GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

Doctoral Thesis

Enhancing Data Efficiency in Reinforcement Learning through Inverse and Transfer Reinforcement Learning

Ph.D. Program in Computer Science: XXXV cycle

Author:

Angelo Damiani
angelo.damiani@gssi.it

Supervisors:

Ph.D. Giorgio Manganini
giorgio.manganini@fraunhofer.it

Prof. Michele Flammini
michele.flammini@gssi.it

February 2024

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

Declaration of Authorship

I, Angelo Damiani, declare that this thesis titled, 'Enhancing Data Efficiency in Reinforcement Learning through Inverse and Transfer Reinforcement Learning' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at GSSI.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 08/02/2024

Abstract

In the Reinforcement Learning (RL) framework, an agent learns to achieve a goal facing a game-like situation in an uncertain and potentially complex environment. The agent gets either rewards or penalties for the actions it performs while aiming to maximize the total reward. Such trial-and-error approach is not always feasible in real applications: for instance, some mistake in the learning process can sometimes lead to critical issues or, in other cases, learning can result very slow, depending on the complexity of the problem at hand. Adopted techniques for these cases let the agent learn only using a set of samples collected from a desired behaviour and exploit it to improve the stability and efficiency of the learning process. This set of methods and algorithms belongs to the so-called Offline RL research area, which has recently attracted a lot of attention in the scientific community. One critical part of RL is the sample budget, i.e. the quantity of data available for learning. The concept of sample complexity arises in RL as a measure of how many samples or interactions an algorithm needs to learn an optimal or near-optimal policy and it is influenced by several factors, including the complexity of the environment, the agent's exploration strategy, and the algorithm's inherent learning capacity. Sample complexity plays a crucial role in determining the efficiency and practicality of RL algorithms. Being able to reduce it is an active area of research and a critical challenge for developing RL methods that can effectively learn in real-world, complex environments. In this thesis, we cope with this problem studying, developing and applying two algorithms in different sub-classes of Offline RL problems: Inverse Reinforcement Learning (IRL) and Transfer Learning (TL). While classic IRL methods only focuses on finding a reward function explaining the behaviour of a demonstrator's trajectories dataset, our first algorithm shows a novel formulation where we do not only accounts for the compatibility with the demonstrator behavior of the identified reward, but also for its effectiveness for the subsequent forward learning phase when the budget of samples is limited. While IRL algorithms are not designed to face the sample complexity budget, TL, in the context of Reinforcement Learning, aims to transfer knowledge from a source domain to a target one to improve the policy learning performances when the available data is limited. Our approach focuses on mapping datasets between domains exploiting specific features of different neural network architectures (GANs and Autoencoders). Using the feature extraction abilities of autoencoders along with the generative power of GANs we train an inter-task mapping to transform samples from a source domain to a target one (namely instance transfer) requiring only a small amount of data from the latter. This kind of sample transformation techniques can alleviate the challenge of sample complexity in RL compensating lack of samples from a task with transformed one from another. While modern transfer learning techniques primarily focus on transferring policies, our approach seeks to transfer samples. The objective is to train a precise transition model for the target domain, which can subsequently be utilized by any model-based control algorithm.

List of Publications

In the following, the list of the author's publications which some chapters of this doctoral dissertation are based on:

- **Balancing Sample Efficiency and Suboptimality in Inverse Reinforcement Learning**. Damiani, A.; Manganini, G.; Metelli, A. M.; and Restelli, M. In International Conference on Machine Learning, **ICML**, 2022
- **A Novel Inverse Reinforcement Learning Formulation for Sample-Aware Forward Learning**. Manganini, G.; Damiani, A.; Metelli, A.; and Restelli, M. In 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making, **RLDM**, 2022
- **Transfer Learning for Dynamical Systems Models via Autoencoders and GANs**. Damiani, A.; Viera Lopez, G.; Manganini, G.; Metelli, A.; and Restelli, M. Accepted at the 43th American Control Conference **ACC**, 2024

Acknowledgements

This doctoral dissertation, marking the conclusion of my Ph.D. studies, embodies not just the culmination of four years of personal dedication but also owes its fruition to the collaborative efforts and support of individuals to whom I express my gratitude.

I am sincerely grateful to my supervisor, Dr. Giorgio Manganini, for his invaluable support and patience throughout the last three years. His commitment to high standards and unwavering dedication have not only pushed me to achieve my best but have also significantly enhanced my capabilities in the field. Additionally, I extend my heartfelt gratitude to Professor Michele Flammini for his indispensable logistical support throughout my entire Ph.D. journey, as he has consistently served as a guiding presence in my academic endeavors from the very first day to the completion of this dissertation. Their support and mentorship have been instrumental in ensuring the smooth progress of my research and studies. Furthermore, I would like to express my appreciation to Prof. Marcello Restelli and Alberto Maria Metelli for introducing me to the topics of Reinforcement Learning and Inverse Reinforcement Learning, enhancing my understanding of these domains. A special acknowledgment goes to Gustavo, a friend and fellow traveler throughout the entire doctoral journey, with whom I shared moments of leisure and work until the very last days.

I extend my sincerest gratitude to all my friends, especially Mattia, Luca, Ilaria, Noemi and Giusy, whose unwavering support and companionship have been a constant source of encouragement throughout this academic pursuit. Their presence in both the challenging and relaxing moments has made the journey much more enjoyable and meaningful. Additionally, I am thankful to my gym partners Sandro, Antonio, Fabio, Michele, Dario and Chris for their camaraderie and shared dedication to fitness and well-being, offering both motivation and a pleasant distraction during my doctoral studies.

I would like to express my gratitude to my family, their enduring encouragement and the safe harbor they provide have been instrumental, granting me the strength to pursue my educational aspirations with confidence.

The most heartfelt thanks goes to Laura for the extraordinary moments she has brought into my life. I am forever indebted to the unconditional love, belief and support, she had demonstrated, especially during my most discouraging and darkest days.

One final acknowledgment goes to myself. For having remained stubborn and persevering even in those moments when everything seemed to be clueless and quitting appeared to be the easiest way out.

"And I knew exactly what to do. But in a much more real sense, I had no idea what to do."

