsymbol{\Psi} = (\mathbf{I} - \tilde{\boldsymbol{\pi}}_{\boldsymbol{\theta}})\boldsymbol{\Phi}$$
5: Apply SVD to orthogonalize $\boldsymbol{\Psi}$
6: Estimate the policy Hessian for each ECO-R $\boldsymbol{\psi}_i$, $i = 1, 2, ..., p$ using the REINFORCE-like estimator with optimal baseline:
   $$\hat{\mathcal{H}}_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}) = \frac{1}{N}\sum_{j=1}^{N}\Big(\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau_j)\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau_j)^T + \mathcal{H}_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau_j)\Big)\Big(\psi_i(\tau_j) - b\Big)$$
7: Discard the ECO-R having indefinite Hessian, switch sign for those having positive semidefinite Hessian, compute the traces of each Hessian and collect them in the vector $\mathbf{tr}$
8: Compute the trace heuristic ECO-R as:
   $$R^{\text{TR-heu}} = \boldsymbol{\Psi}\boldsymbol{\omega}, \qquad \boldsymbol{\omega} = -\frac{\mathbf{tr}}{\|\mathbf{tr}\|_2}$$
9: *(Optional)* Apply penalization to unexplored state-action pairs
```

4.5.4 Multi-Objective interpretation of the second-order criteria

The intuitive idea behind the second-order criteria we presented in the previous sections consists in preferring rewards that penalize the most deviations from the expert's policy. This notion can be formalized, in the general case, as the multi-objective SDP problem of minimizing the vector of eigenvalues of the policy Hessian:

$$\begin{aligned} \underset{\boldsymbol{\omega} \in \mathbb{R}^p}{\text{minimize}} \quad & \boldsymbol{\lambda}(\boldsymbol{\omega}) = \big(\lambda_1(\boldsymbol{\omega}), \lambda_2(\boldsymbol{\omega}), ..., \lambda_k(\boldsymbol{\omega})\big) \\ \text{subject to} \quad & \mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) + \epsilon \mathbf{I} \preceq 0, \\ & \|\boldsymbol{\omega}\|_2 = 1, \end{aligned}$$

where $\lambda_i(\boldsymbol{\omega}) = \lambda_i(\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}))$ for $i = 1, 2, ..., k$ is the i-th largest eigenvalue of the policy Hessian. Clearly, among all possible feasible solutions we seek for the (strict) Pareto optimal ones $\boldsymbol{\omega}^P$, i.e., those for which there exist no feasible weight vector $\boldsymbol{\omega}$ such that $\lambda_i(\boldsymbol{\omega}) \leq \lambda_i(\boldsymbol{\omega}^P)$, $i = 1, 2, ..., k$ with at least one strict inequality. It is well-known that the Pareto frontier (the set of all the Pareto optimal solutions) cannot be computed efficiently in most of the cases. A standard approach to tackle this problem is *scalarization* [49]. Scalarization consists in formulating a single-objective optimization problem such that the optimal solutions to the single-objective problem are Pareto optimal solutions to the multi-objective problem and vice versa. A common choice is linear scalarization, that consists in combining the multiple objectives via a linear function:

$$L(\boldsymbol{\lambda}(\boldsymbol{\omega}), \gamma) = \sum_{i=1}^{k} \gamma_i \lambda_i(\boldsymbol{\omega}) = \boldsymbol{\gamma}^T \boldsymbol{\lambda}(\boldsymbol{\omega}). \tag{4.20}$$

It can be proved that the minimizer of the scalar objective $L(\boldsymbol{\lambda}(\boldsymbol{\omega}), \boldsymbol{\gamma})$ is a Pareto optimal solution for every value of $\boldsymbol{\gamma} \geq 0$. However, this scalarization is guaranteed to yield all Pareto optimal solutions only when the Pareto frontier is convex, which is not the our case as, for instance, $\lambda_k(\boldsymbol{\omega})$ is concave.

It is worth noting that the maximum eigenvalue and the trace optimality criteria fall in this formulation. The former is obtained by setting $\gamma_1 = 1$ and $\gamma_i = 0$, $i = 2, 3, ..., k$, while the latter is obtained by setting $\gamma_i = 1$, $i = 1, 2, ..., k$. Therefore, we are sure that those criteria provide Pareto optimal solutions. Furthermore, the two criteria correspond to popular solution concepts in multi-agent decision theory [116]. The maximum eigenvalue optimality criterion recovers the egalitarian social welfere solution, i.e., the solution that minimizes the maximum unsatisfaction, whereas the trace optimality criterion seeks for the utilitarian solution, i.e., the solution that maximizes the sum of profits. The trace heuristic, however, does not guarantee that the recovered solution is Pareto optimal.

Furthermore, the single-objective function (4.20) is not convex for all values of $\boldsymbol{\gamma}$. It can be proved that a sufficient condition for convexity is that $\gamma_1 \geq \gamma_2 \geq ... \geq \gamma_k \geq 0$ [74], to which both maximum eigenvalue and trace optimality criteria comply.

CR-IRL does not assume to know the state space $\mathcal{S}$ and the action space $\mathcal{A}$, thus the recovered reward is defined only in the state-action pairs visited by the expert along the trajectories in $\mathcal{D}$. When the state and action spaces are known, we can complete the reward function also for unexplored state-action pairs assigning a penalized reward (e.g., a large negative value), otherwise the penalization can be performed online when the recovered reward is used to solve the forward RL problem.

Refer to Alg. 4.1 for a complete overview of CR-IRL.

4.6 Computational analysis

The main sources of computational complexity in CR-IRL are the SVD, the computation of the Hessian and its eigenvalue computation for each ECO-R. For a $m \times n$ matrix it is well known that SVD has complexity $O(\min\{mn^2, nm^2\})$. We apply SVD twice: the first time it is applied to matrix $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}{}^T D_{\mu}^{\pi_{\boldsymbol{\theta}}}$ of dimension $k \times |\mathcal{D}|$ and the second time to matrix Ψ of dimension at most $|\mathcal{D}| \times |\mathcal{D}|$. Assuming $k \leq |\mathcal{D}|$, the second application dominates yielding a complexity of $O(|\mathcal{D}|^3)$. The computation of the Hessian is linear in the number of ECO-R (at maximum $|\mathcal{D}|$) and in the number of samples ($|\mathcal{D}|$) and quadratic in the number of parameters k, thus the complexity is $O(k^2|\mathcal{D}|^2)$. Finally the computation of the eigenvalues is cubic in the order of the matrix (k) and has to be done for every ECO-R, yielding a cost of $O(k^3|\mathcal{D}|)$, which is dominated by $O(k^2|\mathcal{D}|^2)$ as $k \leq |\mathcal{D}|$. Overall the complexity of CR-IRL is $O(\max\{k^2|\mathcal{D}|^2, |\mathcal{D}|^3\})$.

Chapter 5

Experiments

This chapter is devoted to the experimental evaluation of CR-IRL on classic benchmark problems, both discrete and continuous: the Taxi problem (Section 5.2), the Linear Quadratic Gaussian Regulator (Section 5.3) and the Car on the Hill environment (Section 5.4).

We compare CR-IRL with classic IRL algorithms in order to evaluate the performance of the recovered reward functions. In particular the experiments are intended to evaluate both the phases of CR-IRL: the feature construction and the reward recovery. For the feature construction phase, we compare the ECO-R with predetermined set of features automatically generated (Proto-Value Functions); while for the reward recovery phase we evaluate CR-IRL against popular state-of-the-art IRL algorithms able to output a reward as linear combination of features when fed with ECO-R.

We start presenting the evaluation metrics we adopt to compare the considered algorithms (Section 5.1) and then we report the full results of the experiments.

5.1 Evaluation metrics

Defining a metric to evaluate the intuitive notion of "good reward" is a hard task and literature has not formalized it yet. Intuition suggests that we aim to recover a reward function that allows the agent to learn a policy as close as possible to the expert's policy. However, there are problems in which even if the learned policy and the expert's policy are significantly different the average return is very close, and vice versa. In other words, a small policy deviation might attain a large return deviation. Furthermore, all other things being equal, we should prefer a reward function with faster learning curve. Clearly, this last property is algorithm-dependent. For these reasons we resort to multiple metrics:

- *learning speed*: the number of iterations of the forward RL algorithm needed for converging to the optimal policy;

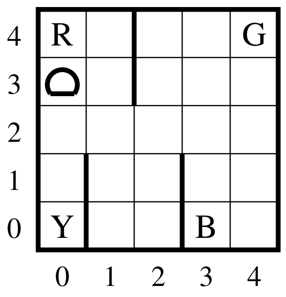

Figure 5.1: The Taxi Domain (from [29]).

- *average return*: the average return $\hat{J}$ of the policy learned with the recovered reward function, compared with expert's average return J^E;
- *policy distance*: the distance between the expert's policy π^E and the learned policy $\hat{\pi}$ (Kullback-Leibler divergence);
- *parameter distance*: when both the expert's policy and the learned policy are parametric policies of the same class, the distance (in norm) between the expert's parameters $\boldsymbol{\theta}^E$ and the learned parameters $\hat{\boldsymbol{\theta}}$.[1]

5.2 Taxi

This section is devoted to the experiments performed on the Taxi domain, defined in [29]. The environment corresponds to a 5x5 grid in which there are 4 locations, labeled by different letters (R, G, B and Y). The job of the taxi driver is to pick up the passenger at one location and drop him off in another. You receive +20 points for a successful dropoff and lose 1 point for every timestep it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions. The available actions are the movements in the four directions (North, East, South, West), pick-up and drop-off (all actions are deterministic). The passenger position, the destination position and the initial position of the taxi represent the state. The distribution of the initial state is uniform and the discount factor is 0.99 (Figure 5.1).

We compute the optimal deterministic policy via value iteration [13] and we use it to fit via maximum likelihood the parameters of expert's policy (5.1) by minimizing the KL-divergence between the deterministic policy and the expert's policy. We consider the class of ϵ-Boltzmann policy with fixed ϵ:[2]

$$\pi_{\boldsymbol{\theta},\epsilon}(a|s) = (1-\epsilon)\frac{e^{\boldsymbol{\theta}_a^T \boldsymbol{\zeta}_s}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}_{a'}^T \boldsymbol{\zeta}_s}} + \frac{\epsilon}{|\mathcal{A}|}, \tag{5.1}$$

[1]Parameter distance is not always a good metric, in particular in presence of parameters that do not affect the policy performance. We will use it only for the case of the LQG.

[2]We made the choice to use ϵ-Boltzmann instead of Boltzmann policy since we can leverage on ϵ to control the exploration-exploitation trade off.

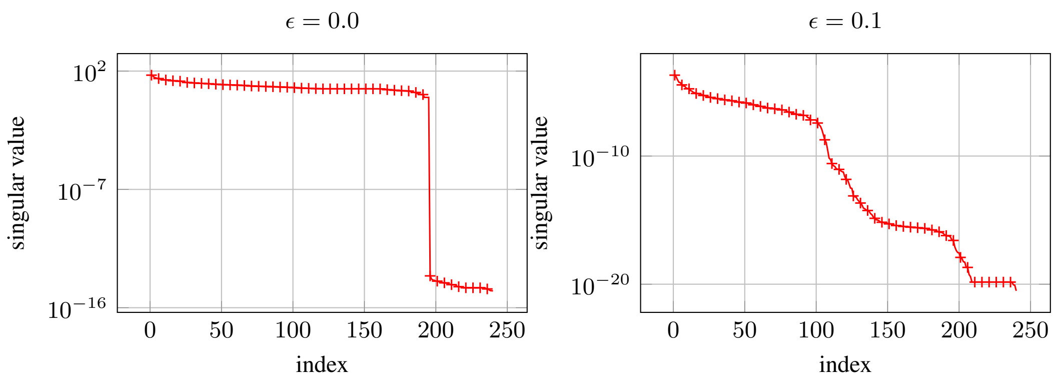

Figure 5.2: Singular values of the $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta},\epsilon}$. When $\epsilon = 0$ the gap between non-zero and zero singular values is clear; while the behavior is smoother when $\epsilon = 0.1$ but the magnitude of the singular values is significantly smaller.

where the policy features $\boldsymbol{\zeta}_s$ are the following state features: current location, passenger location, destination location, whether the passenger has already been picked up. In order to test the robustness of CR-IRL to imperfect experts we introduce a noise ϵ in the expert's policy (also named exploration level in the followings). We consider the cases $\epsilon = 0$ and $\epsilon = 0.1$. The resulting policy has 240 parameters, however the policy rank is smaller (195 for $\epsilon = 0$ and 185 for $\epsilon = 0.1$) due to the presence of parameters that do not affect the policy performance (Figure 5.2). The derivation of the gradient and the Hessian of policy (5.1) are reported in Appendix B.3.

We compare the performance of CR-IRL with trace heuristics criterion,[3] against Behavioral Cloning (BC), obtained by recovering the maximum likelihood Boltzmann policy from expert's trajectories, Maximum Entropy IRL (ME-IRL) [125] and Linear Programming Apprenticeship Learning (LPAL) [112].

In order to recover the set of ECO-R we collect 100 expert's trajectories. For this experiment we evaluate both model-free and model-based versions of CR-IRL. For the latter case the transition model is estimated from expert's trajectories via the Monte Carlo estimator:

$$\hat{P}(s'|s,a) = \frac{\sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \mathbb{1}\left(s_{\tau_i,t+1} = s', a_{\tau_i,t} = a, s_{\tau_i,t} = s\right)}{\sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)-1} \mathbb{1}\left(a_{\tau_i,t} = a, s_{\tau_i,t} = s\right)}. \tag{5.2}$$

Implementation issues

In the Taxi domain when the agent reaches the terminal state it will receives zero reward forever. However, the reward function recovered by CR-IRL could assign non-zero reward to the terminal state. For implementation reasons, we translate the

[3]The maximum eigenvalue optimality criterion is not suitable for this case due to the presence of blocked-to-zero eigenvalues.

recovered reward in order to ensure zero-reward to the terminal state. Furthermore, the recovered reward function is extended to the unvisited state-action pairs assigning the minimum value among the rewards of the visited state-action pairs.

5.2.1 Learning speed experiments

This set of experiments is meant to compare the learning speed of the reward functions recovered by the considered IRL methods when a Boltzmann policy ($\epsilon = 0$) is trained with REINFORCE [121]. The training is performed with Adam [58] (learning rate 0.008, other parameters with default value), at each iteration 100 trajectories are used to estimate the gradient, the maximum number of iterations is 1000 and the initial parameter is $\boldsymbol{\theta}^{(0)} = \mathbf{0}$ so that the initial policy $\pi_{\boldsymbol{\theta},0}^{(0)}$ is the random policy. All data are averaged over 40 runs and the error bars correspond to the 95% confidence intervals.

Figure 5.3 shows that model-free CR-IRL, with 100 expert's trajectories, outperforms the original reward function in terms of convergence speed regardless the exploration level. Behavioral Cloning (BC), obtained by recovering the maximum likelihood Boltzmann policy from expert's trajectories, is very susceptible to noise, as expected.

We compare also the second-order criterion of CR-IRL to single out the reward function with Maximum Entropy IRL (ME-IRL) [125] and Linear Programming Apprenticeship Learning (LPAL) [112] using as reward features the set of model-free ECO-R. We can see in Figure 5.3 that ME-IRL does not perform well when $\epsilon = 0$, since the transition model is badly estimated. The convergence speed remains very slow also for $\epsilon = 0.1$, since ME-IRL does not guarantee that the recovered reward is a maximum of J. LPAL provides as output an apprenticeship policy (not a reward function) and, like BC, is very sensitive to noise and to the quality of the estimated transition model.

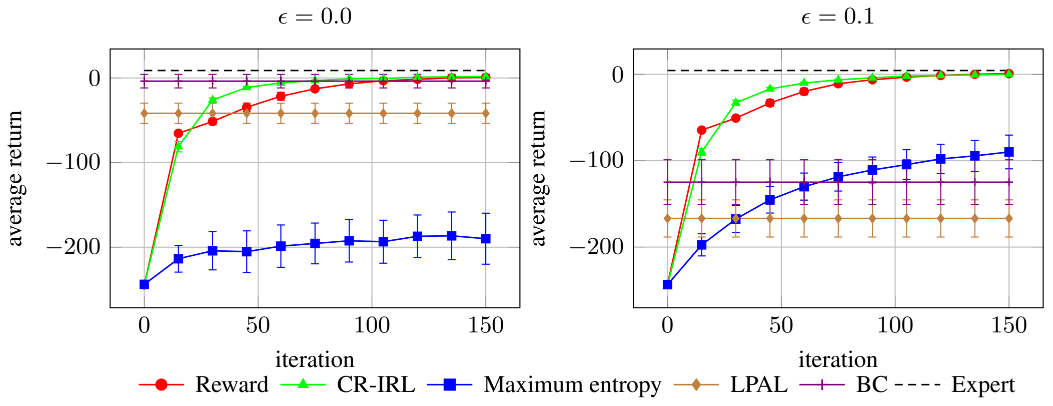

Figure 5.3: Average return of the Taxi problem as a function of the number of iterations of REINFORCE for model-free ECO-R.