Steve Carell as Michael Scott
The Office, Season 7, Episode 19

Contents

List of Figures

List of Acronyms

AC: Actor Critic

BIRL: Bayesian Inverse Reinforcement Learning

API: Approximate Policy Iteration

CP: Cart-Pole

DDPG: Deep Deterministic Policy Gradient

DPG: Deterministic Policy Gradient

DQN: Deep Q-Networks

GAN: Generative Adversarial Networks

GIRL: Gradient Inverse Reinforcement Learning

GPIRL: Gaussian Process Inverse Reinforcement Learning

IP: Inverted Pendulum

IRL: Inverse Reinforcement Learning

LSTDQ: Least-Squares Temporal Difference Q-learning

MC: Mountain Car

MDP: Markov Decision Process

MSE: Mean-Squared Error

MSPBE: Mean-Squared Projected Bellman Error

PAC: Probably Approximately Correct

PPO: Proximal Policy Optimization

RL: Reinforcement Learning

TD: Temporal Difference

TL: Transfer Learning

UMA: Unsupervised Manifold Alignment

Symbols

$\triangleq$	an equality derived from a definition
$\mathbb{R}$	set of scalar real numbers
$\mathbb{R}^d$	set of d-dimensional real vectors
$\mathcal{M}$	Markov decision process
$\mathcal{A}$	MDP's action space
$\mathcal{S}$	MDP's state space
$\mathcal{R}$	MDP's reward space
$P(s'\|s,a)$	MDP's state transition density
τ	trajectory
$\rho_{\boldsymbol{\eta}}(\cdot)$	probability distribution induced by parametric policy $\pi_{\boldsymbol{\eta}}$
$r(s)/r(s,a)$	MDP's reward function
$\boldsymbol{\phi}(s)/\boldsymbol{\phi}(s,a)$	reward features vector
$\boldsymbol{\theta}$	reward parameter vector
$d_{\boldsymbol{\theta}}$	reward parameter vector's dimension
$r_{\boldsymbol{\theta}}(s)/r_{\boldsymbol{\theta}}(s,a)$	MDP's parameterized reward function
$r(\tau)$	reward function over trajectory τ
$\mu(s)$	MDP's initial state distribution
γ	MDP's discount factor
$\pi(s)$	deterministic policy
$\pi(a\|s)$	stochastic policy
$\boldsymbol{\eta}$	policy parameter vector
$\Pi_{\boldsymbol{\eta}}$	parametric policy space
$\pi_{\boldsymbol{\eta}}(a\|s)$	parametric stochastic policy
$\mathcal{Q}$	state-action value function space
$\mathcal{G}[Q]$	space of greedy policies w.r.t. value function Q
$Q(s,a)$	generic state-action value function
$Q^\pi(s,a)$	state-action value function of policy π
$Q^*(s,a)$	optimal state-action value function

$\boldsymbol{\omega}$	state-action value function parameter vector
$d_{\boldsymbol{\omega}}$	state-action value function parameter vector's dimension
$\boldsymbol{\psi}(s,a)$	value function features vector
$Q_{\boldsymbol{\omega}}(s,a)$	state-action value function of parameter $\boldsymbol{\omega}$
$Q^{\pi}_{r,\gamma}(s,a)$	state-action value function induced by policy π, reward r and discount factor γ
$\mathcal{V}$	state value function space
$V(s)$	generic state value function
$V^{\pi}(s)$	state value function of policy π
$V^{*}(s)$	optimal state value function
T^{π}	Bellman operator of policy π
T^{*}	Bellman optimality operator
∇_x	gradient operator with respect to variable x
$\mathcal{F}$	generic function space
$\succeq$	component-wise vectorial inequality
ζ	sample complexity accuracy level
$\mathcal{D}$	generic dataset
$\boldsymbol{\varphi}(\pi)$	feature expectation induced by policy π
$\boldsymbol{\varphi}(\tau)$	feature count of trajectory τ
$W_p(\cdot,\cdot)$	L_p-Wasserstein Distance
M	Generic Inter-Task Mapping
M_x	Inter-Task Mapping from source to target for variable x
M_x^{-1}	Inter-Task Mapping from target to source for variable x
$\mathscr{M}$	Task space
$\mathcal{Z}$	Triplet space

Chapter 1

Introduction

When we think about "learning" in everyday life, the first thing that comes to mind is the process of acquiring knowledge through interactions and observations in our environment.

Let's imagine the case we've all been into: learning to drive a car. Learning to drive is a journey that requires individuals to develop a set of skills and make critical decisions on the road. As a novice driver, one must learn to process and analyze the *state* of the environment, assess potential risks, and choose appropriate *actions* to ensure safety. During the initial stages of learning, individuals begin by familiarizing themselves with the basic controls of a vehicle. They learn to make decisions such as when to accelerate, brake, and steer. These decisions are influenced by factors such as road conditions, traffic signals, and the behavior of other drivers. As the learning process progresses, novice drivers encounter various traffic scenarios and must make decisions in real-time. They face situations such as merging into traffic, navigating intersections, and choosing appropriate speeds. Decision-making in driving also involves managing potential risks and hazards. Novice drivers must learn to anticipate the actions of other drivers, identify potential dangers, and react accordingly. Over time, with practice and experience, novice drivers gain confidence and develop a more intuitive decision-making approach. They become adept at assessing situations quickly, making split-second decisions, and adjusting their driving behavior accordingly. This evolution reflects the growth of their decision-making skills and their ability to navigate the road with greater proficiency.

However, this online learning paradigm may not always be suitable in real-world applications; we all agree that is unsafe and dungerous to introduce a novice driver directly in a metropolis traffic on its first day of driving. In such cases, the concept of offline learning comes into play. Before stepping into a vehicle, individuals often undergo classroom training or study driving manuals to understand the principles and regulations governing road usage. They learn about traffic signals, right-of-way rules, and road markings, which form the foundation for their decision-making during driving. Additionally, learners engage in interactive simulations or virtual driving exercises that mimic real-world scenarios. These simulations expose them to a range of driving situations, such as navigating intersections, responding to hazards, and making critical decisions. Through these simulated experiences, they gain exposure to various driving challenges and practice decision-making in a controlled environment. Even without attending this kind of courses,

they can understand driving notions by observing and listening experienced drivers at work. Knowledge derived from this kinds of training helps the novice driver determine the feedback they receive from the surrounding environment and act appropriately. After acquiring theoretical knowledge and simulated practice (i.e. after learning offline how to drive), learners progress to practical driving sessions. In this setting, they apply what they have learned without immediate feedback (offline) to real-world situations (online). In this case the offline learning is ended and they proceeds to the refinement of their notions observing other drivers, understanding their decision-making processes, and internalizing their strategies.