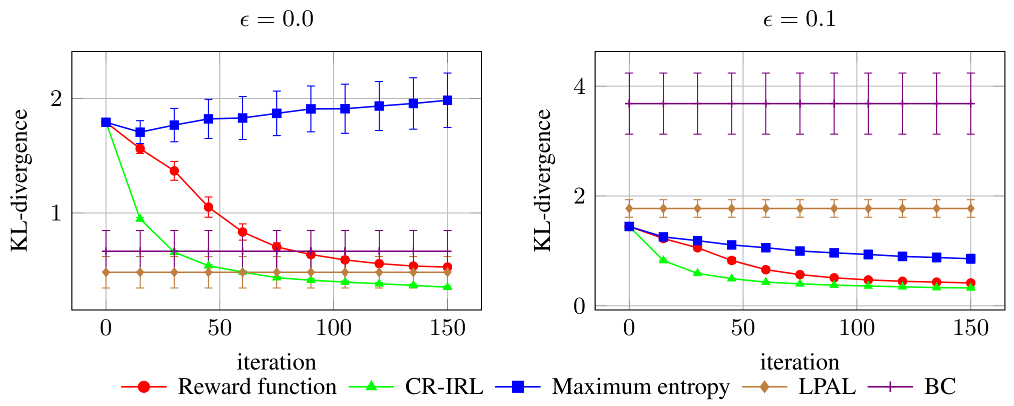

Figure 5.4: Kullback-Leibler divergence between expert's policy and learned policy in the Taxi problem as a function of the number of iterations of REINFORCE for model-free ECO-R.

Besides the average return, we compare the recovered reward functions in terms of distance between the expert's policy and the learned policy (Kullback-Leibler divergence) estimated with (4.3). Figure 5.4 confirms the faster convergence of CR-IRL over the original reward and ME-IRL. We observe that LPAL is able to recover a policy that is more similar to the expert's policy w.r.t. that of BC, however the average return is worse (this is a consequence of the fact that LPAL does not learn well to perform pick-up and drop-off actions that are associated with the largest rewards).

We also test the model-based version of CR-IRL, where the transition model is estimated with (5.2). In Figure 5.5 and Figure 5.6 we notice that the usage of model-based ECO-R instead of model-free ECO-R has no relevant impact on CR-IRL and LPAL, while ME-IRL benefits from the model-based features only when the expert is not noisy.

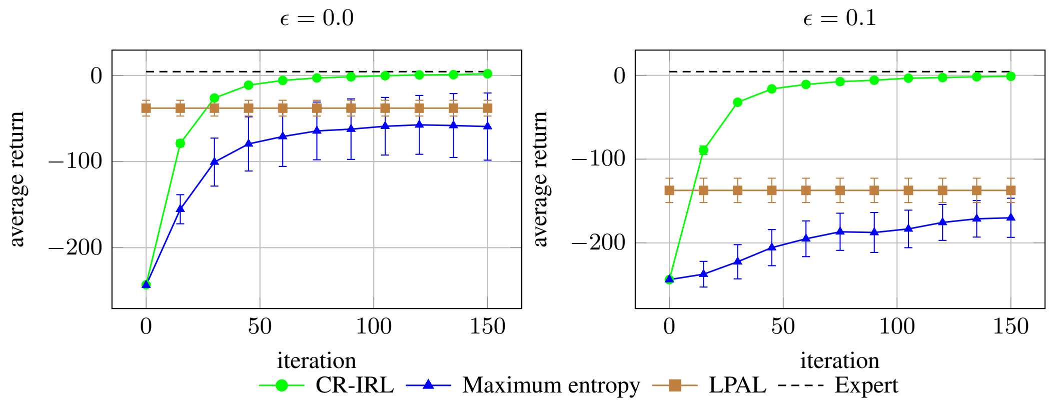

Figure 5.5: Average return of the Taxi problem as a function of the number of iterations of REINFORCE for reward functions recovered from model-based ECO-Rs.

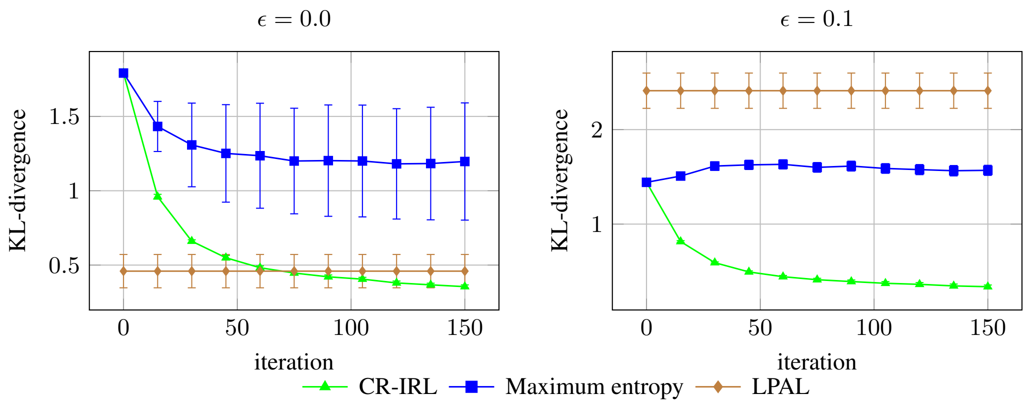

Figure 5.6: Kullback-Leibler divergence between expert's policy and learned policy in the Taxi problem as a function of the number of iterations of REINFORCE for model-based ECO-R.

The previous experiments demonstrate that the set of ECO-R constructed by CR-IRL when coped with the trace heuristic criterion allows to recover policies that outperform popular IRL methods and display a faster learning curve w.r.t. the original reward function. The fact that ECO-R do not perform well with traditional IRL methods is simply explained. ECO-R are built exploiting a first-order optimality condition on the policy gradient with no need of a preselection of an approximation space; traditional IRL methods are designed to deal with handcrafted features that, in some intuitive sense, represent the underlying dynamics of the problem.

The trace heuristic can be also used when a given set of features (different from ECO-R) is provided. Since the usage of the Hessian makes sense only when all the considered features are a stationary point of the policy gradient we first need to remove the orthogonal projections over the space spanned by the gradient log-policy. We compare (Figure 5.7) the learning performance of the set of the first 100 Proto-Value Functions (PVF) [67] when linearly combined with ME-IRL, LPAL and our trace heuristic. We notice that LPAL outperforms both the maximum entropy and the Hessian approaches regardless of the exploration level. Trace heuristic is slightly more effective w.r.t. maximum entropy when the expert is not noisy. The poor performance of trace heuristic with predefined approximation space is due to the fact that such space might not be sufficiently expressive to represent a sharp maximum of J.

Overall, we notice that the two phases of CR-IRL (feature construction and reward recovery) are not fully independent because both share a strong relation to optimality conditions on the policy gradient and Hessian. In all scenarios we considered the best performance is obtained with the combination of the two phases.

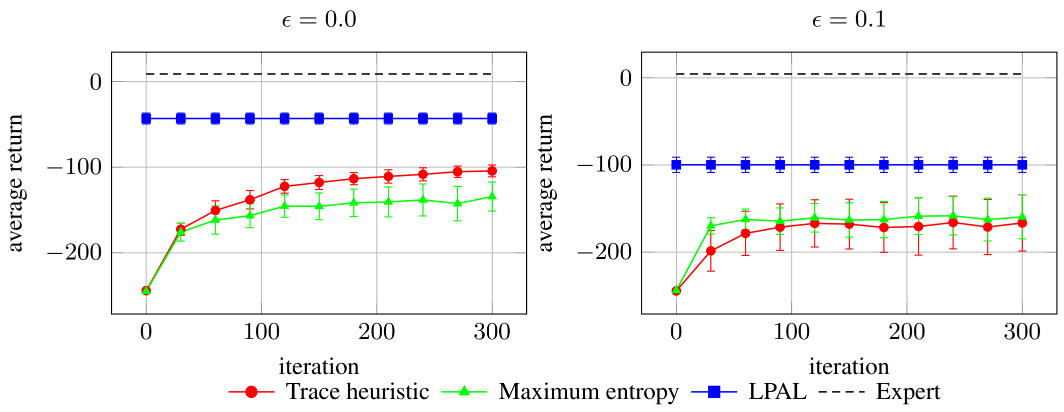

Figure 5.7: Average return of the Taxi problem as a function of the number of iterations of REINFORCE with the reward function obtained from PVFs.

5.2.2 Sensitivity to the number of expert's trajectories

The last set of experiments is aimed to evaluate the effect of the number of expert's trajectories on the average return of the recovered reward functions (Figure 5.8 and Figure 5.9). The experiment is performed with model-free ECO-R. We notice that CR-IRL is susceptible to the number of expert's trajectories only when $\epsilon = 0.1$: the expert demonstrates a suboptimal behavior and this is more likely when the trajectories are many. This reflects on the estimation of the reward function that does not penalize suboptimal actions performed by the expert. LPAL shows the expected behavior, improving the average return as the number of trajectories increases. Also BC improves overall with the number of trajectories, more effectively when the expert is not noisy.

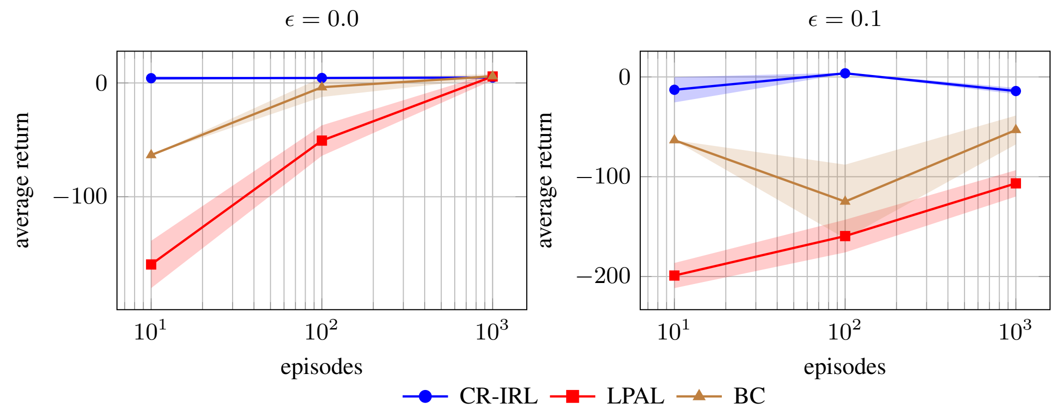

Figure 5.8: Average return of the Taxi problem as a function of the number of expert's trajectories.

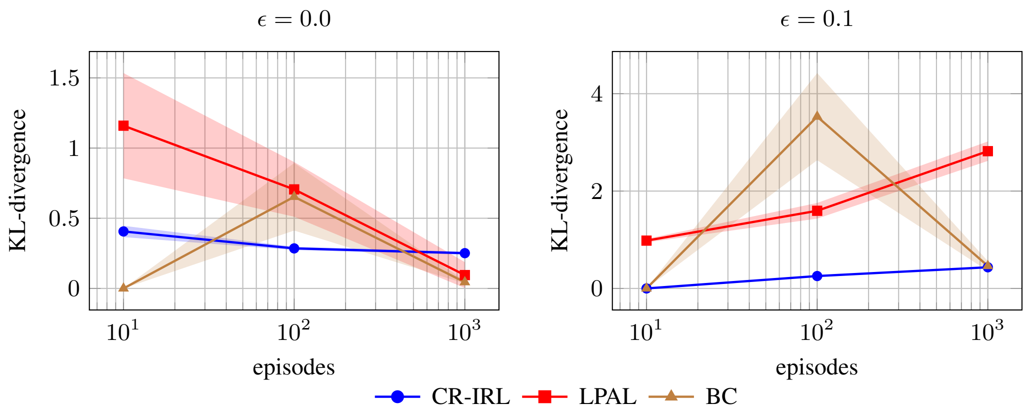

Figure 5.9: KL-divergence between expert's policy and learned policy for the Taxi problem as a function of the number of expert's trajectories.

5.3 Linear Quadratic Gaussian Regulator

The Linear Quadratic Gaussian Regulator (LQG) [30] corresponds to the discrete time control problem in a continuous state-action space. The state and reward equations are given by:

$$\begin{aligned} \mathbf{s}_{t+1} &= \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{a}_t + \boldsymbol{\eta}_t, \\ r_t &= -\mathbf{s}_t^T \mathbf{Q}\mathbf{s}_t - \mathbf{a}_t^T \mathbf{R}\mathbf{a}_t, \end{aligned}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}$ and $\mathbf{R}$ are coefficient matrices and $\boldsymbol{\eta}_t$ is a noise process assumed to be a Gaussian white noise $\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda})$ with uncorrelated components $\boldsymbol{\Lambda} = \lambda^2 \mathbf{I}$. The goal of the agent is to reach as soon as possible the origin since it receives, at each time step, a penalization proportional to the magnitude of the state and the action. The optimal control policy in steady state conditions is the linear controller $\mathbf{a}_t = \mathbf{K}\mathbf{s}_t$ where matrix $\mathbf{K}$ can be found by solving the Riccati equation [30]. For the expert's policy, we consider the class of Gaussian policies $\mathbf{a}_t \sim \mathcal{N}(\mathbf{K}\mathbf{s}_t, \boldsymbol{\Sigma})$ with parameter $\mathbf{K}$ ($\boldsymbol{\Sigma}$ is fixed):

$$\mathbf{a}_t = \mathbf{K}\mathbf{s}_t + \boldsymbol{\xi}_t, \tag{5.3}$$

where $\boldsymbol{\xi}_t$ is a Gaussian white noise $\boldsymbol{\xi}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with uncorrelated components ($\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$) and uncorrelated with $\boldsymbol{\eta}_t$. For implementation reasons, we consider a bounded state-action space $\mathcal{S} = [-10, -10]^n$, $\mathcal{A} = [-8, 8]^n$, where n is the dimension of the state (and action) space. All components of the initial state are fixed to -4, the discount factor is 0.99 and the trajectory horizon 100. Like in the Taxi problem we consider the case of imperfect expert; we use σ^2 (policy variance) to tune the degree of exploration of the expert (we consider $\sigma^2 \in \{1, 0.01\}$).

We perform experiments in the one-dimensional case ($n = 1$, 1D-LQG), the values of the coefficients are reported in Table 5.1. The derivation of k^*, the gradient and Hessian of policy (5.3) are reported in Appendix B.4.

Table 5.1: Coefficients of the 1D-LQG experiments.

a	b	q	r	λ^2	k^*
1.0	1.0	0.9	0.9	0.1	−0.61525

Table 5.2: Policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of the recovered reward functions of 1D-LQG for different values of σ^2.

Reward function	$\sigma^2 = 1$	$\sigma^2 = 0.01$
Original reward	-0.09203 ± 0.5076	0.03185 ± 0.08893
Advantage function	-0.09356 ± 0.4983	0.9083 ± 1.467
CR-IRL	2.822 ± 10.77	-0.7514 ± 55.45
GIRL-abs-val	(-2.587 ± 1.422)e-15	(1.858 ± 6.068)e-14
GIRL-square	(-1.256 ± 1.534)e-14	(-2.144 ± 5.232)e-14

Table 5.3: Policy Hessian $\mathcal{H}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of the recovered reward functions of 1D-LQG for different values σ^2.

Reward function	$\sigma^2 = 1$	$\sigma^2 = 0.01$
Original reward	-3.064 ± 7.262	-6.371 ± 5.847
Advantage function	-3.601 ± 6.521	-223.1 ± 114.4
CR-IRL	-1854 ± 147.5	-28702 ± 2586
GIRL-abs-val	3.572 ± 12.33	-17.25 ± 32.84
GIRL-square	-0.3941 ± 8.218	-6.213 ± 4.956

We compare CR-IRL with GIRL [94] using two linear parametrizations of the reward function:

$$R(s, a, \boldsymbol{\omega}) = \omega_1 s^2 + \omega_2 a^2 \qquad \text{(GIRL-square)},$$
$$R(s, a, \boldsymbol{\omega}) = \omega_1 |s| + \omega_2 |a| \qquad \text{(GIRL-abs-val)}.$$

Table 5.2 and Table 5.2 show the values of the policy gradient and the policy Hessian computed for the different recovered reward functions. GIRL, by construction, yields the reward functions with the smallest gradient in absolute value. On the contrary CR-IRL provides a reward function with larger gradient variance; this is justified by the fact that the Hessian is a large negative number making the policy parameter value a very unstable point for the expected return and so the gradient estimation more noisy.