What is described above is what happens when you are appropriately trained for the task. But, what will you do if you can't afford such a training and everything you can do is to observe experienced drivers? You must infer the underlying goals and intentions of experienced drivers by observing their actions and imitating their behaviors. During this learning process, novice driver pays attention to the subtle cues and signals exhibited by experienced drivers. They observe factors like the positioning of the vehicle, the timing of lane changes, the speed adjustments made, and the negotiation of right-of-way. By analyzing these observations, learners gradually build a mental model of the driving task and begin to **mimic the behaviors** they have witnessed.

On the other hand, let's consider a person who already knows how to drive a scooter and now wants to learn how to drive a car. The learning process of driving a car can be seen as a natural progression and adaptation of their existing driving skills from the scooter to the car. Having prior experience in driving another vehicle, the individual already possesses a foundation of driving skills and knowledge about traffic rules and regulations. This existing knowledge serves as a valuable starting point when transitioning to driving a car. The learner can leverage their familiarity with fundamental concepts such as acceleration, braking, and steering from their experience with the scooter. However, there are notable differences between driving a scooter and driving a car, such as the presence of a clutch, gear shifting, and the overall handling characteristics. The learner adapts their existing driving skills to account for these differences by understanding the specific controls and adjustments required for operating a car. For the same safety reason of above we can't let the driver directly learn on the street. Of course this novice driver would need less time in the driving school since they already know most of the basic street rules. This happens because they can **leverage pre-existent knowledge from a similar task**.

These basic human behaviors have long been a source of inspiration for the development of artificial intelligence and machine learning algorithms. In particular, the way humans learn from the interaction and observations and adapt their behaviors in new situations has motivated the exploration of various learning paradigms in the field of reinforcement learning. What is described above is the human version of what researchers call offline reinforcement learning (RL) and, in particular, inverse reinforcement learning (IRL) and transfer learning (TL). Through a comprehensive analysis of offline RL, including the exploration of state-of-the-art algorithms and methodologies, this thesis aims to contribute to the advancement of RL techniques proposing novel approaches in the aforementioned sub-areas of offline RL.

Reinforcement, Inverse Reinforcement and Transfer Learning: an overview

Reinforcement learning [18, 105] aims to enable computational agents to learn about cause-effect relationships by interacting with their environment and receiving feedback in the form of numerical rewards or punishments. The agent's objective is to discover a policy, a mapping from states to actions, that maximizes the cumulative reward over time. Nevertheless, the online learning paradigm may not always be applicable or optimal in practical scenarios. In such cases, the concept of offline reinforcement learning comes into play. Offline RL [69] is an extension of RL that focuses on learning from pre-collected datasets of past experiences. Instead of interacting with the environment in real-time, offline RL leverages these historical data to learn a policy. This approach is particularly useful in scenarios where online interactions are impractical, costly, or unsafe (e.g. the novice driver directly put inside the traffic of a city). By utilizing offline data (e.g. driving school and simulators), offline RL aims to optimize the agent's decision-making process without requiring real-time exploration and without relying on explicit knowledge of the underlying dynamics. The agent learns from the observed data, identifying patterns and making informed decisions based on the experiences captured in the dataset. The goal of offline RL is to bridge the gap between offline data and optimal decision-making, enabling agents to effectively leverage prior experiences to achieve high-performance policies. By learning from historical data, offline RL offers a practical and efficient approach for decision-making tasks where online interactions are not feasible or desirable. Current offline RL research managed to achieve extraordinary goals in areas such as autonomous driving [27, 101], healthcare [121] and robotics [49, 68, 88, 104]

Inverse Reinforcement Learning

Offline RL provides a powerful framework for learning from pre-collected datasets, leveraging historical experiences to improve decision-making. However, it assumes the availability of accurate reward signals in the dataset. In practice, obtaining such reward signals can be challenging and often requires expert knowledge or manual annotation. This reliance on accurate reward signals limits the scalability and applicability of offline RL, as it may not be feasible or cost-effective to collect such data for every possible application. Additionally, the availability of accurate reward signals may not always align with the goals and constraints of a given task or system, further complicating the use of offline RL in real-world scenarios. As a result, there is a pressing need for research and development in the field of inverse reinforcement learning (IRL) [84, 96] to address these limitations and offer a more flexible approach to learning from demonstrations without the strict requirement of precisely defined reward functions. IRL a technique that aims to infer the underlying reward structure from observed behavior (e.g. observations of experienced drivers). Instead of assuming that the reward function is known, IRL aims to estimate it by analyzing the demonstrated behavior of one or a group of experts. The fundamental idea behind IRL is that humans or experts, through their actions, implicitly encode a reward structure. By observing their behavior, IRL algorithms aim to recover this hidden reward function effectively reverse engineering the decision-making process of the expert. The inferred reward function obtained

through IRL can then be used within the RL framework to optimize decision-making policies. By integrating IRL, RL algorithms can broaden their horizons by acquiring knowledge from the deduced reward function, thus reducing their dependence on the environment's provided rewards, which should not be taken for granted. As a result, this can lead to enhanced learning efficiency, greater adaptability, and improved overall performance in various applications, ultimately underscoring the significance of IRL in the field of RL.

Transfer Learning in Offline RL

While IRL focuses on inferring reward functions, transfer learning (TL) [63, 109] broadens the scope to encompass various aspects of pre-learned knowledge, including features, representations, and even entire datasets. Just as IRL extracts underlying rewards from expert demonstrations, transfer learning identifies shared patterns, structures, and relationships across different tasks to facilitate knowledge transfer. By recognizing similarities between tasks, transfer learning enables the efficient utilization of previously acquired expertise, thus fostering a more streamlined and resource-efficient approach to solving novel problems (e.g. learning to drive a car already knowing how to drive a scooter). This not only accelerates the learning process but also optimizes resource allocation, potentially reducing the need for extensive data collection and computation. Consequently, transfer learning plays a pivotal role in advancing machine learning capabilities, making it an invaluable tool for various domains and applications.

Research Problem

Inverse reinforcement learning and transfer learning share common motivations to leverage existing knowledge or demonstrations to improve the performance and generalization of RL agents. The common thread lies in their shared goal of extracting and utilizing relevant knowledge from related tasks or expert demonstrations, enabling agents to expedite learning, enhance adaptability, and achieve improved performance in new and similar environments.

In this dissertation we want to tackle, through the application of novel IRL and TL approaches, a crucial question in the study of efficient RL algorithms: **sample complexity**. The sample complexity problem [48] constitutes a fundamental challenge that centers on determining the required number of interactions or experiences an RL agent must acquire from its environment to attain a proficient and effective policy. At its essence, sample complexity encapsulates the concept of data efficiency in RL. When an RL agent exhibits high sample complexity, it signifies that a substantial quantity of interactions or samples is essential before the agent can reach a performance level deemed satisfactory or practically applicable. In practical terms, this can translate to an agent requiring an impractical number of trials, attempts, or data points to master a task. Such a scenario poses a significant challenge in real-world applications, where protracted or resource-intensive learning processes are neither feasible nor desirable. For instance, in the domain of robotics, a robot grappling with high sample complexity may necessitate an extensive number of attempts to successfully accomplish a task, such as grasping an object. This not only prolongs the learning phase but also potentially entails wear and tear on the robot or, in extreme

cases, damage to the objects it interacts with. This sample complexity problem extends its influence across various other domains, encompassing autonomous systems, healthcare, finance, and beyond. Therefore, the central aspect of addressing the sample complexity problem in RL is to find ways to make the learning process more efficient, enabling RL agents to achieve competence with a reduced number of interactions.

This thesis's contributions on the sample complexity problem for RL is characterized by the development of IRL and TL techniques duly modified:

1. **A novel IRL algorithm taking into account the sample complexity of the reinforcement learning phase.** The concept of sample complexity has not been a primary concern in the field of IRL. While the sample complexity in RL is concerned with how many iterations or episodes are needed to converge to an optimal policy, in IRL, the sample complexity is related to the number of demonstrations or expert trajectories required to accurately estimate the reward function [54, 55, 66]. What we propose is a novel approach to the RL sample complexity problem through a IRL perspective. Our algorithm introduce an innovative formulation that goes beyond merely assessing the compatibility of the identified reward with the demonstrator's behavior, trading off its effectiveness in guiding subsequent forward learning phases and the amount of available data, particularly in scenarios where the sample budget is constrained.

2. **A novel deep transfer learning technique overcoming assumptions and limitations of state-of-the-art algorithms.** Differently from IRL, TL was born with the objecting of significantly alleviate the sample complexity of RL algorithms [21, 63, 109, 135]. By transferring learned representations or policies from a source task to a target one, RL agents benefit from pre-existing knowledge and experience, thus reducing the amount of data needed for effective learning in the target one. By adopting transfer learning, RL agents can leverage shared information and exploit similarities across tasks, ultimately improving sample efficiency and enabling faster convergence to optimal policies. In order to perform a transfer of knowledge across different domains (or tasks) it is required an inter-task mapping function. The majority of modern methods described in existing literature either maps a policy learned from source tasks from source tasks [131] or exploit it to guide the learning process in the target task [38, 129]. Other approaches have explored another form of knowledge transfer: transfer of samples (or instance transfer) [64, 114, 115]. However, all these methods are burdened by significant assumptions restricting their effectiveness in diverse contexts. For example policy transfer approaches depend on a previously learned policy in the source task while instance transfer methods require similar dynamics between tasks, same state and action spaces or high computational costs. In this thesis we propose a different instance transfer approach that overcome these limitation through the adoption of advanced supervised learning neural architectures like GANs [36] and autoencoders [57].

Structure of the thesis

This document is organized into chapters. Following this introduction, Chapter 2 will cover the foundational principles of reinforcement learning and sample complexity, while Chapter 3 will delve into the formulation of inverse reinforcement learning, including a comprehensive review of relevant literature. We will subsequently present our own IRL approach along with the corresponding numerical findings. Moving on to Chapter 4, we will introduce the concept of transfer learning and discuss the main algorithms in this field together with a brief overview on the performance metrics. Additionally, we will detail our instance transfer approach and share the numerical results. Conclusions and discussion on future avenues of research will follow.

Chapter 2

Reinforcement Learning

In this chapter, the key concepts of reinforcement learning will be introduced. As a first step the mathematical structure of the problem will be described and it will be followed by an overview on the main approaches of reinforcement learning with a taxonomy on its families of algorithms. After that, the sample complexity concept will be introduced, along with its quantification for some main RL algorithms. Finally, the offline RL problem will briefly introduced along with its associated challenges, and an overview of how this thesis aims to contribute to overcoming these challenges will be provided.

2.1 Introduction to Reinforcement Learning

In the quest to create intelligent machines that can learn and adapt, researchers have turned to a fascinating concept inspired by human learning processes: reinforcement learning (RL). Similar to how humans learn from experiences, RL agents acquire knowledge by actively engaging with their environments through a trial and error mechanism. Through a series of actions, these agents receive feedback in the form of rewards or penalties, providing them with valuable insights into the consequences of their decisions. Just like a human learns to avoid fire once he gets burn, this feedback-driven learning paradigm allows RL agents to optimize their behavior over time avoiding counterproductive actions during the achievement of their goal.

Thanks to its potential in solving complex problems and its similarities with human learning, reinforcement learning is rapidly growing in popularity within the artificial intelligence research community. While reinforcement learning shares striking similarities with human learning processes, it also exhibits notable dissimilarities compared to traditional supervised learning approaches. Unlike supervised learning, which relies on labeled data and explicit guidance, reinforcement learning agents usually learn from sparse and delayed rewards, making decisions based on long-term consequences. This unique characteristic enables RL to tackle complex decision-making problems where explicit instruction may be impractical or unavailable. Reinforcement learning has demonstrated its versatility and potential impact across a wide range of domains. In robotics, RL algorithms have been utilized to teach robots to perform intricate tasks with dexterity and

adaptability [53, 56, 133]. In finance, RL techniques have shown promise in optimizing investment strategies and portfolio management [33, 44–46, 71]. In healthcare, RL has been employed to develop personalized treatment plans and improve patient outcomes [20, 26, 52, 95, 121, 130]. These are just a few examples of the vast applicability of RL, and the potential for its utilization continues to expand.

How can we describe the reinforcement learning process? The core of RL is the learning agent, the entity interacting with the environment to solve a specific task or problem. As aforementioned, the RL agent operates within an environment and takes actions based on the observed states or observations. The goal of the agent is to maximize the cumulative reward it receives from the environment after applying actions and modifying its state. To achieve this, the agent employs a learning algorithm or policy that guides its decision-making process.