In Figure 5.10 the recovered reward functions are represented. We can see that GIRL-square is able to recover almost exactly the original reward (but it is very sensitive to the available expert's trajectories), indeed the original reward function falls into the considered class of rewards. CR-IRL recovers rewards that penalize significantly the disadvantageous regions, this favors the learning speed as the experiments demonstrate. Clearly, as σ^2 increases the penalized region reduces.

5.3.1 Learning speed experiments

The training is performed using REINFORCE [121] with Adam [58] (learning rate 0.002, other parameters with default value), at each iteration 100 trajectories are used to estimate the gradient, the maximum number of iterations is 600 and the initial parameter is $k = -0.2$. All data are averaged over 40 runs and the error bars correspond to the 95% confidence intervals.

Figure 5.11 shows the parameter value learned with REINFORCE using a Gaussian policy with two different variances. We notice that CR-IRL, fed with 20 expert's trajectories, converges closer and faster to the expert's parameter w.r.t. to the original reward, advantage function and GIRL with both parametrizations when $\sigma^2 = 0.01$. Instead, the convergence value of CR-IRL when the expert is explorative $\sigma^2 = 1.0$ deviates significantly from the optimal value. This however, does not affect the average return, as shown in Figure 5.12. Indeed, when the variance is large the effect of the parameter k (the mean of the distribution) on the average return is less relevant.

5.3.2 Sensitivity to the number of expert's trajectories

We investigate the effect of the number of expert's demonstrations on the performance of the considered IRL algorithms. Figure 5.13 reports the convergence parameter value and the average return for a variable number of expert's trajectories. We notice that CR-IRL is less sensitive w.r.t. to GIRL since, even if few expert's trajectories are provided, CR-IRL penalizes the regions that the expert did not visit, avoiding the agent to fall into low-reward regions.

5.3.3 Soft penalization

In order to extend the recovered reward function to unseen state-action pairs we adopt a 2NN model with a Gaussian kernel (bandwidth $\sigma_{\mathcal{S}} = \sigma_{\mathcal{A}} = 4$). Differently from the discrete case, in the continuous case the penalization must be "soft", i.e., the more a non-visited state-action pair is far from the visited ones the more it is penalized. The penalized reward function is obtained as a convex combination ($\alpha = 0.5$) between the non-penalized reward function and a measure of the state-action occupancy:

$$R^{\text{penalized}}(\mathbf{s},\mathbf{a}) = \alpha \bar{R}^{\text{non-penalized}}(\mathbf{s},\mathbf{a}) + (1-\alpha)\bar{p}(\mathbf{s},\mathbf{a}), \qquad \forall (\mathbf{s},\mathbf{a}) \notin \mathcal{D},$$

where $\bar{R}^{\text{non-penalized}}$ is a scaled version of the recovered reward within the interval $[0,1]$ and $\bar{p}$ is computed with a "non-normalized" Gaussian kernel KNN approach

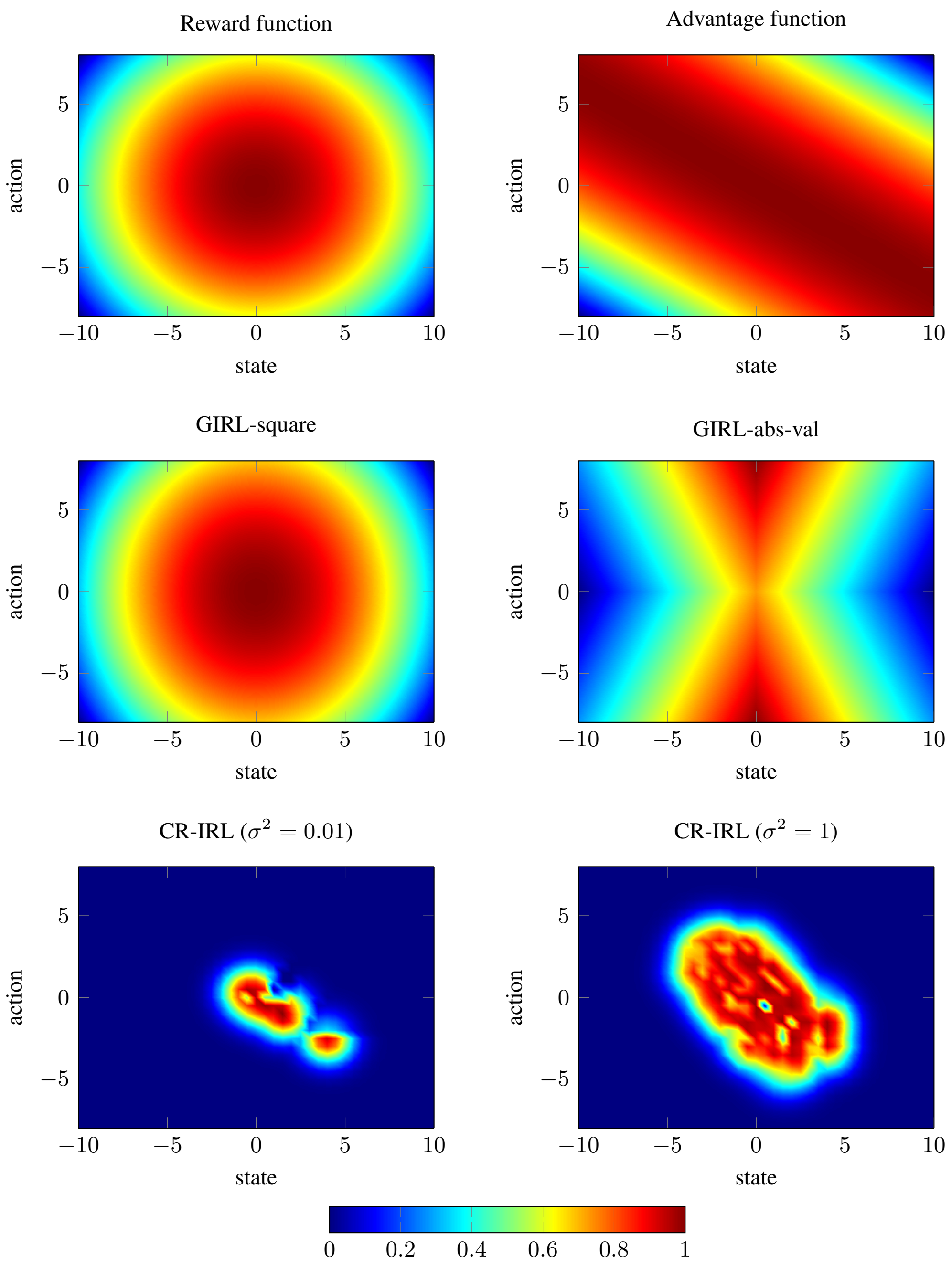

Figure 5.10: Representation of the recovered reward functions for 1D-LQG.

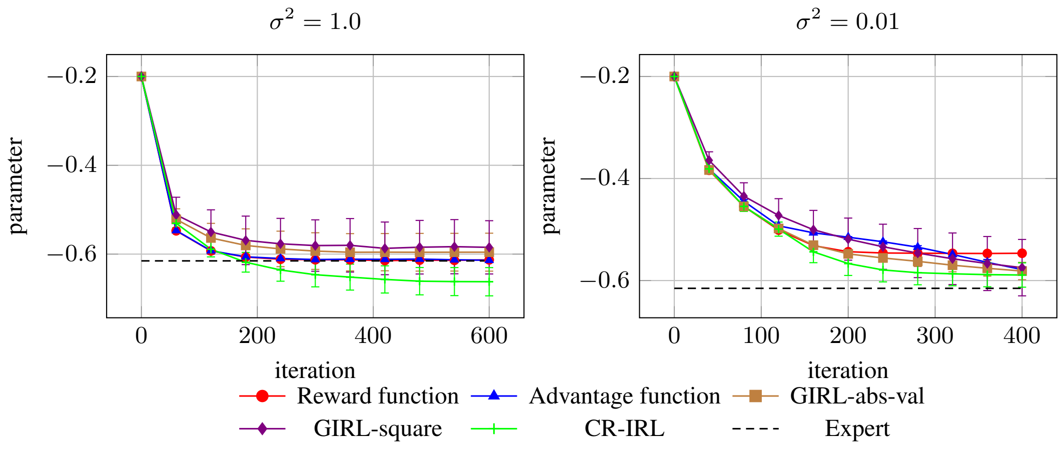

Figure 5.11: Parameter value of 1D-LQG as a function of the number of iterations of REINFORCE.

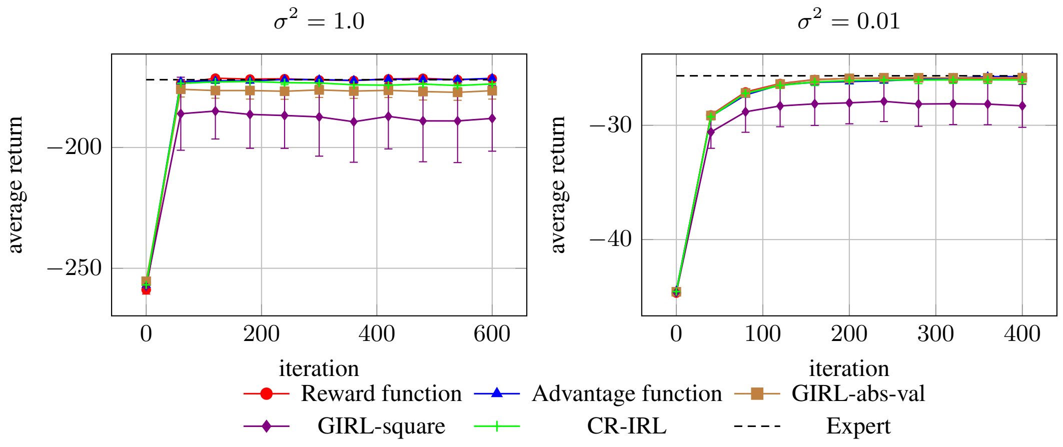

Figure 5.12: Average return of 1D-LQG as a function of the number of iterations of REINFORCE.

(bandwidth $\sigma_{\bar{p}} = 4$) as:

$$\bar{p}(\mathbf{s}, \mathbf{a}) = \frac{\sum_{(\mathbf{s}', \mathbf{a}') \in \mathrm{KNN}((\mathbf{s}, \mathbf{a}), k, \mathcal{D})} \mathcal{K}((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}'))}{\max_{(\mathbf{s}'', \mathbf{a}'') \in \mathcal{D}} \sum_{(\mathbf{s}', \mathbf{a}') \in \mathrm{KNN}((\mathbf{s}'', \mathbf{a}''), k, \mathcal{D})} \mathcal{K}((\mathbf{s}'', \mathbf{a}''), (\mathbf{s}', \mathbf{a}'))}, \quad \forall (\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A},$$

where $\mathrm{KNN}((\mathbf{s}, \mathbf{a}), k, \mathcal{D})$ is the set of the k closest state-action pairs according to the Euclidean distance and given samples in the dataset $\mathcal{D}$ (we use $k = 5$) and $\mathcal{K}$ is a Gaussian kernel. The resulting penalized reward function ranges in the interval $[0, 1]$. We tested other approaches to perform penalization, such as $R^{\mathrm{penalized}}(\mathbf{s}, \mathbf{a}) = \bar{R}^{\mathrm{non-penalized}}(\mathbf{s}, \mathbf{a}) \bar{p}(\mathbf{s}, \mathbf{a})$, no significant difference in performance was detected.

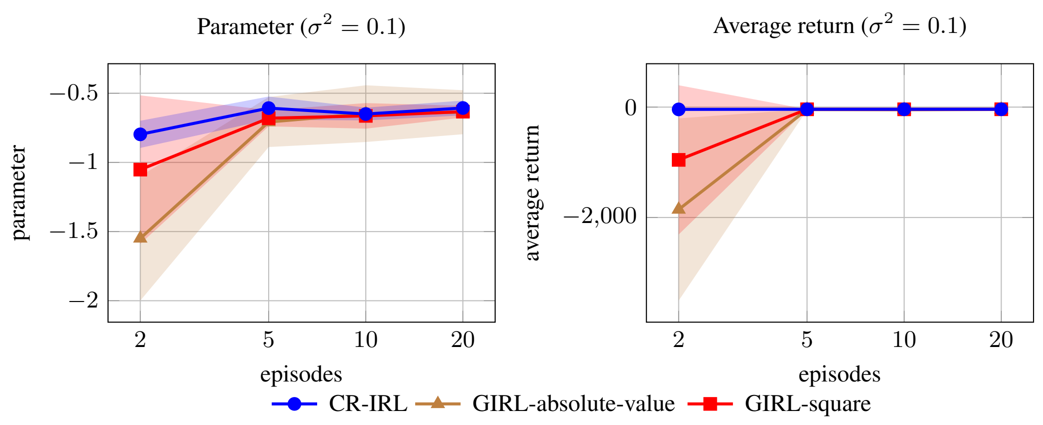

Figure 5.13: Parameter and average return of 1D-LQG as a function of the number of expert's trajectories.

5.3.4 Sensitivity to penalization and number of neighbors

We also investigate the effect of the number of neighbors in the extension of the recovered reward function on unvisited state-action pairs. Figure 5.14 shows the parameter value and the average return as a function of the number of iterations (the bandwidth of the Gaussian kernel are fixed to $\sigma_{\mathcal{S}} = \sigma_{\mathcal{A}} = 4$) and for different values of the expert's policy variance σ^2. The KNN interpolation allows reconstructing the reward even in regions of the state-action space that are very far from the region visited by the expert, however, in those regions, the recovered reward is not reliable. We compared the performance of the recovered reward function as it is and the performance achieved by penalizing the state-action pairs far from the expert exploration region. In the former case (dashed lines) the convergence is slow and displays a high variance, while in the latter case (solid lines) the learning curve converges faster to the optimal parameter. The number of neighbors has no relevant impact, at least for sufficiently small values, when penalization is applied; while when no penalization is applied the effect is more visible. In particular, when the expert is explorative the recovered reward is smoother (many state-action pairs are visited) so 1NN yields the best performance. As the expert becomes less explorative, a large number of neighbors (50NN) performs better since the recovered reward is more discontinuous. We notice that the penalization becomes more beneficial when the expert's policy is exploitative ($\sigma^2 = 0.01$).

5.3.5 Experiments with two-dimensional LQG

In order to compare the second-order criteria we discussed in Chapter 4, we perform a series of experiments with the two-dimensional LQG (2D-LQG). We consider two settings, as reported in Table 5.4. In the first case (a) the dynamics of the two states are fully independent, this is equivalent to considering two one-dimensional LQG running in parallel. In the second case (b), instead, one state influences the other in

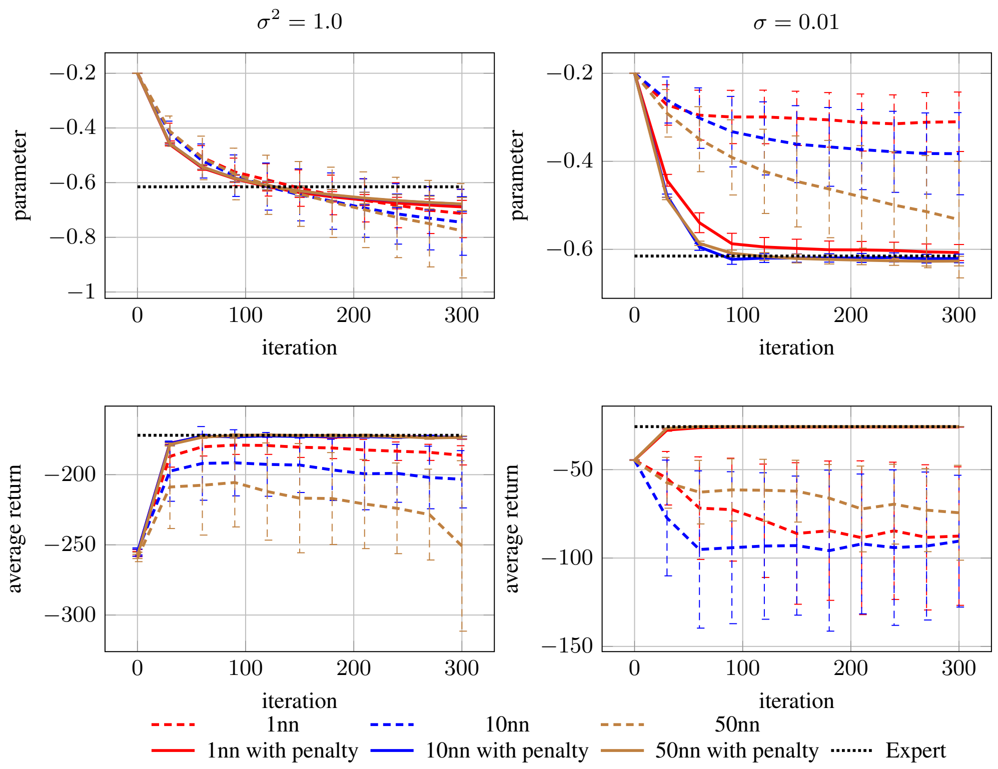

Figure 5.14: Parameter value and average return of 1D-LQG as a function of the number of iterations of REINFORCE with different number of neighbors.