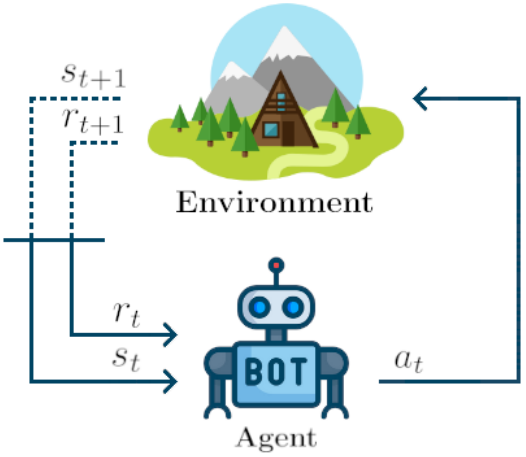


FIG. 2.1 Reinforcement learning framework. s_t and r_t respectively represents the environment's state at time t and the reward for the agent for time t after applying action a_{t-1} at time $t-1$.

Figure 2.1 shows an high-level view of the RL framework, while the mathematical meaning of environment, states, actions and policies will be part of the following section. This framework is deliberately described with an high level of abstraction in order to fit different kind of control and decision-making scenarios. For example we can consider the agent as a car driver, actions as the steering angle and pressure on the throttle, states can be composed by the position of the vehicle and fuel in the tank while the environment is the surrounding world. The reward can be tricky to be defined because it is something strictly related to the task the agent is going to learn. In the previous car context, a reward can be a positive value received after reaching a specific point or it can be a negative value in a specific point if the driver is wasting fuel or if it is late on a specific schedule.

Generally speaking, the reinforcement learning framework is able to generalize the problem of learning from interaction. It manages to reduce whatever the learner is described or able to perceive from the environment and how to interact with it as three signals passing back and forth between the agent itself and its surrounding: one signal to represent the actions the learner takes, one signal to represent what brought to this decision (the current state), and one signal to define the reward the learner takes after executing a given action.

2.1.1 Markov Decision Processes

The sequential decision-making process described in the introduction finds its mathematical formulation into Markov decision processes (MDP) [91]. MDPs are a form of mathematical modelling that allow a user to make decisions and formulate policies that optimize a given goal. At their core, MDPs are based on Markov chains, a type of probability model where the probability of a state (or outcome) depends solely on the current state. In other terms, this property, known as *Markov property*, encapsulates the memoryless nature of Markov Chains. A discrete-time Markov chain satisfies the Markov property if, for any positive integer t, and any states $s_0, s_1, \ldots, s_t$, the probability of transitioning to state s_t at time step t depends solely on the state s_{t-1} and is independent of any preceding state. More formally:

$$Pr(S_t = s_t | S_{t-1} = s_{t-1}, \ldots, S_0 = s_0) = Pr(S_t = s_t | S_{t-1} = s_{t-1}) \tag{2.1}$$

In contrast, MDPs take this concept one step further and allow the user to determine the probability of each state, given the current state and a set of actions.

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

- $\mathcal{S}$ is a measurable state space;
- $\mathcal{A}$ is a measurable action space;
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the Markovian state transition density $P(s'|s, a)$ defined for every triple (s', a, s);
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function;
- $\mu : \mathcal{S} \to [0, 1]$ is the initial state distribution;
- $\gamma \in [0, 1)$ is the discount factor.

State space $\mathcal{S}$ is the set of the key information or features capturing the current situation or configuration of the environment, and it can be discrete or continuous. A state provides relevant information for the RL agent to make decisions about the appropriate actions to take. States can vary in complexity and representation depending on the specific RL problem and domain. In some cases, they can be simple and directly observable, such as the position and velocity of a robotic arm, the pixel values of a game screen, or sensors data for vehicles. However, in many RL problems, states may be high-dimensional and include a large number of variables or features. For example, in autonomous driving, the state might consist of the current position, speed, orientation, and the surrounding objects' positions.

Action space $\mathcal{A}$ represents the set of choices or decisions the RL agent can take to interact with the environment and modify its state. Like the states, also actions can vary in nature and depend on the specific problem or domain being addressed. Similarly to the states, they can be discrete, where the agent chooses from a finite set of predefined options (for example, in a game-playing

scenario, the actions may correspond to moving left, right, up, or down) or continuous, where the agent can choose any value within a specific range. This is often the case in tasks that require fine-grained control or involve physical actions with continuous parameters (e.g. in autonomous driving, the actions may involve specifying the desired acceleration or steering angle).

Transition density function P reflects the probability to move from a state to another one upon taking an action. This concepts extend the aforementioned description of the Markov property. In this case, given a sequence of states and actions $s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t$, the probability of transitioning to state s_t depends solely on the state s_{t-1} and action a_{t-1}, thus rewriting equation 2.1 as:

$$P(s_t|s_{t-1}, a_{t-1}) \triangleq Pr(S_t = s_t|S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1} \ldots, S_0 = s_0, A_0 = a_0) \tag{2.2}$$

$$= Pr(S_t = s_t|S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}) \tag{2.3}$$

Reward function r produces a numerical value, usually decided by the learning engineers, the agent receives by applying an action in a given state or by reaching specific states. It is important to highlight that the reward is the key component leading to the resolution of a task, since it serves as feedback to guide the agent's learning process, indicating the desirability or quality of the agent's behavior.

Initial state distribution μ represents the probability distribution that characterizes the likelihood of an RL agent starting in a particular state at the beginning of an episode or task. In other words, μ describes the probability of each possible state being the initial state.

Finally, γ is a discount factor reducing the effect of future rewards with respect to the instant one. The discount factor determines the relative importance or weight given to immediate rewards versus delayed rewards. Intuitively, a discount factor closer to 1 indicates that the RL agent has a long-term perspective and values future rewards more highly. This implies that the agent is more patient and willing to sacrifice immediate rewards for greater long-term gains. On the other hand, a discount factor closer to 0 means that the agent focuses predominantly on immediate rewards, displaying a more myopic decision-making behavior.

Given this formal definition of the environment, we can introduce the concept of a policy π as the behaviour of the RL agent given the current configuration of the environment. This behaviour may be either stochastic, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(\cdot|s)$ denotes the probability function of the action argument "$\cdot$", or deterministic, $\pi : \mathcal{S} \to \mathcal{A}$.