a symmetric manner; as a consequence, the optimal controller matrix $\mathbf{K}^*$ is no longer diagonal. We collect 20 trajectories of length 100 from a Gaussian expert's policy $\mathcal{N}(\mathbf{K}^*\mathbf{s}, \boldsymbol{\Sigma})$, with covariance matrix $\boldsymbol{\Sigma} = 0.01\mathbf{I}$. We use them to fit via maximum likelihood the parameters (see Table 5.4 for the ML parameter) of a Gaussian policy:

$$\pi_{\mathbf{k},\boldsymbol{\sigma}^2} \sim \mathcal{N}\big(\mathrm{diag}(\mathbf{k})\mathbf{s}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big). \tag{5.4}$$

Essentially, we restrict our attention to the diagonal control matrices $\mathbf{K} = \mathrm{diag}(\mathbf{k})$ and we assume that the components of the Gaussian policy are fully uncorrelated $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma}^2) = 0.01\mathbf{I}$. Clearly, the optimal Gaussian controller for the case in which the state dynamics are independent falls into this class of policies (a), whereas these policies are not able to represent the optimal controller for the second case (b).

We compare the maximum eigenvalue optimality criterion (ME-opt), the trace optimality criterion (TR-opt) and the trace heuristic (TR-heu). Tables 5.5 and 5.6 report the eigenvalues and the trace of the recovered reward functions for cases (a) and (b) respectively. In Figure 5.15 the expected return is represented with its second order approximation as a paraboloid in canonical form:

$$J(\tilde{k}_1, \tilde{k}_2) \approx J(k_1^{ML}, k_2^{ML}) + \lambda_1(\tilde{k}_1^{\,2} - k_1^{ML})^2 + \lambda_2(\tilde{k}_2^{\,2} - k_2^{ML})^2, \tag{5.5}$$

where $\tilde{k}_1$ and $\tilde{k}_2$ correspond to the directions associated to the first and the

Table 5.4: Coefficients of the 2D-LQG experiments.

case	$\mathbf{A}$	$\mathbf{B}$	$\mathbf{Q}$	$\mathbf{R}$	λ^2	$\mathbf{K}^*$	$\mathbf{k}^{\mathrm{ML}}$
(a)	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\mathbf{I}$	$0.9\mathbf{I}$	$0.9\mathbf{I}$	0.1	$\begin{pmatrix} -0.618 & 0 \\ 0 & -0.618 \end{pmatrix}$	$\begin{pmatrix} -0.618 \\ -0.618 \end{pmatrix}$
(b)	$\begin{pmatrix} 1 & 0.1 \\ 0.1 & 1 \end{pmatrix}$	$\mathbf{I}$	$0.9\mathbf{I}$	$0.9\mathbf{I}$	0.1	$\begin{pmatrix} -0.621 & -0.083 \\ -0.083 & -0.621 \end{pmatrix}$	$\begin{pmatrix} -0.690 \\ -0.690 \end{pmatrix}$

Table 5.5: Eigenvalues and trace of the policy Hessian for the recovered reward functions for the 2D-LQG case (a).

Reward function	first eigenvalue	second eigenvalue	trace
ME-opt	$(-1.34 \pm 0.129)e+4$	$(-1.34 \pm 0.129)e+4$	$(-2.68 \pm 0.258)e+4$
TR-opt	$(-2.72 \pm 0.273)e+4$	$(-6.66 \pm 1.46)e+3$	$(-3.40 \pm 0.285)e+4$
TR-he	$(-1.58 \pm 0.171)e+4$	$(-7.79 \pm 1.13)e+3$	$(-2.36 \pm 0.249)e+4$

Table 5.6: Eigenvalues and trace of the policy Hessian for the recovered reward functions for the 2D-LQG case (b).

Reward function	first eigenvalue	second eigenvalue	trace
ME-opt	$(-2.79 \pm 0.478)e+4$	$(-2.18 \pm 0.232)e+4$	$(-4.97 \pm 0.664)e+4$
TR-opt	$(-2.94 \pm 0.112)e+5$	$(-1.58 \pm 0.920)e+3$	$(-2.96 \pm 0.113)e+5$
TR-he	$(-1.72 \pm 0.134)e+5$	$(-7.76 \pm 1.50)e+3$	$(-1.80 \pm 0.146)e+5$

second eigenvector. Note that the first degree term is missing since (k_1^{ML}, k_2^{ML}) is a stationary point of the expected return.

From Figure 5.15 we observe that for case (a) the recovered rewards display a similar shape. This is a consequence of the fact that the two dynamics are independent, therefore the directions of the eigenvectors are not affected one another. On the contrary in case (b), we observe some differences. The ME-opt criterion tends to recover rewards with isotropic shape: the eigenvalues (see Tables 5.5 and 5.6) are very close. The TR-opt and the TR-heu, which is its surrogate, tend to sharpen the curve in one direction as much as possible leaving the other direction almost flat.

In terms of learning speed we observe in Figure 5.16 that all the reward functions recovered by CR-IRL allow reaching closely the optimal parameter faster w.r.t. the original reward function. However, in case (a), it is visible that the original reward function converges closer to the optimal parameter. The three criteria, in case (a), do not display significant differences. On the contrary, in case (b), we observe that the reward function recovered by ME-opt displays a better performance w.r.t. the original reward function, converging closer and faster to the optimal parameter. The deviation from the optimal parameter, however, does not have a visible effect on the average return, since all curves in Figure 5.17 are almost overlapping.

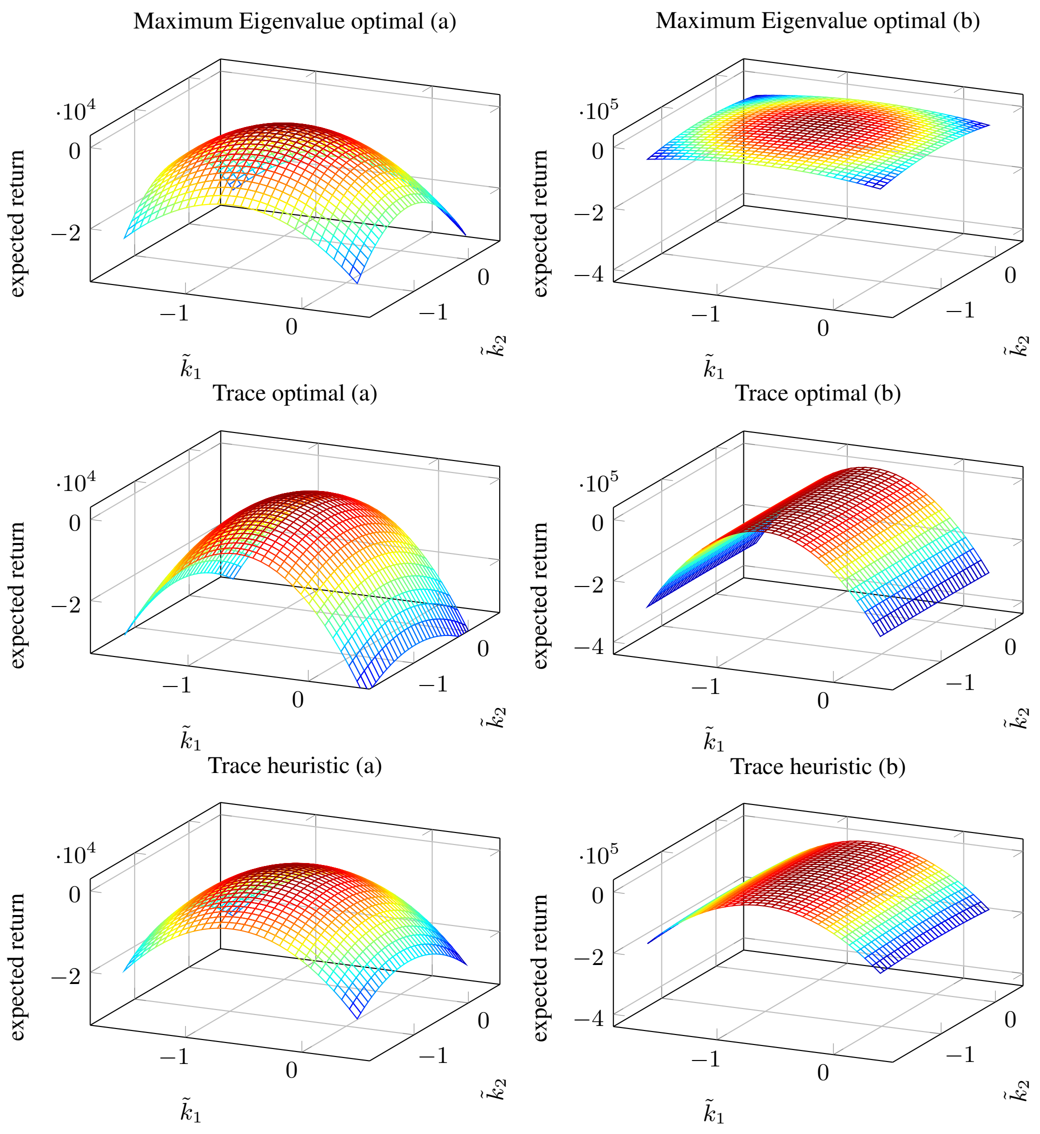

Figure 5.15: Second order approximation of the expected return for the 2D-LQG.

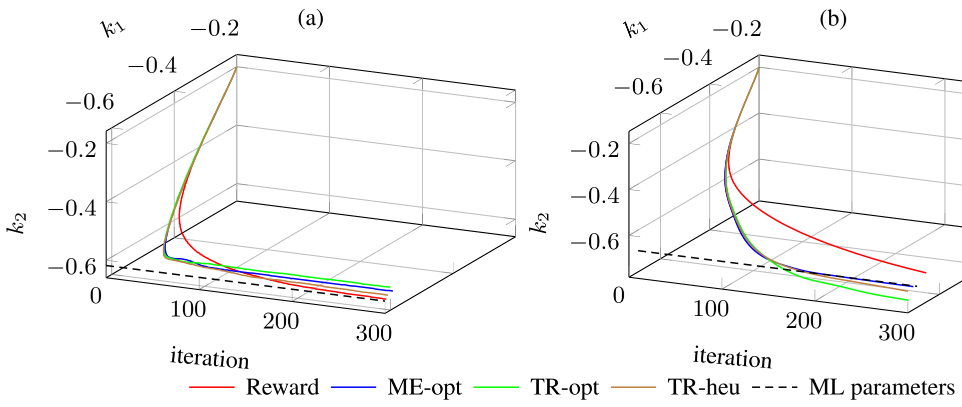

Figure 5.16: Parameter value for the 2D-LQG as a function of the number of iterations of REINFORCE.

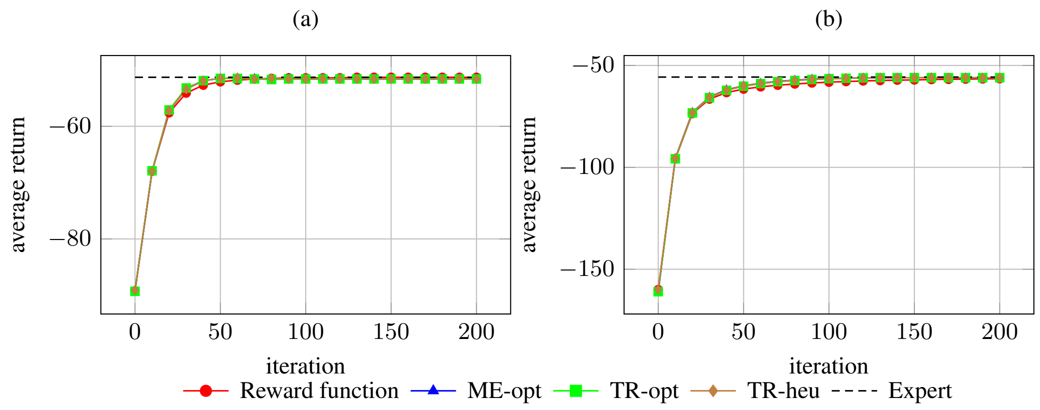

Figure 5.17: Average return for the 2D-LQG as a function of the number of iterations of REINFORCE.

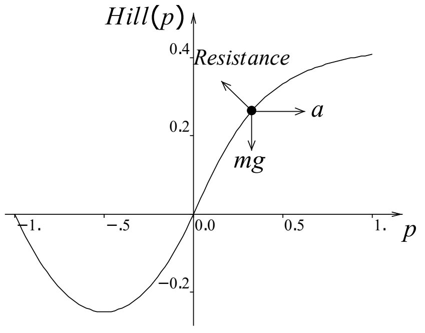

Figure 5.18: Representation of Hill(p) (shape of the hill) and of the different forces applied to the car (from [33]).

5.4 Car on the Hill

We further experiment CR-IRL in the continuous Car on the Hill domain [33]. The environment considers a car modeled by a point mass that is traveling on a hill (the shape of which is given by the function Hill(p) of Figure 5.18). The action a acts directly on the acceleration of the car. The original formulation assumes to have only two extreme values (full acceleration ($a = 4$) or full deceleration ($a = -4$)), but for our convenience we admit the full range $a \in [-4, 4]$. The control problem objective is roughly to bring the car in a minimum time to the top of the hill ($p = 1$ in Figure 5.18) while preventing the position p of the car to become smaller than -1 and its speed v to go outside the interval $[-3, 3]$. Therefore, the state space is continuous of dimension two (the position p and the speed v of the car).

The dynamics of the problem is reported in the followings:

$$\begin{aligned}
\dot{p} &= v, \\
\dot{v} &= \frac{a}{m(1 + \mathrm{Hill}'(p)^2)} - \frac{g\mathrm{Hill}'(p)}{1 + \mathrm{Hill}'(p)^2} - \frac{v^2\mathrm{Hill}'(p)\mathrm{Hill}''(p)}{1 + \mathrm{Hill}'(p)^2},
\end{aligned}$$

where m is set to 1 and $g = 9.81$ is the gravitational acceleration. The function Hill(p) is defined as:

$$\mathrm{Hill}(p) = \begin{cases} p^2 + p & \text{if } p < 0, \\ \frac{p}{\sqrt{1+5p^2}} & \text{if } p \geq 0. \end{cases} \tag{5.6}$$

The discrete-time dynamics is obtained by discretizing the time with the time between t and $t + 1$ chosen equal to $0.1s$. If p_{t+1} and v_{t+1} are such that $|p_{t+1}| > 1$ or $|v_{t+1}| > 3$ then a terminal state is reached. The reward function $R(p, v, a)$ is defined through the following expression:

$$R(p_t, v_t, a_t) = \begin{cases} -1 & \text{if } p_{t+1} < -1 \text{ or } |v_{t+1}| > 3, \\ +1 & \text{if } p_{t+1} > 1 \text{ and } |v_{t+1}| \leq 3, \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

The decay factor γ has been chosen equal to 0.95 and the initial state is $p_0 = -0.5$ and $v_0 = 0$. From now on we will indicate with $\mathbf{s}$ the pair (p, v).

We approximate the optimal deterministic policy π^E_{FQI} via FQI [33] run on 1000 random trajectories, using 50 Extremely Randomized Trees [39] (criterion: mse, min samples split: 5, min samples leaf: 2, iterations: 20). We consider different levels of noisy expert, so that a random action is selected with probability $\epsilon \in \{0, 0.1, 0.2\}$:

$$\pi^E_\epsilon(a|\mathbf{s}) = (1 - \epsilon)\pi^E_{FQI}(a|\mathbf{s}) + \epsilon\pi_{\text{random}}(a|\mathbf{s}).$$

We exploit $N = 20$ expert's trajectories to estimate the parameters $\mathbf{w}$ of a Gaussian policy $\pi_\mathbf{w}(a|\mathbf{s}) \sim \mathcal{N}(y_\mathbf{w}(\mathbf{s}), \sigma^2)$, with σ^2 set to 0.01. The mean $y_\mathbf{w}(\mathbf{s})$ is obtained as a radial basis function network [16]:

$$y_\mathbf{w}(\mathbf{s}) = \sum_{i=1}^{k} w_i e^{-\delta\|\mathbf{s}-\mathbf{s}_i\|^2}, \tag{5.8}$$

where $\delta = 0.01$ and we consider 40×40 centers $\mathbf{s}_i$ uniformly distributed in the two-dimensional (position-speed) state space $[-4, 4] \times [-3, 3]$. The detailed derivation of gradient and Hessian of the policy is reported in Appendix B.6. Notice that this policy allows to perform actions in the interval $[-4, 4]$ and not only in $\{-4, 4\}$.

Figure 5.19 shows the original reward function and the reward functions recovered by CR-IRL for the different values of ϵ. The original reward function displays three regions associated to the reward values $+1$, 0 and -1. However, most of the $+1$ region is never reached resulting in penalization when considering the reward functions recovered by CR-IRL. Those functions assign non-zero reward only to the region to which the original reward would assign 0 and to the absorbing state reached as the car gets to the top of the hill with sufficiently small speed. At first glance the reward functions recovered by CR-IRL are significantly different w.r.t. the original reward and seem not to be informative. Nevertheless, the empirical evaluation (see 5.4.1) demonstrates that they allow computing the optimal policy.

5.4.1 Learning speed experiments

This experiment is intended to show that the reward function recovered by CR-IRL does not necessary need to be used with policy gradient approaches. We compare the average return as a function of the number of iterations of FQI, fed with the different recovered rewards.

Figure 5.20 shows that FQI converges faster to optimal policy when coped with the reward recovered by CR-IRL rather than with the original reward, regardless of

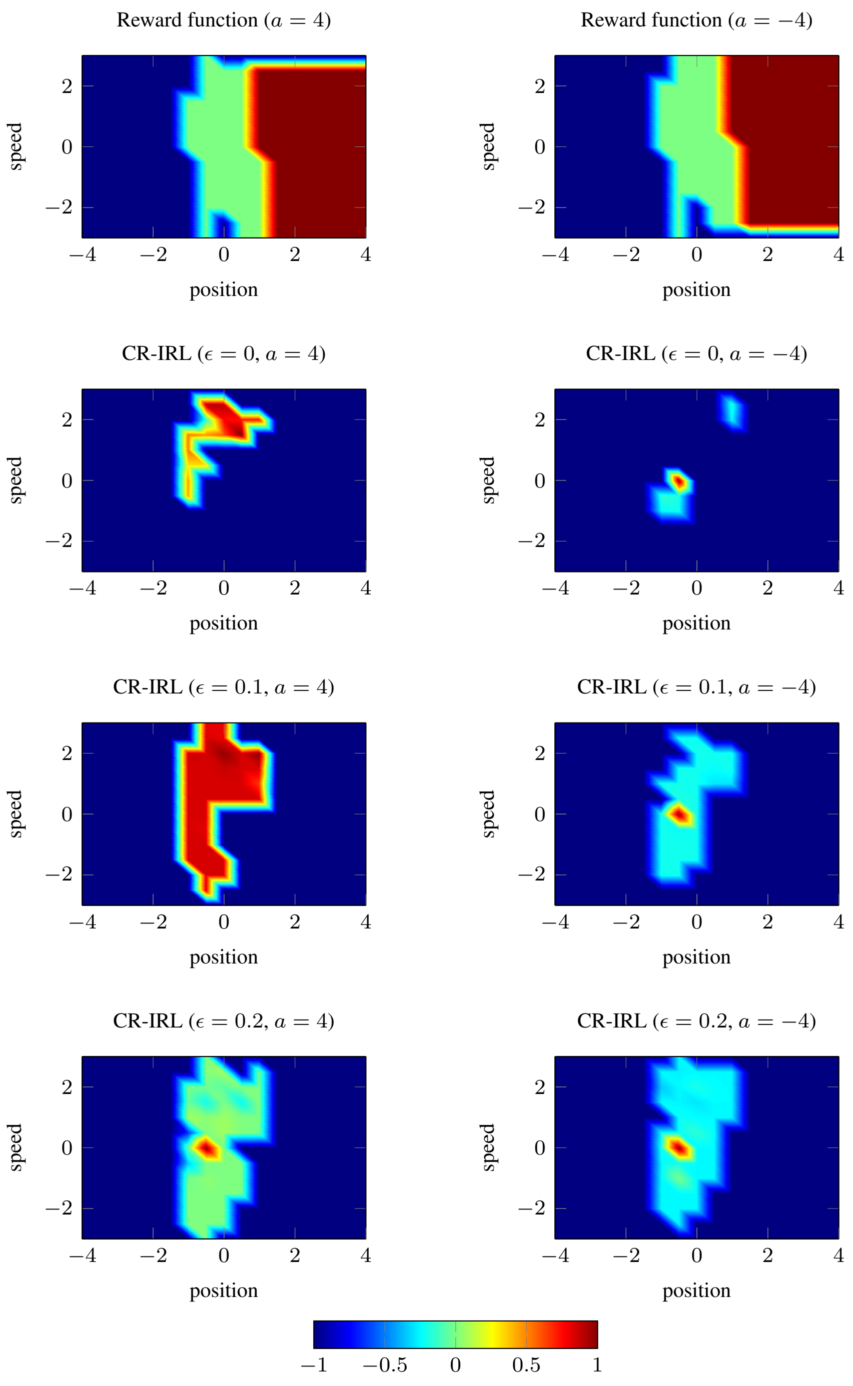

Figure 5.19: Representation of the recovered reward functions for Car on the Hill.

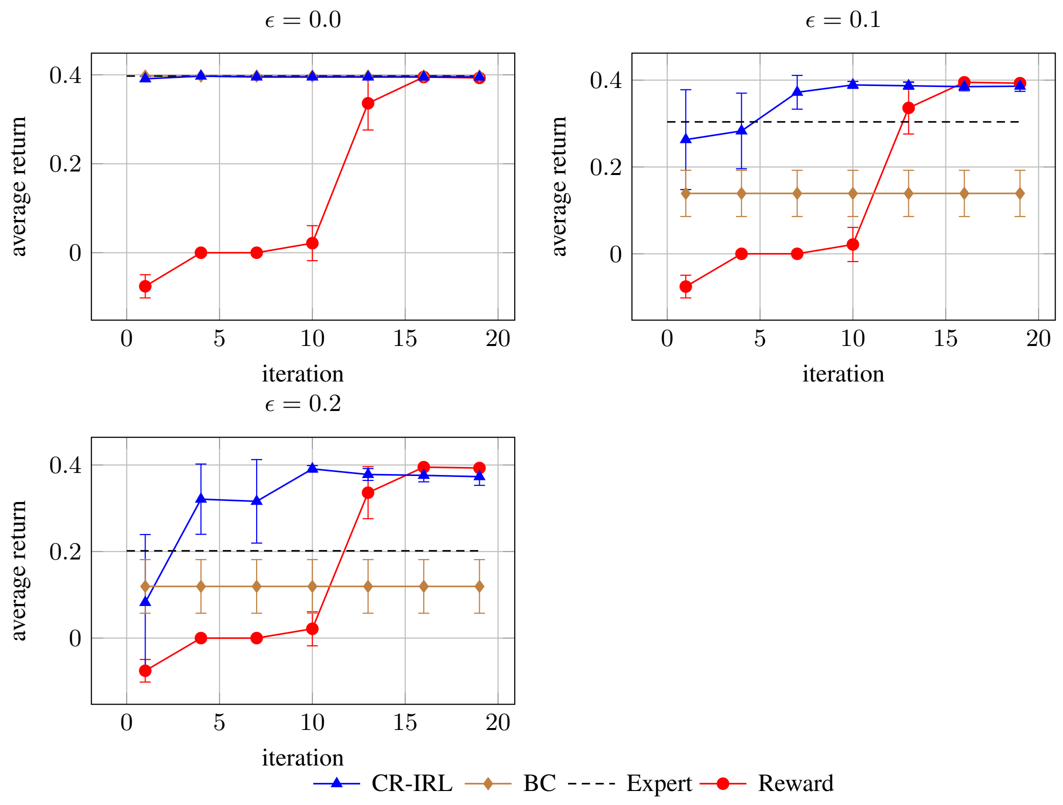

Figure 5.20: Average return of Car On Hill problem as a function of FQI iterations varying the value of exploration ϵ.

the exploration level ϵ. When the expert is deterministic ($\epsilon = 0$), BC reaches the expert's performance, since the considered class of parametric policies is sufficiently expressive to represent the optimal deterministic policy. Also CR-IRL reaches the expert's performance in one iteration only.

BC however is not resilient to noise, displaying a significant performance degradation as ϵ increases. CR-IRL, on the other hand, is able to recover the optimal policy quickly even in case of explorative experts.

In Figure 5.21 we compare the trajectories of the expert's policy, the maximum likelihood policy (BC) and the policy computed via FQI from the reward recovered by CR-IRL. We can see that when the expert is deterministic the trajectories are almost overlapping. On the contrary, when the exploration rate ϵ increases we can see that some expert's trajectories fail to reach the profitable absorbing state. This is a consequence of the fact that a random action is taken with probability ϵ. BC is almost always able to get the final $+1$ reward but not optimally in terms of number of decision epochs required. Finally, CR-IRL, even if trained with a noisy expert, recovers a reward function that induces a policy which is near-optimal, as all trajectories get to the $+1$ final reward in almost the minimum number of steps.

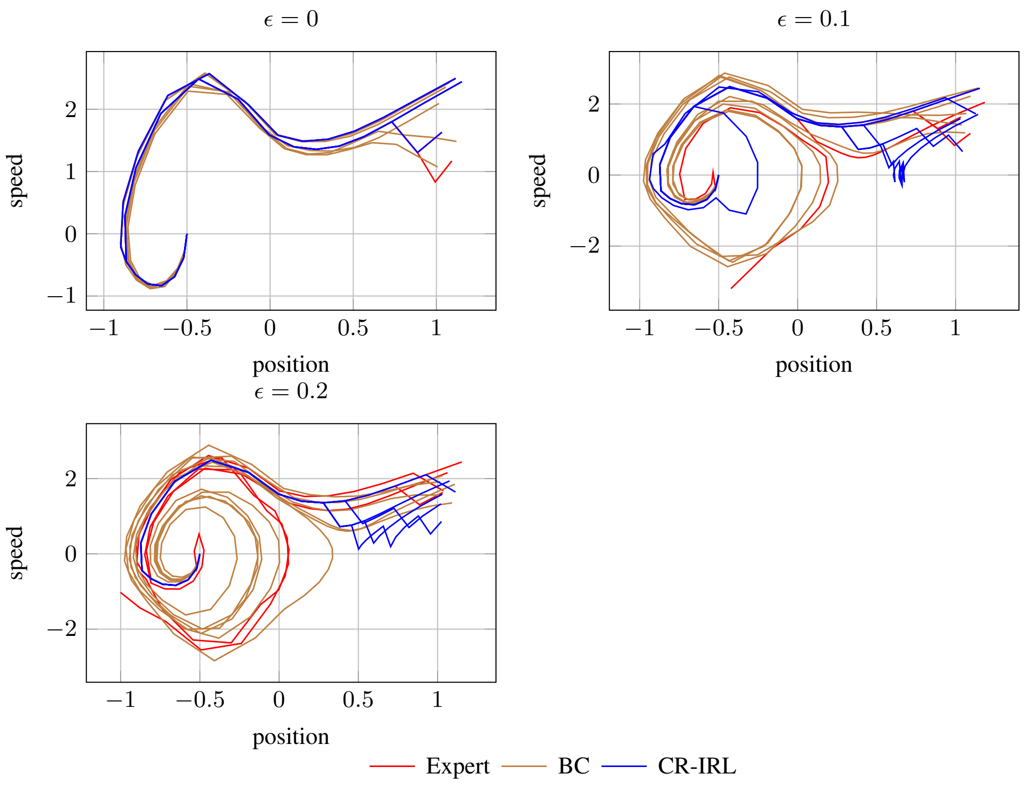

Figure 5.21: Trajectories of the expert's policy, the ML policy and the policy computed via FQI from the reward recovered by CR-IRL for different values of ϵ.

Chapter 6

Conclusions

This thesis provided several algorithmic and experimental contributions to the Inverse Reinforcement Learning area. IRL methods experienced a notable advancement in the last years and several successful applications have been proposed, mainly related to the field of autonomous locomotion. However, most of the state-of-the-art algorithms remain limited to simple domains due to the need of the environment model and to the computational demands. Early approaches to IRL are not, in most of the cases, suitable for real-world applications due to the requirement of solving the forward RL problem for each candidate reward function. Besides being expensive in terms of samples and execution time, this step requires having access to the environment in order to learn the optimal policy or even to know the complete transition kernel. Moreover, the need of defining a priori a set of engineered reward features represents a further obstacle. Approaches based on deep learning overcome both problems, exploiting the representational capabilities of neural networks. Nevertheless, those methods require the availability of huge amount of expert's data and powerful computational resources.

The demand of approaching the IRL problem in a flexible and efficient way is becoming more crucial. Clearly, behavioral cloning reduces the computational effort and does not require the access to the environment. Nevertheless, BC is constrained to the choice of a specific policy space, sometimes unable to represent the expert's behavior with sufficient degree of accuracy. Furthermore, the optimal policy, as recovered by BC, cannot be transferred across different environments. The optimal policy and the reward function are different ways to encode an optimal behavior: a reward function induces an optimal policy, while a policy is optimal under certain classes of rewards. However, these concepts are not exchangeable when the dynamics of the environment in unknown. Knowing the reward allows recomputing the optimal policy even under modifications of the environment model, whereas the optimal policy is deeply related to the specific environment.

We believe that a key step towards enforcing the scalability of IRL to real-world domains is the availability of an algorithm able to construct transferable reward functions requiring just a set of expert's trajectories. Most of the contributions of this work point to that direction. In Chapter 4 we presented CR-IRL that mixes BC and IRL in order to construct the features and single out a reward function. BC represents a preliminar step, necessary in order to extract the set of ECO-Q making the gradient vanish. It is worth noting that, for our algorithm, the choice of the policy model is less constraining w.r.t. pure behavioral cloning. The maximum likelihood policy is never used to collect samples, thus the performance gap w.r.t. the expert's policy is not so relevant. Nevertheless, this represents a limitation of CR-IRL, as a poor approximation would make the algorithm extract a suboptimal set of ECO-Q. The main theoretical contribution of this thesis lies in the notion of policy rank. We think that the informativity of a parametric policy is an appealing notion that, together with the bound (4.1), can be used also outside the area of IRL and particularized for specific classes of policies. Once the ECO-Q have been extracted, we proposed two methods to construct the set of ECO-R. Clearly, we focused on the model-free approach that exploits reward shaping in order to build the space of advantage functions. This choice, of course, causes the reduction of the space dimensionality that prevents CR-IRL from considering certain classes of rewards. Nevertheless, using the advantage function as reward function is beneficial for almost all RL algorithms. Finally, we introduced several second-order approaches to select a reward function in the constructed space. The multi-objective interpretation offers a unifying view of the problem, that we particularized for specific choices of optimality criteria, ending in a heuristic that provides good empirical results.

Comparing the quality of different reward functions for the same environment is a non trivial task. The metrics we choose are directed towards the evaluation of the ability of the recovered reward functions of quickly learning the optimal policy. Clearly, this notion is algorithm-dependent, but it is so intuitive that we believe it is the most suitable comparison approach. In Chapter 5 we showed that CR-IRL is able to recover reward functions that can be used to learn optimal policies (in terms of expected return) but at a faster learning rate w.r.t. the original reward function of the problem. Moreover, even though CR-IRL exploits BC to get a parametric representation of the expert's policy, the policies learned with the reward function recovered by CR-IRL significantly outperform BC. We think this is a key strength of our algorithm, resulting from merging BC and IRL. BC brings to CR-IRL the ability to avoid low-reward regions that the expert has never visited, while IRL, producing a reward representation, allows to overcome the limit of the fixed policy model. Furthermore, CR-IRL outperforms several popular IRL methods designed to recover a reward function as a linear combination of given features, when fed with the set of ECO-R. On the other hand, the performance of the trace heuristic when CR-IRL is fed with automatically generated features, such as Proto-Value Functions, is somehow unsatisfactory. This is a consequence of the fact that the two phases of CR-IRL, feature construction and reward recovery, both originate from conditions on the policy gradient and Hessian and therefore are not independent. The second-order optimality criteria make sense only when the policy gradient is null and take into account only the regions of the environment visited by the expert.