The interaction between the agent and the environment starts at time $t = 0$ from a state $s_0 \sim \mu(s)$. The agent applies an action $a_0 \sim \pi(\cdot|s_0)$ using its policy π thus modifying the environment's state to $s_1 \sim P(\cdot|s_0, a_0)$ and receiving a reward $r_0 = r(s_0, a_0)$. An interaction with the environment (or a *transition*) can be summarized as a tuple (s_t, a_t, r_t, s_{t+1}), and we call *trajectory* of length T a sequence

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T). \tag{2.4}$$

The goal of RL is to find an optimal policy π^* that maximizes the cumulative expected return, for every initial state $s_0 \in S$. The expected infinite-horizon discounted return under a policy π is given by:

$$R^\pi(s_0) = \mathbb{E}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{2.5}$$

2.1.2 Value Functions

Value functions are essential components in the resolution of Markov Decision Processes. More specifically, they represent an estimation of the previously-defined expected discounted return that an agent obtains while starting from a given state (or state-action pair) and following a policy π.

Definition 2.2 (State-Action Value Function)**.** Let π be a policy, $s \in \mathcal{S}$ be any state, $a \in \mathcal{A}$ be any action. The *state-action value function* (or *Q-function*) of π in the state s while applying action a is given by:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s, a) + \gamma R^\pi(s') \right]. \tag{2.6}$$

Definition 2.3 (State Value Function)**.** Let π be a policy, $s \in \mathcal{S}$ be any state. The *state value function* (or *V-function*) of π in the state s is given by:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} \left[Q^\pi(s, a) \right] = R^\pi(s). \tag{2.7}$$

By using the Q-function and V-function we perform a *policy evaluation*, in the sense that we can assess the quality of the decision-making process through the expected return of an agent from a given starting state.

Definition 2.4 (Optimal Value Functions)**.** The optimal Q-function Q^* (V-function V^*) is defined as the best Q-function (V-function) that can be obtained by any policy π:

$$Q^*(s, a) = \max_\pi Q^\pi(s, a), \tag{2.8}$$

$$V^*(s) = \max_\pi V^\pi(s). \tag{2.9}$$

In terms of value function, any deterministic policy that selects at each state an action with the largest Q-value, i.e. satisfying:

$$\pi(s) \in \arg\max_{a \in \mathcal{A}} Q(s, a), \quad \forall s \in \mathcal{S}, \tag{2.10}$$

is called *greedy* in this Q-function. In particular, a deterministic policy π^* greedy in the optimal Q-function Q^* is optimal for the resolution of the MDP.

2.1.3 Bellman Operators

Value functions are characterized by the *Bellman equations*, i.e., a recursive representation of the value of a decision at a certain point as the payoff from some initial choices and the value of the remaining decision problem that results from those initial choice.

Definition 2.5 (Bellman equations)**.** Let π be any policy, then

$$Q^\pi(s,a) = \mathbb{E}_{\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}}\left[r(s,a) + \gamma Q^\pi(s',a')\right], \tag{2.11}$$

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right]. \tag{2.12}$$

Equations (2.11) and (2.12) can be expressed in a more convenient form by utilizing the concept of the Bellman operator.

Definition 2.6 (Bellman operators)**.** Let π be any policy, let $\mathcal{Q}$ be the set of all Q-functions, let $Q \in \mathcal{Q}$ be any state-action value function. We define the Bellman operator $T^\pi : \mathcal{Q} \to \mathcal{Q}$ for Q as:

$$(T^\pi Q)(s,a) \triangleq \mathbb{E}_{\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}}\left[r(s,a) + \gamma Q^\pi(s',a')\right]. \tag{2.13}$$

Analogously, let $\mathcal{V}$ be the set of all V-functions, let $V \in \mathcal{V}$ be any state value function. We define the Bellman operator $T^\pi : \mathcal{V} \to \mathcal{V}$ for V as:

$$(T^\pi V)(s) \triangleq \mathbb{E}_{\substack{s'\sim P(\cdot|s,a)\\ a\sim\pi(\cdot|s)}}\left[r(s,a) + \gamma V^\pi(s')\right]. \tag{2.14}$$

Given Definition 2.6, we can rewrite equations (2.11) and (2.12) in a more concise way as $T^\pi Q^\pi = Q^\pi$ and $T^\pi V^\pi = V^\pi$. A crucial characteristic of these operators is that, for $\gamma < 1$, they are γ-contractions with respect to the infinity norm, i.e., it can be proven that for any pair of functions $Q, Q' \in \mathcal{Q}$, it holds that:

$$||T^\pi Q - T^\pi Q'||_\infty \leq \gamma ||Q - Q'||_\infty. \tag{2.15}$$

Since T^π is a contraction, Q^π not only is a fixed-point, but it is also unique [42]. The same holds for state value-functions.

Similarly to the value functions induced by a given policy π, we can characterize optimal value functions Q^* and V^* through the so-called *Bellman optimality equations*:

Proposition 2.7 (Bellman Optimality Equations)**.** *Optimal value functions Q^* and V^* satisfy:*

$$Q^*(s,a) = \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[r(s,a) + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')\right], \tag{2.16}$$

$$V^*(s) = \max_{a\in\mathcal{A}} \mathbb{E}_{s'\sim P(\cdot|s,a)}\Big[r(s,a) + \gamma V^*(s')\Big]. \tag{2.17}$$

Analogously we can derive the respective Bellman optimality operators as:

Definition 2.8 (Bellman Optimality Operators)**.** Let $\mathcal{Q}$ be the set of all Q-functions, let $Q \in \mathcal{Q}$ be any state-action value function. We define the Bellman optimality operator $T^* : \mathcal{Q} \to \mathcal{Q}$ for

Q as:

$$(T^*Q)(s,a) \triangleq \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right]. \tag{2.18}$$

Similarly, let $\mathcal{V}$ be the set of all V-functions, let $V \in \mathcal{V}$ be any state value function. We define the Bellman operator $T^* : \mathcal{V} \to \mathcal{V}$ for V as:

$$(T^*V)(s) \triangleq \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim P(\cdot|s,a)} \Big[r(s,a) + \gamma V(s') \Big]. \tag{2.19}$$

Just like T^π, also T^* is a γ-contraction in the infinity norm, so, for any pair of functions $Q, Q' \in \mathcal{Q}$, it holds that:

$$||T^*Q - T^*Q'||_\infty \leq \gamma ||Q - Q'||_\infty. \tag{2.20}$$

This means that, if we consider $Q' \triangleq Q^*$ and repeatedly applying the operator T^* to Q we can ultimately converge to a fixed-point for T^*, which is Q^* itself. As we will see, Bellman optimality operators are the basis of value iteration algorithms that aim to directly find the optimal value function and derive, as described in section 2.1.2, the optimal policy through a greedy approach.