Even though the algorithm we proposed resulted effective in several problems, numerous questions remain open. We believe there is space for further research in this topic, mainly directed towards theoretical and experimental settings. We outline in the followings the main research lines.

Theoretical analysis of the maximum likelihood policy

CR-IRL, as already seen, requires to estimate a parametric representation of the expert's policy from the available trajectories. This is a critical point since it requires the choice of a policy model. Although the estimated policy is not used to collect samples, its accuracy might affect the construction of the ECO-Q space and, as a result, the recovered reward function. We have identified two sources of error. First, the maximum likelihood estimation is based on expert's trajectories, few trajectories lead to poor approximation even if the expert's policy falls into the chosen policy space. Second, when the expert's policy cannot be represented by the policy space, how does the distance between the expert's policy and the maximum likelihood policy affect the reward construction? In particular, it should be interesting deriving a bound relating the performance gap and the policy distance and study how this error propagates in the policy gradient and for the construction of the set of ECO-Q and ECO-R. We have already encountered this problem in the empirical evaluation of the Car on the Hill problem. The expert's policy, built via FQI with ExtraTrees, is used to collect samples and fit a Gaussian policy. Although the latter displayed a significantly smaller expected return, the reward provided by CR-IRL allows learning, via FQI, a policy whose performance is comparable with that of the expert.

Direct construction of ECO-R

Our approach requires a two-step procedure to build the approximation space for the reward function. Besides introducing further sources of computational complexity (in particular the usage of SVD to orthogonalize the set of ECO-R), this contributes to the propagation of the error introduced in the maximum likelihood estimation of the expert's policy. In order to overcome this limitation we could resort to trajectory-based formulations of the policy gradient, like (2.30), in which the reward function appears in place of the Q-function. These equations can be used to build directly the set of ECO-R with no need to pass through the approximation space of the Q-function.

Experiments extension

One of the main goals of CR-IRL is to overcome the need of the transition model in order to scale IRL to real-world applications. The experimental evaluation we proposed is based on classic benchmark problems. Even though they allow having full control on the different phases of the algorithm and interpreting effectively the results, we cannot avoid the comparison in more complex domains. We believe that an extension of the experimental evaluation is needed in order to consider real-world domains, such as Atari games. Furthermore, in those cases, we need to evaluate CR-IRL against suitable IRL methods, such as deep learning IRL approaches.

Reward transferability

The issue of reward transferability is substantial in any real-world application of IRL. This problem has been addressed in few works and only for simple problems, such as the Grid World and the simulated highway driving. In this thesis we have not evaluated the transferability properties of the reward functions recovered by CR-IRL. Intuitively, CR-IRL penalizes deviations from the expert's policy thus the recovered reward drives the learning process towards the expert's behavior. Thus it seems that no transferability is enforced by CR-IRL as the agent will tend to reproduce the expert's policy. Clearly, IRL methods that exploit handcrafted features, encoding implicitly similarities between states and actions, are more suitable to enforce tranferability. Nevertheless, if we consider very explorative experts, yielding trajectories in which almost all states and actions are visited, CR-IRL would assign a non-penalized reward even to suboptimal trajectories, producing a more transferable reward.

From reward to policy

The common thread of this thesis is the duality between the optimal policy and the reward function. We exercised this duality in the direction of IRL only: given the expert's (optimal) policy we recovered a reward function making the expert optimal. Most of the considerations made in this thesis hold also for the other direction. It is worth investigating whether the knowledge of the reward model can be exploited to restrict the class of policies to consider in order to determine the optimal one. Clearly, this goal lies outside the scope of IRL, but it might bring beneficial results in the context of policy search methods.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, page 1. ACM, 2004.

[2] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[3] Shun-ichi Amari and Scott C. Douglas. Why natural gradient? In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE international conference on*, volume 2, pages 1213–1216. IEEE, 1998.

[4] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[6] Julien Audiffren, Michal Valko, Alessandro Lazaric, and Mohammad Ghavamzadeh. Maximum entropy semi-supervised inverse reinforcement learning. In *IJCAI*, pages 3315–3321. AAAI Press, 2015.

[7] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L. Littman. Apprenticeship learning about multiple intentions. In *ICML*, pages 897–904, 2011.

[8] Leemon C Baird I. I. I. Advantage updating. Technical report, DTIC Document, 1993.

[9] Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[10] Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[11] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[12] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[13] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[14] Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[15] Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

[16] Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[17] Wendelin Böhmer, Steffen Grünewälder, Yun Shen, Marek Musial, and Klaus Obermayer. Construction of approximation spaces for reinforcement learning. *Journal of Machine Learning Research*, 14(1):2067–2118, 2013.

[18] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *AISTATS*, pages 182–189, 2011.

[19] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[20] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[21] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *AAMAS*, page 233. ACM, 2007.

[22] Jaedeug Choi and Kee-Eung Kim. Bayesian nonparametric feature construction for inverse reinforcement learning. In *IJCAI.* Citeseer, 2013.

[23] Fan R. K. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[24] Ronald R. Coifman, Stephane Lafon, Ann B. Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21):7426–7431, 2005.

[25] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[26] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1–2):1–142, 2013.

[27] Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *AAMAS*, pages 433–440. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[28] Dotan Di Castro and Shie Mannor. Adaptive bases for reinforcement learning. *Machine Learning and Knowledge Discovery in Databases*, pages 312–327, 2010.

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

[30] Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear Quadratic Control: An Introduction.* Krieger Publishing Co., Inc., Melbourne, FL, USA, 2000.

[31] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[32] Peter Englert and Marc Toussaint. Inverse kkt–learning cost functions of manipulation tasks from demonstrations. In *Proceedings of the International Symposium of Robotics Research*, 2015.

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

[34] Mahdi Milani Fard, Yuri Grinberg, Amir massoud Farahmand, Joelle Pineau, and Doina Precup. Bellman error based feature generation using random projections on sparse spaces. In *NIPS*, pages 3030–3038, 2013.

[35] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 49–58. JMLR.org, 2016.

[36] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.

[37] Thomas Furmston and David Barber. Variational methods for reinforcement learning. In *AISTATS*, volume 9, pages 241–248, 2010.

[38] Thomas Furmston and David Barber. A unifying perspective of parametric policy search methods for markov decision processes. In *NIPS*, pages 2717–2725, 2012.

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

[40] Peter W. Glynn. Likelilood ratio gradient estimation: an overview. In *Proceedings of the 19th conference on Winter simulation*, pages 366–375. ACM, 1987.

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

[42] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

[43] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[44] Marek Grzes and Daniel Kudenko. Learning shaping rewards in model-based reinforcement learning. In *AAMAS*, volume 115, 2009.

[45] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[46] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, pages 4565–4573, 2016.

[47] Jonathan Ho, Jayesh K. Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2760–2769. JMLR.org, 2016.

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

[49] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making-methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media, 2012.

[50] Tetsunari Inamura Masayuki Inaba Hirochika Inoue, M. Inamura, and H. Inaba. Acquisition of probabilistic behavior decision model based on the interactive teaching method. In *Proceedings of the Ninth International Conference on Advanced Robotics, ICAR99*, 1999.

[51] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In *NIPS*, pages 703–710. Morgan-Kaufmann, 1994.

[52] Edwin T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[53] Sham Kakade. Optimizing average reward using discounted rewards. In *International Conference on Computational Learning Theory*, pages 605–615. Springer, 2001.

[54] Sham Kakade. A natural policy gradient. In *NIPS*, pages 1531–1538. MIT; 1998, 2002.

[55] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1331–1336. IEEE, 2013.

[56] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *ICML*, pages 449–456. ACM, 2006.

[57] David A. Kendrick. *Stochastic control for economic models*. McGraw-Hill New York, 1981.

[58] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[59] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[60] Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In *NIPS*, pages 1007–1015. Curran Associates, Inc., 2012.

[61] Edouard Klein, Bilal Piot, Matthieu Geist, and Olivier Pietquin. A cascaded supervised learning approach to inverse reinforcement learning. In *ECML/PKDD (1)*, volume 8188 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013.

[62] Jens Kober and Jan R. Peters. Policy search for motor primitives in robotics. In *NIPS*, pages 849–856, 2009.

[63] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *AAAI*, 2011.

[64] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Feature construction for inverse reinforcement learning. In *NIPS*, pages 1342–1350. Curran Associates, Inc., 2010.

[65] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *NIPS*, pages 19–27, 2011.

[66] Hamid Reza Maei. *Gradient Temporal-difference Learning Algorithms*. PhD thesis, Edmonton, Alta., Canada, 2011. AAINR89455.

[67] Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *ICML*, pages 553–560. ACM, 2005.

[68] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

[69] Sridhar Mahadevan, Mauro Maggioni, Kimberly Ferguson, and Sarah Osentoski. Learning representation and control in continuous markov decision processes. In *AAAI*, volume 6, pages 1194–1199, 2006.

[70] Giorgio Manganini, Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Following newton direction in policy gradient with parameter exploration. In *IJCNN*, pages 1–8. IEEE, 2015.

[71] Yishay Mansour and Satinder Singh. On the complexity of policy iteration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 401–408. Morgan Kaufmann Publishers Inc., 1999.

[72] Peter Marbach and John N. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[73] Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *ICML*, pages 664–671. ACM, 2008.

[74] Emre Mengi, E. Alper Yildirim, and Mustafa Kilic. Numerical optimization of eigenvalues of hermitian matrix functions. *SIAM Journal on Matrix Analysis and Applications*, 35(2):699–724, 2014.

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

[76] Katja Mombaur, Anh Truong, and Jean-Paul Laumond. From human to humanoid locomotion-an inverse optimal control approach. *Autonomous robots*, 28(3):369–383, 2010.

[77] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, 1990.

[78] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[79] Gergely Neu and Csaba Szepesvári. Training parsers by inverse reinforcement learning. *Machine Learning*, 77(2-3):303–337, 2009.

[80] Andrew Y. Ng. *Shaping and policy search in reinforcement learning.* PhD thesis, University of California, Berkeley, 2003.

[81] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[82] Andrew Y. Ng, Stuart J. Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[83] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[84] Sarah Osentoski and Sridhar Mahadevan. Learning state-action basis functions for hierarchical mdps. In *ICML*, pages 705–712. ACM, 2007.

[85] Michael L. Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 9(2):256–268, 1988.

[86] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML*, pages 752–759. ACM, 2008.

[87] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *ICML*, pages 737–744. ACM, 2007.

[88] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, 2010.

[89] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[90] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. In *European Conference on Machine Learning*, pages 280–291. Springer, 2005.

[91] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008.

[92] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted and reward-regularized classification for apprenticeship learning. In *AAMAS*, pages 1249–1256. IFAAMAS/ACM, 2014.

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

[94] Matteo Pirotta and Marcello Restelli. Inverse reinforcement learning through policy gradient minimization. In *AAAI*, pages 1993–1999, 2016.

[95] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[96] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[97] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[98] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[99] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[100] Nathan Ratliff, David Bradley, J Andrew Bagnell, and Joel Chestnutt. Boosting structured prediction for imitation learning. In *NIPS*, pages 1153–1160. MIT Press, 2006.

[101] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 729–736. ACM, 2006.

[102] Nathan D. Ratliff, David Silver, and J. Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[103] Silvia Richter, Douglas Aberdeen, Jin Yu, et al. Natural actor-critic for road traffic optimisation. *NIPS*, 19:1169, 2007.

[104] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[105] Gavin A. Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems.* University of Cambridge, Department of Engineering, 1994.

[106] Claude Sammut, Scott Hurst, Dana Kedzier, Donald Michie, et al. Learning to fly. In *Proceedings of the ninth international workshop on Machine learning*, pages 385–393, 2014.

[107] Joe Saunders, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 118–125. ACM, 2006.

[108] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *NIPS*, pages 1038–1044, 1996.

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

[110] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063. The MIT Press, 1999.

[111] Richard S. Sutton, David A. McAllester, Satinder P. Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[112] Umar Syed, Michael H. Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 1032–1039. ACM, 2008.

[113] Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS*, pages 1449–1456, 2007.

[114] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[115] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *ICML*, pages 896–903. ACM, 2005.

[116] Jose M. Vidal and José M Vidal. Fundamentals of multiagent systems. 2006.

[117] Monica C. Vroman. *Maximum likelihood inverse reinforcement learning.* PhD thesis, Rutgers University-Graduate School-New Brunswick, 2014.

[118] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge England, 1989.

[119] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc., 2001.

[120] Eric Wiewiora, Garrison Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *ICML*, pages 792–799, 2003.

[121] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[122] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *CoRR*, 2015.

[123] Matthew D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[124] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *NIPS*, pages 262–270, 2011.

[125] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendix A

Matrix notation for finite Markov Decision Processes

In this Appendix we report the conventions we adopted for the matrix representation of the finite MDPs.

The transition model P is represented as a $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ stochastic matrix $\mathbf{P}$, such that:

$$P_{sas'} = P(s'|s,a), \qquad \forall s, s' \in \mathcal{S}, \quad a \in \mathcal{A}.$$

Sometimes it is convenient to think to the transition model as a $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ stochastic tensor P:

$$\mathsf{P}_{sas'} = P(s'|s,a), \qquad \forall s, s' \in \mathcal{S}, \quad a \in \mathcal{A}.$$

The policy of a MDP is represented as a $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ block-diagonal matrix $\boldsymbol{\pi}$, defined as:

$$\pi_{ss'a} = \begin{cases} \pi(a|s) & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases}, \qquad \forall s, s' \in \mathcal{S}, \quad a \in \mathcal{A},$$

$$\boldsymbol{\pi} = \begin{pmatrix} \pi(a_1|s_1) & \dots & \pi(a_{|\mathcal{A}|}|s_1) & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & \pi(a_1|s_{|\mathcal{S}|}) & \dots & \pi(a_{|\mathcal{A}|}|s_{|\mathcal{S}|}) \end{pmatrix}.$$

The distribution of the initial state is represented by a stochastic vector $\boldsymbol{\mu} \in [0,1]^{|\mathcal{S}|}$. The reward is represented by a vector $\mathbf{r} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. The state value function of policy π is a vector $\mathbf{v}^{\pi} \in \mathbb{R}^{|\mathcal{S}|}$, whereas the action value function and the advantage function are vectors $\mathbf{q}^{\pi}, \mathbf{a}^{\pi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. Furthermore, we denote with $\mathbf{P}^{\pi}$ the state transition kernel, i.e., the $|\mathcal{S}| \times |\mathcal{S}|$ stochastic matrix defined as:

$$\mathbf{P}^{\pi} = \boldsymbol{\pi}\mathbf{P}.$$

We indicate with $\mathbf{r}^{\pi} \in \mathbb{R}^{|\mathcal{S}|}$ the state reward, defined as:

$$\mathbf{r}^{\pi} = \boldsymbol{\pi}\mathbf{r}.$$

Using Neumann series, we can define the state occupancy in matrix form as:

$$\mathbf{d}_{\mu}^{\pi} = \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P}^{\pi}\right)^{t} \right)^{T} \boldsymbol{\mu} = \left(\mathbf{I} - \gamma \mathbf{P}^{\pi}\right)^{-T} \boldsymbol{\mu}. \tag{A.1}$$

Moreover, by unfolding the series we obtain a recursive equation:

$$\begin{aligned}
\mathbf{d}_{\mu}^{\pi} &= \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P}^{\pi}\right)^{t} \right)^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\mu} + \left(\sum_{t=1}^{+\infty} \left(\gamma \mathbf{P}^{\pi}\right)^{t} \right)^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\mu} + \gamma \left(\mathbf{P}^{\pi}\right)^{T} \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P}^{\pi}\right)^{t} \right)^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\mu} + \gamma \left(\mathbf{P}^{\pi}\right)^{T} \mathbf{d}_{\mu}^{\pi}.
\end{aligned} \tag{A.2}$$

Similarly, the state-action occupancy is defined as:

$$\boldsymbol{\delta}_{\mu}^{\pi} = \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P} \boldsymbol{\pi}\right)^{t} \right)^{T} \boldsymbol{\pi}^{T} \boldsymbol{\mu} = \left(\mathbf{I} - \gamma \mathbf{P} \boldsymbol{\pi}\right)^{-T} \boldsymbol{\pi}^{T} \boldsymbol{\mu}, \tag{A.3}$$

where $\boldsymbol{\pi}^{T} \boldsymbol{\mu}$ is a $|\mathcal{S}||\mathcal{A}|$-dimensional vector representing the state-action initial distribution, i.e., the probability to start the problem in state s performing action a. Like before, we can derive a recursive equation:

$$\begin{aligned}
\boldsymbol{\delta}_{\mu}^{\pi} &= \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P} \boldsymbol{\pi}\right)^{t} \right)^{T} \boldsymbol{\pi}^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\mu} + \left(\sum_{t=1}^{+\infty} \left(\gamma \mathbf{P} \boldsymbol{\pi}\right)^{t} \right)^{T} \boldsymbol{\pi}^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\pi}^{T} \boldsymbol{\mu} + \gamma \boldsymbol{\pi}^{T} \mathbf{P}^{T} \left(\sum_{t=0}^{+\infty} \left(\gamma \mathbf{P} \boldsymbol{\pi}\right)^{t} \right)^{T} \boldsymbol{\pi}^{T} \boldsymbol{\mu} = \\
&= \boldsymbol{\pi}^{T} \left(\boldsymbol{\mu} + \gamma \mathbf{P}^{T} \boldsymbol{\delta}_{\mu}^{\pi}\right).
\end{aligned} \tag{A.4}$$

Clearly, $\mathbf{d}_{\mu}^{\pi} = \boldsymbol{\pi} \boldsymbol{\delta}_{\mu}^{\pi}$. Now we can express the expected return in different ways:

$$J = \left(\mathbf{d}_{\mu}^{\pi}\right)^{T} \mathbf{r}^{\pi} = \left(\boldsymbol{\delta}_{\mu}^{\pi}\right)^{T} \mathbf{r} = \boldsymbol{\mu}^{T} \mathbf{v}^{\pi} = \boldsymbol{\mu}^{T} \boldsymbol{\pi} \mathbf{q}^{\pi}. \tag{A.5}$$

When considering parametric policies $\pi_{\boldsymbol{\theta}}$, the gradient of log policies $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$ are $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}| \times k$ tensors. However, for computational reasons they are typically treated as $|\mathcal{S}||\mathcal{A}| \times k$ matrices:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta} saj} = \frac{\partial \log \pi_{\boldsymbol{\theta}}(a|s)}{\partial \theta_j}, \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A}, \quad j = 1, 2, ..., k.$$

Appendix B

Proofs and derivations

This Appendix provides the proofs and derivations omitted in the thesis.

B.1 Kullback Leibler minimization and Maximum Likelihood estimation

We show that maximum likelihood estimation of the policy parameters is identical to the minimization of empirical Kullback-Leibler divergence between the expert's policy and the approximating parametric policy.

Proof. Let $\mathcal{D} = \{(s_{\tau_i,0}, a_{\tau_i,0}), ..., (s_{\tau_i,T(\tau_i)}, a_{\tau_i,T(\tau_i)})\}_{i=1}^N$ be a set of independent trajectories, collected with the expert's policy π^E. The likelihood function is given by:

$$\mathcal{L}(\boldsymbol{\theta}) = p(\tau_1, \tau_2, ..., \tau_N|\boldsymbol{\theta}) = \prod_{i=1}^N p_{\boldsymbol{\theta}}(\tau_i).$$

Taking the logarithm of previous equation we get:

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) &= \sum_{i=1}^M \log p_{\boldsymbol{\theta}}(\tau_i) = \\ &= \sum_{i=1}^M \left[\log \mu(s_{\tau_i,0}) + \sum_{t=0}^{T(\tau_i)-1} \log P(s_{\tau_i,t+1}|s_{\tau_i,t}, a_{\tau_i,t}) + \sum_{t=0}^{T(\tau_i)-1} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}) \right].\end{aligned}$$

The only terms depending on $\boldsymbol{\theta}$ are the $\log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})$, thus maximizing the log-likelihood is equivalent to maximizing only the quantity:

$$\sum_{i=1}^M \sum_{t=0}^{T(\tau_i)-1} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}).$$

Let us now consider the KL-div from samples between the expert's policy π^E and the approximating parametric policy $\pi_{\boldsymbol{\theta}}$:

$$\begin{aligned}
\hat{d}_{KL}(\pi^E, \pi_{\boldsymbol{\theta}}) &= \frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{T(\tau_i)-1} \log\left(\frac{\pi^E(a_{\tau_i,t}|s_{\tau_i,t})}{\pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})}\right) = \\
&= \frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{T(\tau_i)-1} \log \pi^E(a_{\tau_i,t}|s_{\tau_i,t}) - \frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{T(\tau_i)-1} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}).
\end{aligned}$$

As before, the only terms depending on $\boldsymbol{\theta}$ are the $\log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t})$. Thus, minimizing the KL-div corresponds to minimizing only:

$$-\frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{T(\tau_i)-1} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t}|s_{\tau_i,t}).$$

□

B.2 Mean and Variance of occupancy estimators

We derive the mean and the variance of the state occupancy estimator as defined in (4.9).

Proof. Let us consider a set of N independent trajectories:

$$\begin{aligned}
\mathbb{E}_\tau\left[\hat{d}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s)\right] &= \mathbb{E}_\tau\left[\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T(\tau_i)} \gamma^t \mathbb{1}(s_{\tau_i,t} = s)\right] = \\
&= \mathbb{E}_\tau\left[\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t} = s)\right] = \\
&= \sum_{t=0}^{+\infty} \gamma^t \mathbb{P}(s_t = s) = \\
&= d^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s),
\end{aligned} \tag{B.1}$$

where we exploited the identity $\mathbb{E}[\mathbb{1}(\omega)] = \mathbb{P}(\omega)$. Thus, the estimator is unbiased. Since the trajectories are independent we just need to compute the variance of one of them, as:

$$\mathrm{Var}_\tau\left[\hat{d}^{\pi_{\boldsymbol{\theta}}}_{\mu,\gamma}(s)\right] = \frac{1}{N}\mathrm{Var}_\tau\left[\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t} = s)\right].$$

We decompose the variance into:

$$\mathrm{Var}_\tau\left[\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t} = s)\right] = \mathbb{E}_\tau\left[\left(\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t} = s)\right)^2\right] - \mathbb{E}_\tau\left[\left(\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t} = s)\right)\right]^2. \tag{B.2}$$

We compute only the first term of the sum, as the second term has already been computed in equation (B.1):

$$\mathbb{E}_\tau\left[\left(\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t}=s)\right)^2\right] = \mathbb{E}_\tau\left[\sum_{t=0}^{T(\tau)} \gamma^{2t}\mathbb{1}(s_{\tau,t}=s)^2 + \right.$$
$$\left. + 2\sum_{t=0}^{T(\tau)}\sum_{t'=t+1}^{T(\tau)} \gamma^{t+t'}\mathbb{1}(s_{\tau,t}=s)\mathbb{1}(s_{\tau,t'}=s)\right].$$

We observe that $\mathbb{1}(s_{\tau,t}=s)^2 = \mathbb{1}(s_{\tau,t}=s)$ and $\mathbb{1}(s_{\tau,t}=s)\mathbb{1}(s_{\tau,t'}=s) = \mathbb{1}(s_{\tau,t}=s, s_{\tau,t'}=s)$, by taking the expectation we get:

$$\mathbb{E}_\tau\left[\left(\sum_{t=0}^{T(\tau)} \gamma^t \mathbb{1}(s_{\tau,t}=s)\right)^2\right] = \sum_{t=0}^{+\infty}\gamma^{2t}\mathbb{P}(s_t=s) + 2\sum_{t=0}^{+\infty}\sum_{t'=t+1}^{+\infty}\gamma^{t+t'}\mathbb{P}(s_t=s, s_{t'}=s). \quad \text{(B.3)}$$

By replacing it into equation (B.2):

$$\mathrm{Var}_\tau\left[\sum_{t=0}^{T(\tau)}\gamma^t\mathbb{1}(s_{\tau,t}=s)\right] = \sum_{t=0}^{+\infty}\gamma^{2t}\mathbb{P}(s_t=s) + 2\sum_{t=0}^{+\infty}\sum_{t'=t+1}^{+\infty}\gamma^{t+t'}\mathbb{P}(s_t=s,s_{t'}=s) +$$
$$-\left(\sum_{t=0}^{+\infty}\gamma^t\mathbb{P}(s_t=s)\right)^2 =$$
$$= \sum_{t=0}^{+\infty}\sum_{t'=t+1}^{+\infty}\gamma^{t+t'}\mathbb{P}(s_t=s,s_{t'}=s) - \mathbb{P}(s_t=s)\mathbb{P}(s_{t'}=s).$$

Finally the variance is given by:

$$\mathrm{Var}_\tau\left[\hat{d}_{\mu,\gamma}^{\pi_{\boldsymbol{\theta}}}(s)\right] = \frac{1}{N}\sum_{t=0}^{+\infty}\sum_{t'=t+1}^{+\infty}\gamma^{t+t'}\Big(\mathbb{P}(s_t=s,s_{t'}=s) - \mathbb{P}(s_t=s)\mathbb{P}(s_{t'}=s)\Big).$$

Therefore the estimator is consistent. □

B.3 Gradient and Hessian for ϵ-Boltzmann policy

We derive the gradient and Hessian for the case in which the features depend on the state-action pair $\boldsymbol{\zeta}_{sa}$ and the parameters are independent from states and actions $\boldsymbol{\theta}$. The case considered in the thesis (state-dependent features $\boldsymbol{\zeta}_s$ and action-dependent parameters $\boldsymbol{\theta}_a$) can be rephrased in this shape by considering the following matrix construction:

$$\boldsymbol{\theta} = \begin{pmatrix} \boldsymbol{\theta}_{a_1} \\ \boldsymbol{\theta}_{a_2} \\ \vdots \\ \boldsymbol{\theta}_{a_{|\mathcal{A}|}} \end{pmatrix} \in \mathbb{R}^{k|\mathcal{A}|}, \qquad \boldsymbol{\zeta} = \begin{pmatrix} \boldsymbol{\zeta}_{s_1}^T & \mathbf{0}^T & \dots & \mathbf{0}^T \\ \mathbf{0}^T & \boldsymbol{\zeta}_{s_1}^T & \dots & \mathbf{0}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}^T & \mathbf{0}^T & \dots & \boldsymbol{\zeta}_{s_1}^T \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\zeta}_{s_{|\mathcal{S}|}}^T & \mathbf{0}^T & \dots & \mathbf{0}^T \\ \mathbf{0}^T & \boldsymbol{\zeta}_{s_{|\mathcal{S}|}}^T & \dots & \mathbf{0}^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}^T & \mathbf{0}^T & \dots & \boldsymbol{\zeta}_{s_{|\mathcal{S}|}}^T \end{pmatrix} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times k|\mathcal{A}|}.$$

The ϵ-Boltzmann policy can be written as:

$$\pi_{\boldsymbol{\theta},\epsilon}(a|s) = (1-\epsilon)\frac{e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}} + \frac{\epsilon}{|\mathcal{A}|}$$

The gradient of the log ϵ-Boltzmann policy is given by:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta},\epsilon}(a|s) &= \nabla_{\boldsymbol{\theta}} \log \left((1-\epsilon)\frac{e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}} + \frac{\epsilon}{|\mathcal{A}|} \right) = \\ &= \nabla_{\boldsymbol{\theta}} \left(\log \left((1-\epsilon) e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}} + \frac{\epsilon}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}} \right) - \log \sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}} \right) = \\ &= \frac{(1-\epsilon)\boldsymbol{\zeta}_{sa} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}} + \frac{\epsilon}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} \boldsymbol{\zeta}_{sa} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}}{(1-\epsilon) e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}} + \frac{\epsilon}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}} - \frac{\sum_{a' \in \mathcal{A}} \boldsymbol{\zeta}_{sa'} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}}.\end{aligned} \tag{B.4}$$

In the remarkable case when $\epsilon = 0$ (Boltzmann policy) the gradient becomes:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta},0}(a|s) = \boldsymbol{\zeta}_{sa} - \frac{\sum_{a' \in \mathcal{A}} \boldsymbol{\zeta}_{sa'} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}}{\sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}}.$$

For sake of brevity we define the following symbols:

$$\begin{aligned} & Z_{\boldsymbol{\theta}}(s,a) = e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}}, \\ & Z_{\boldsymbol{\theta}}(s) = \sum_{a' \in \mathcal{A}} Z_{\boldsymbol{\theta}}(s,a') = \sum_{a' \in \mathcal{A}} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}, \\ & \nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\zeta}_{sa} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}}, \\ & \nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s) = \sum_{a' \in \mathcal{A}} \nabla_{\boldsymbol{\theta}}, Z_{\boldsymbol{\theta}}(s,a') = \sum_{a' \in \mathcal{A}} \boldsymbol{\zeta}_{sa'} e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}, \\ & \mathcal{H}_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s,a) = \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^T Z_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\zeta}_{sa} \boldsymbol{\zeta}_{sa}^T e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa}}, \\ & \mathcal{H}_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s) = \sum_{a' \in \mathcal{A}} \mathcal{H}_{\boldsymbol{\theta}}, Z_{\boldsymbol{\theta}}(s,a') = \sum_{a' \in \mathcal{A}} \boldsymbol{\zeta}_{sa'} \boldsymbol{\zeta}_{sa'}^T e^{\boldsymbol{\theta}^T\boldsymbol{\zeta}_{sa'}}, \\ & W_{\boldsymbol{\theta},\epsilon}(s,a) = (1-\epsilon) Z_{\boldsymbol{\theta}}(s,a) + \frac{\epsilon}{|\mathcal{A}|} Z_{\boldsymbol{\theta}}(s), \\ & \nabla_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a) = (1-\epsilon) \nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s,a) + \frac{\epsilon}{|\mathcal{A}|} \nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s), \end{aligned}$$

$$\mathcal{H}_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a) = (1-\epsilon)\mathcal{H}_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s,a) + \frac{\epsilon}{|\mathcal{A}|}\mathcal{H}_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s).$$

Thus the gradient (B.4) becomes:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta},\epsilon}(a|s) &= \frac{(1-\epsilon)\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s,a) + \frac{\epsilon}{|\mathcal{A}|}\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s)}{(1-\epsilon)Z_{\boldsymbol{\theta}}(s,a) + \frac{\epsilon}{|\mathcal{A}|}Z_{\boldsymbol{\theta}}(s)} - \frac{\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s)}{Z_{\boldsymbol{\theta}}(s)} \\ &= \frac{\nabla_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a)}{W_{\boldsymbol{\theta},\epsilon}(s,a)} - \frac{\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s)}{Z_{\boldsymbol{\theta}}(s)}.\end{aligned}$$

We now can compute the Hessian:

$$\begin{aligned}\mathcal{H}_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta},\epsilon}(a|s) &= \nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^T \log \pi_{\boldsymbol{\theta},\epsilon}(a|s) \\ &= \frac{\mathcal{H}_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a) W_{\boldsymbol{\theta},\epsilon}(s,a) - \nabla_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a)\nabla_{\boldsymbol{\theta}} W_{\boldsymbol{\theta},\epsilon}(s,a)^T}{W_{\boldsymbol{\theta},\epsilon}(s,a)^2} + \\ &- \frac{\mathcal{H}_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s) Z_{\boldsymbol{\theta}}(s) - \nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s)\nabla_{\boldsymbol{\theta}} Z_{\boldsymbol{\theta}}(s)^T}{Z_{\boldsymbol{\theta}}(s)^2}.\end{aligned} \tag{B.5}$$

B.4 Gradient and Hessian for LQG Gaussian policy

The LQG policy is a multivariate Gaussian policy in n dimensions having matrix $\mathbf{K}$ as parameter:

$$\pi_{\mathbf{K},\boldsymbol{\Sigma}}(\mathbf{a}|\mathbf{s}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\Big(-\frac{1}{2}(\mathbf{a}-\mathbf{Ks})^T \boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{Ks})\Big). \tag{B.6}$$

Using some matrix equalities provided in [91] we compute the gradient:

$$\begin{aligned}\nabla_{\mathbf{K}} \log \pi_{\mathbf{K},\boldsymbol{\Sigma}}(\mathbf{a}|\mathbf{s}) &= \nabla_{\mathbf{K}} \log \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} - \nabla_{\mathbf{K}}\Big(-\frac{1}{2}(\mathbf{a}-\mathbf{Ks})^T \boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{Ks})\Big) = \\ &= \boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{Ks})\mathbf{s}^T.\end{aligned}$$

In order to compute the Hessian, it is convenient to rewrite matrix $\mathbf{K}$ in terms of its components:

$$\mathbf{K} = \sum_{i=1}^n \sum_{j=1}^n K_{ij}\mathbf{J}^{ij},$$

where $\mathbf{J}^{ij}$ is the single-entry matrix with 1 at (i,j) and zero elsewhere. Now the gradient can be rewritten as:

$$\begin{aligned}\nabla_{\mathbf{K}} \log \pi_{\mathbf{K},\boldsymbol{\Sigma}}(\mathbf{a}|\mathbf{s}) &= \boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{Ks})\mathbf{s}^T = \\ &= \boldsymbol{\Sigma}^{-1}\mathbf{a}\mathbf{s}^T - \boldsymbol{\Sigma}^{-1}\mathbf{K}\mathbf{s}\mathbf{s}^T = \\ &= \boldsymbol{\Sigma}^{-1}\mathbf{a}\mathbf{s}^T - \boldsymbol{\Sigma}^{-1}\left(\sum_{i=1}^n \sum_{j=1}^n K_{ij}\mathbf{J}^{ij}\right)\mathbf{s}\mathbf{s}^T = \\ &= \boldsymbol{\Sigma}^{-1}\mathbf{a}\mathbf{s}^T - \sum_{i=1}^n \sum_{j=1}^n K_{ij}\boldsymbol{\Sigma}^{-1}\mathbf{J}^{ij}\mathbf{s}\mathbf{s}^T.\end{aligned}$$

Thus, the Hessian, which is a $n \times n \times n \times n$ tensor, can be computed element-wise as:

$$\left(\mathcal{H}_{\mathbf{K}} \log \pi_{\mathbf{K},\boldsymbol{\Sigma}}(\mathbf{a}|\mathbf{s})\right)_{ij} = -\boldsymbol{\Sigma}^{-1}\mathbf{J}^{ij}\mathbf{s}\mathbf{s}^T \qquad \forall i,j = 1,2,...,n. \tag{B.7}$$

B.4.1 One-dimensional case

When we consider the one-dimensional case ($n = 1$), the gradient and the Hessian are simply computed:

$$\begin{aligned}
&\pi_{k,\sigma^2}(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(a-ks)^2}{\sigma^2}},\\
&\nabla_k \log \pi_{k,\sigma^2}(a|s) = \nabla_k \log \frac{1}{\sqrt{2\pi\sigma^2}} - \nabla_k \frac{1}{2}\frac{(a-ks)^2}{\sigma^2} = \frac{s(a-ks)}{\sigma^2},\\
&\mathcal{H}_k \log \pi_{k,\sigma^2}(a|s) = \nabla_k \nabla_k^T \log \pi_{k,\sigma^2}(a|s) = \nabla_k \frac{s(a-ks)}{\sigma^2} = -\frac{s^2}{\sigma^2}.