2.2 Taxonomy of Reinforcement Learning Algorithm

Exact methods vs approximate methods Exact methods, often associated with tabular methods, aim to find the precise solution to a reinforcement learning problem. The term "exact" refers to the strong assumption of having a complete knowledge of the MDP. In tabular methods, such as Q-iteration [105], an exhaustive table is maintained to store values representing the expected cumulative rewards associated with each state-action pair. This approach is particularly suited for environments with small, discrete state and action spaces. Exact methods guarantee convergence to the optimal policy given enough time and memory. In contrast, approximate methods seek efficient solutions for RL problems with larger, continuous, or high-dimensional state spaces. These methods, including deep reinforcement learning [8], tackle the curse of dimensionality (i.e. the problem of manipulating and storing values for large-dimensional problems) by employing function approximators, typically neural networks, to estimate value functions (e.g. $Q_{\boldsymbol{\omega}}(s,a) = f_Q(s,a;\boldsymbol{\omega})$) or policies (e.g. $\pi_{\boldsymbol{\eta}}(a|s) = f_\pi(a|s;\boldsymbol{\eta})$). The key distinction between exact and approximate methods lies in their computational demands and convergence properties. Exact methods require extensive memory and time resources, limiting their applicability in practice while approximate methods sacrifice the guarantee of exact optimality, offering scalability and adaptability.

Value-based vs policy-based methods Value-based algorithms primarily concentrate on computing or approximating a value function, typically without explicitly modeling a policy. For example, as mentioned in section 2.1.2, a possible approach is to determine the optimal action-value function, denoted as Q^*, and derive the optimal policy by greedily choosing actions from it. In contrast, policy-based methods directly explore the policy space, often without involving a value function. In this context, parametrized stochastic policies $\pi_{\boldsymbol{\eta}}(a|s)$ are employed, and the goal is to find optimal parameters that maximize the considered objective function, such as the expected return. There's also another family of algorithms which mixes these two classes:

actor-critic algorithms. This family of algorithms simultaneously learn a parametrized policy (*actor*) and a parametrized value function (*critic*). The actor search for improved returns, while the critic evaluates its performance, resulting in closely alternating learning processes for both components.

Model-based vs model-free methods Model-based RL relies on constructing an explicit model of the environment. This model includes information about the transition dynamics and the rewards. Model-based methods are sample-efficient, requiring fewer interactions with the environment; however, they demand an accurate model, which can be challenging to obtain in complex, real-world scenarios. Conversely, model-free RL learns directly from interaction without explicitly modeling the environment. Model-free methods are more robust to model inaccuracies but tend to require more data and may converge more slowly. Essentially, while model-based RL relies on a model to plan and make decisions, offering efficiency but necessitating an accurate model, model-free RL learns directly from experience, sacrificing some sample efficiency for greater robustness.

Online vs offline reinforcement learning In online RL each sample from the environment is collected with the current learning policy and it is directly employed to update the policy itself. In offline RL, instead, samples are collected from the environment by another policy (namely *behavioural policy*) and stored inside a buffer. The learning agent never interacts with the environment but exploits the dataset provided by the behavioural policy to optimize himself with respect to the task. Between online and offline RL stands the off-policy approach; in this context, the agent collects a batch of data from the environment with the current learning policy and stores it in a buffer. Once the collection is completed, the agent update its policy exploiting the collected batches of data. As we will see, some example of off-policy algorithm are deep Q-networks [80] and deep deterministic policy gradient [72], where the agent collects a bunch of trajectories before updating the policy using data collected from previous iterations. Figure 2.2 shows how the difference between the three classes of algorithms lies on the moment in which the training of the policies and the collection of the data happens.

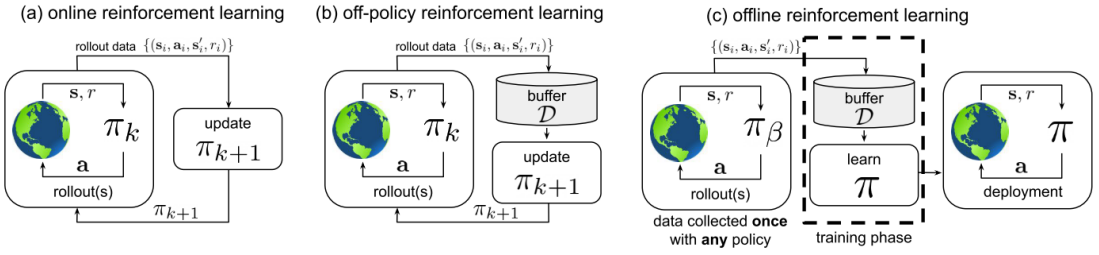


FIG. 2.2 [Taken from Levine et al. [69]]. In online RL (a), the policy π_k is updated with streaming data collected by π_k itself. In the classic off-policy setting (b), the agent's experience is appended to a data buffer $\mathcal{D}$, and each new policy π_k collects additional data, such that $\mathcal{D}$ is composed of samples from $\pi_0, \ldots, \pi_k$ and all of this data is used to train an updated new policy π_{k+1}. In contrast, offline RL employs a dataset $\mathcal{D}$ collected by some (potentially unknown) behavior policy π_β. The dataset is collected once, and is not altered during training, which makes it feasible to use large previous collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained.

Reinforcement learning research field is extensive, and not all approaches align with the focus of this thesis. In the following subsections, we will only delve into the principal methodologies of reinforcement learning, particularly those pertinent to this thesis.

2.2.1 Exact Methods

Exact methods, including policy iteration and value iteration, are fundamental techniques for solving the MDP control problem. *Policy iteration* iteratively refines a policy by alternating between *policy evaluation* steps, where the value function is computed for the current policy, and *policy improvement* steps, where actions are selected to maximize expected returns based on the value function. *Value iteration*, on the other hand, directly seeks the optimal value function by iteratively updating value functions according to the maximum achievable expected return in a single step. These methods provide a rigorous framework for finding optimal policies and are essential tools for understanding the core principles of RL.