\end{aligned}$$

B.5 Optimal control for the LQG

We outline the derivation of the optimal controller for the LQG and we solve it for the one-dimensional case, the complete derivation can be found in [57]. We aim to minimize the cost function:

$$\begin{aligned}
J(\mathbf{K}) &= \mathbb{E}_\tau \left[\mathbf{s}_{\tau,T}^T \mathbf{Q} \mathbf{s}_{\tau,T} + \sum_{t=0}^{T-1} \gamma^t \left(\mathbf{s}_{\tau,t}^T \mathbf{Q} \mathbf{s}_{\tau,t} + \mathbf{a}_{\tau,t}^T \mathbf{R} \mathbf{a}_{\tau,t} \right) \right] = \\
&= \mathbb{E}_\tau \left[\mathbf{s}_{\tau,T}^T \mathbf{Q} \mathbf{s}_{\tau,T} \right] + \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_\tau \left[\mathbf{s}_{\tau,t}^T \mathbf{Q} \mathbf{s}_{\tau,t} + \mathbf{a}_{\tau,t}^T \mathbf{R} \mathbf{a}_{\tau,t} \right],
\end{aligned}$$

where we considered a finite horizon T. If $\mathbf{Q}$ and $\mathbf{R}$ are positive semidefinite matrices, minimizing $J(\mathbf{K})$ corresponds to minimizing each term $J_t(\mathbf{K}) = \mathbf{s}_{\tau,t}^T \mathbf{Q} \mathbf{s}_{\tau,t} + \mathbf{a}_{\tau,t}^T \mathbf{R} \mathbf{a}_{\tau,t}$ for all time instants t. We restrict our analysis to the linear time-variant controllers of the form $\mathbf{a}_t = \mathbf{K}_t \mathbf{s}_t$. Following the derivation shown in [57], the optimal parameter $\mathbf{K}_t$ evolves according to equation:

$$\mathbf{K}_t = -\left(\mathbf{B}^T \mathbf{X}_{t+1} \mathbf{B} + \mathbf{R}\right)^{-1} \mathbf{B}^T \mathbf{X}_{t+1} \mathbf{A} \qquad t = T-1, ..., 0, \tag{B.8}$$

where $\mathbf{X}_t$ is determined by the following matrix Riccati difference equation (DRE) that runs backward in time, with starting condition $\mathbf{X}_T = \mathbf{Q}$:

$$\mathbf{X}_t = \mathbf{A}^T \mathbf{X}_{t+1} \mathbf{A} - \mathbf{A}^T \mathbf{X}_{t+1} \mathbf{B} \left(\mathbf{B}^T \mathbf{X}_{t+1} \mathbf{B} + \mathbf{R}\right)^{-1} \mathbf{B} \mathbf{X}_{t+1} \mathbf{A} + \mathbf{Q} \qquad t = T-1, ..., 0, \tag{B.9}$$

where we impose $\mathbf{X}_t$ to be symmetric positive definite. When the time horizon T tends to infinity the optimal controller becomes time invariant and the difference Riccati equation is replaced with the algebraic Riccati equation (ARE):

$$\mathbf{X} = \mathbf{A}^T \mathbf{X} \mathbf{A} - \mathbf{A}^T \mathbf{X} \mathbf{B} \left(\mathbf{B}^T \mathbf{X} \mathbf{B} + \mathbf{R}\right)^{-1} \mathbf{B} \mathbf{X} \mathbf{A} + \mathbf{Q}, \tag{B.10}$$

which yields the optimal steady-state controller with parameter $\mathbf{K} = -\big(\mathbf{B}^T\mathbf{X}\mathbf{B} + \mathbf{R}\big)^{-1}\mathbf{B}^T\mathbf{X}\mathbf{A}$. The solution of the ARE can be found by using iterative methods. For the one dimensional case we can determine the closed form solution.

B.5.1 Optimal LQG control in one dimension

The ARE in the one dimensional case becomes scalar:

$$x = a^2x - \frac{a^2b^2x^2}{b^2x + r} + q,$$

Since $x \geq 0$ and $r > 0$ the denominator never vanishes. Thus the equation is equivalent to finding the positive solutions of the second degree equation:

$$b^2x^2 + \big[(1-a^2)r - b^2q\big]x - qr = 0,$$

yielding the real solutions with opposite sign:

$$x = \frac{1}{2b^2}\Big(- \big[(1-a^2)r - b^2q\big] \pm \sqrt{\big[(1-a^2)r - b^2q\big]^2 + 4b^2qr}\Big).$$

We are interested in the positive solution only, which is the one obtained with the + sign. We can now compute the optimal parameter:

$$k = -\frac{bax}{b^2x + r} = \frac{1}{2abr}\Big(- \big[(a^2-1)r - b^2q\big] + \sqrt{\big[(1-a^2)r - b^2q\big]^2 + 4b^2qr}\Big).$$

In the case we considered in the experiments, i.e., $a = b = 1$ and $r = q = 0.9$, the parameter is given by $k = 1 - \varphi = \frac{1}{2}(1 - \sqrt{5}) \simeq -0.618$, where φ is the golden ratio.

B.6 Gradient and Hessian for Car on the Hill policy

The policy is a univariate Gaussian:

$$\pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(a-y(\mathbf{s}))^2}{\sigma^2}},$$

where the mean is defined as a radial basis function network:

$$y_{\mathbf{w}}(\mathbf{s}) = \sum_{i=1}^{k} w_i e^{-\delta\|\mathbf{s}-\mathbf{s}_i\|^2} = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{s}),$$

where $\phi_i(\mathbf{s}) = e^{-\delta\|\mathbf{s}-\mathbf{s}_i\|^2}$, for $i = 1, 2, ..., k$. We compute the gradient using the chain rule. We first derive:

$$\nabla_{y_{\mathbf{w}}(\mathbf{s})} \log \pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) = \nabla_{y_{\mathbf{w}}(\mathbf{s})} \log \frac{1}{\sqrt{2\pi\sigma^2}} - \nabla_{y_{\mathbf{w}}(\mathbf{s})} \frac{1}{2}\frac{(a - y_{\mathbf{w}}(\mathbf{s}))^2}{\sigma^2} = \frac{1}{\sigma^2}(a - y_{\mathbf{w}}(\mathbf{s})),$$

$$\nabla_{\mathbf{w}} y_{\mathbf{w}}(\mathbf{s}) = \nabla_{\mathbf{w}}\mathbf{w}^T\boldsymbol{\phi}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s}),$$

and now the gradient is:

$$\nabla_{\mathbf{w}} \log \pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) = \nabla_{y_{\mathbf{w}}(\mathbf{s})} \log \pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) \nabla_{\mathbf{w}} y_{\mathbf{w}}(\mathbf{s}) = \frac{1}{\sigma^2}(a - y_{\mathbf{w}}(\mathbf{s}))\boldsymbol{\phi}(\mathbf{s}).$$

We differentiate again to get the Hessian:

$$\begin{aligned}\mathcal{H}_{\mathbf{w}} \log \pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) &= \nabla_{\mathbf{w}} \nabla_{\mathbf{w}}^T \log \pi_{\mathbf{w},\sigma^2}(a|\mathbf{s}) = \\ &= \nabla_{\mathbf{w}} \frac{1}{\sigma^2}(a - y_{\mathbf{w}}(\mathbf{s}))\boldsymbol{\phi}(\mathbf{s})^T = \\ &= \frac{1}{\sigma^2}(a - y_{\mathbf{w}}(\mathbf{s}))\boldsymbol{\phi}(\mathbf{s})\boldsymbol{\phi}(\mathbf{s})^T.\end{aligned}$$

B.7 Closed form solution for ML parameters of Car on the Hill policy

The log-likelihood function, taken over N independent expert's trajectories, is given by:

$$\log \mathcal{L}(\mathbf{w}) = N \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)} (a_{\tau_i,t} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{s}_{\tau_i,t}))^2,$$

which is maximized as the following term is minimized:

$$\sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)} (a_{\tau_i,t} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{s}_{\tau_i,t}))^2.$$

This corresponds to a linear regression with $\boldsymbol{\phi}$ as basis functions. The solution can be found in closed form as:

$$\mathbf{w}^{\mathrm{ML}} = \left(\sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)} \boldsymbol{\phi}(\mathbf{s}_{\tau_i,t}) \boldsymbol{\phi}(\mathbf{s}_{\tau_i,t})^T \right)^{-1} \left(\sum_{i=1}^{N} \sum_{t=0}^{T(\tau_i)} \boldsymbol{\phi}(\mathbf{s}_{\tau_i,t}) a_{\tau_i,t} \right).$$

List of Acronyms

ADP Approximate Dynamic Programming. 8, 31

AI Artificial Intelligence. i, 1

AL Apprenticeship Learning. 2, 41, 42

ARE Algebraic Riccati Equation. 124, 125

BC Behavioral Cloning. 2, 41, 42, 44, 66, 83–85, 87, 101, 103, 104

BEBF Bellman Error Basis Functions. 29

BNP-FIRL Bayesian Non-Parametric Feature construction for Inverse Reinforcement Learning. 62

CR-IRL Compatible Reward Inverse Reinforcement Learning. ix, x, xx, xxiii, 4, 5, 66, 67, 78, 80, 81, 83–87, 89, 90, 95, 98, 99, 101, 102, 104–106

CSI Cascaded Supervised Inverse Reinforcement Learning. 58

DeepIRL Deep Inverse Reinforcement Learning. 55, 61

DP Dynamic Programming. 2, 7, 8, 20, 22–24

DQfD Deep Q-learning from Demonstrations. 61, 62

DRE Difference Riccati Equation. 124

ECO-Q Expert COmpatible Q features. xvi, 68, 69, 71–73, 78, 104, 105

ECO-R Expert COmpatible Reward features. xix, 72–78, 80, 81, 83–87, 104, 105

FD Finite Differences. 34

FIM Fisher Information Matrix. 39

FIRL Feature construction for Inverse Reinforcement Learning. 3, 4, 60, 61, 63

FQI Fitted Q-Iterations. xx, 99, 101, 105

GIRL Gradient Inverse Reinforcement Learning. 3, 58, 59, 63, 89, 90

GMM Gaussian Mixture Models. 43

GPIRL Gaussian Processes Inverse Reinforcement Learning. 61, 63

IRL Inverse Reinforcement Learning. xvi, 2–5, 41, 42, 45, 47, 51, 54–59, 61–63, 65–67, 73, 74, 81, 83, 84, 86, 90, 103–106

KKT Karush-Kuhn-Tucker. 58

KL Kullback-Leibler. xix, 45, 67, 68, 82, 88, 119, 120

KNN K-Nearest Neighbors. 30, 72, 90, 92, 93

LP Linear Programming. 22

LPAL Linear Programming Apprenticeship Learning. 51, 62, 83–87

LQG Linear Quadratic Gaussian regulator. xvii, xix–xxi, 82, 88, 89, 91–97, 123–125

LSMC Least Squares Monte Carlo. 27

LSMCQ Least Squares Monte Carlo Q. 28

LSPI Least Squares Policy Iteration. 28

LSTD Least Squares Temporal Difference. 27

LSTDQ Least Squares Temporal Difference Q. 28

MAP Maximum A Posteriori. 45

MC Monte Carlo, Markov Chain. 12, 23, 24, 27

MDP Markov Decision Process. xvi, xxiii, 1, 4–24, 26, 28–30, 32, 35–39, 41, 42, 45–47, 51, 53, 60, 68–72, 117

ME-IRL Maximum Entropy Inverse Reinforcement Learning. 56, 63, 83–86

ME-opt Maximum Eigenvalue Optimality criterion. 75, 94, 95

MESS-IRL Maximum Entropy Semi-Supervised Inverse Reinforcement Learning. 55

ML Maximum Likelihood. 45, 68, 94, 95

MLIRL Maximum Likelihood Inverse Reinforcement Learning. 56

MMP Maximum Margin Planning. 56, 57

MRP Markov Reward Process. 12

MWAL Multiplicative Weights for Apprenticeship Learning. 50, 51

NN Neural Network, Nearest Neighbors. 44, 90, 93

NR-IRL Ng-Russell Inverse Reinforcement Learning. 45, 47, 62

PGM Policy Gradient Methods. 32–34, 40

PGT Policy Gradient Theorem. 35, 36

PS Policy Search. 8, 31, 32

PVF Proto-Value Functions. xix, 9, 29, 30, 72, 86, 87

RBF Radial Basis Functions. 28

RE-IRL Relative Entropy Inverse Reinforcement Learning. 55

REPS Relative Entropy Policy Search. 55

RF REINFORCE. 35, 38

RL Reinforcement Learning. xv, 2, 3, 7, 8, 14, 18, 19, 23–25, 43, 48–50, 58, 59, 74, 80, 81, 103, 104

SARSA State Action Reward State Action. xxiii, 24–27

SCIRL Structured Classification based Inverse Reinforcement Learning. 56

SDP Semi-Definite Programming. 75–77, 79

SVD Singular Value Decomposition. 71, 74, 78, 80, 105

TD Temporal Difference. 23, 24, 27

TR-heu Trace Heuristic criterion. 76, 94, 95

TR-opt Trace Optimality criterion. 76, 94, 95

TV Total Variation. 45