2.2.1.1 Policy Iteration

Policy iteration [15] methods aim to iteratively build an optimal policy through a sequence of evaluation and improvement steps starting from an arbitrary initial policy π_0.

$$\pi_0 \xrightarrow{evaluation} Q^{\pi_0} \xrightarrow{improvement} \pi_1 \xrightarrow{evaluation} ... \xrightarrow{improvement} \pi^*$$

Policy evaluation step Let π be the policy to be evaluated and let's consider, for the sake of simplicity, our interest lies in the calculation of Q^π. Given our knowledge of the MDP, we can make use of the fixed-point representation of Q^π. This feature, combined with the contraction property of the Bellman operator T^π, enables us to employ the following iterative approach to derive a solution. We begin with an arbitrarily initialized state-action value function Q_0 (for each state and action pair) and, at each iteration step, we apply the Bellman operator to obtain the subsequent value function:

$$Q_{k+1} = T^\pi Q_k. \tag{2.21}$$

As mentioned in subsection 2.1.3, T^π is a γ-contraction in the infinity norm and so it has a unique fixed point $Q^\pi = T^\pi Q^\pi$. This means that, for any Q-functions Q and Q', it holds that:

$$||T^\pi(Q) - T^\pi(Q')||_\infty \leq \gamma ||Q - Q'||_\infty, \tag{2.22}$$

and, in particular:

$$\begin{aligned} ||Q_k - Q^\pi||_\infty = ||T^\pi Q_{k-1} - T^\pi Q^\pi||_\infty &\leq \gamma ||Q_{k-1} - Q^\pi||_\infty \\ &\leq ... \leq \gamma^k ||Q_0 - Q^\pi||_\infty. \end{aligned} \tag{2.23}$$

Equation (2.23) guarantees that as we continue with the iterations, we obtain a more accurate approximation of Q^π compared to the previous iterations. At this point one can utilize the infinity norm of the difference between the value functions calculated in two consecutive iterations, denoted as $||Q_k - Q_{k+1}||_\infty$, to evaluate the proximity of Q_k to Q^π. This, in return, can serve as a termination criterion to guarantee a desired level of accuracy.

Policy improvement step Once that, at a generic iteration k, the policy π_k is evaluated through a policy evaluation step, and a state-action value function is computed, we greedily

define with respect to Q^{π_k}:

$$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} Q^{\pi_k}(s,a), \quad \forall s \in \mathcal{S} \tag{2.24}$$

The *policy improvement theorem* assures that π_{k+1} is an improvement with respect to π_k.

Theorem 2.9 (Policy Improvement Theorem [105])**.** *Let π and π' be any pair of deterministic policies such that,*

$$Q^{\pi}(s,\pi'(s)) \geq Q^{\pi}(s,\pi(s)), \quad \forall s \in \mathcal{S} \tag{2.25}$$

Then π' must be better, or at least equal than, π:

$$Q^{\pi'}(s,\pi'(s)) \geq Q^{\pi}(s,\pi(s)), \quad \forall s \in \mathcal{S} \tag{2.26}$$

Essentially, through a sequence of evaluation and improvement step, the algorithm iteratively converge to an optimal policy π^* for each state of the MDP. When there's no policy update between consecutive iterations (i.e. $\pi_k = \pi_{k+1}$), the algorithm stops and π_k is returned. Algorithm 1 presents the pseudo-code of the policy iteration algorithm.

Algorithm 1 Policy iteration algorithm

1: Initialize policy π_0
2: **do** for $k = 0, 1, ...$
3: Iteratively compute Q^{π_k} for each (s,a) pair.
4: $\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} Q^{\pi_k}(s,a), \quad \forall s \in \mathcal{S}$
5: **while** $\pi_k \neq \pi_{k+1}$
6: return $\pi^* = \pi_k$, $Q^* = Q^{\pi_k}$

2.2.1.2 Value Iteration

Unlike policy iteration, which alternates between policy evaluation and policy improvement steps, value iteration focuses solely on updating the state-value function iteratively. This singular emphasis on the value function leads to several distinctions. Firstly, value iteration tends to converge faster, as it doesn't involve the potentially slower policy evaluation step present in policy iteration. Secondly, it is computationally more efficient and memory-friendly, especially when dealing with large state spaces, as it avoids storing an entire policy. It is still important to recall that in very large and/or continuous state and action spaces both these algorithms are practically unfeasible.

The idea behind value iteration algorithms is to exploit the Bellman optimality operator to iteratively improve the Q^* estimation. The algorithm starts with an arbitrarily initialized value function Q_0. At a certain iteration k we update the Q^* estimation as $Q_k = T^* Q_{k-1}$. By doing so, we obtain, by substituting Q with Q_k and Q' with Q^* in equation (2.20):

$$\begin{aligned} ||Q_k - Q^*||_\infty = ||T^\pi Q_{k-1} - T^\pi Q^*||_\infty &\leq \gamma ||Q_{k-1} - Q^*||_\infty \\ &\leq ... \leq \gamma^k ||Q_0 - Q^*||_\infty \end{aligned} \tag{2.27}$$

This means that at each iteration, Q_k is getting closer and closer to the optimal value function Q^*. Similarly to the policy evaluation step of policy-iteration, the algorithm can be stopped as $||Q_k - Q_{k+1}||_\infty$ reach a desired level of accuracy ϵ and a policy can be derived through a greedy approach on the last Q-function. Algorithm 2 shows the pseudocode for the Q-iteration algorithm. A similar algorithm can be also derived for the state value-function V^* counterpart.

Algorithm 2 Q-Iteration algorithm

1: Initialize Q-function Q_0
2: **do** for $k = 0, 1, ...$
3: **for** every $(s, a) \in \mathcal{S} \times \mathcal{A}$ **do**
4: $Q_{k+1}(s, a) \leftarrow \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q_k(s', a') \right]$
5: **end for**
6: **while** $||Q_{k+1} - Q_k||_\infty > \epsilon$
7: $Q^* \leftarrow Q_{k+1}$
8: $\pi^*(s) \in \arg\max_{a \in \mathcal{A}} Q^*(s, a), \quad \forall s \in \mathcal{S}$
9: return π^*, Q^*

2.2.2 Approximate Methods

Until now, we have operated under the assumption that our value function and policy estimates are structured as tables, with each state or state–action pair having its own entry. While this approach is straightforward and informative, it is constrained to scenarios involving a limited number of states and actions. The issue extends beyond merely the memory requirements for extensive tables; it also encompasses the substantial time and data demands necessary for precise table population. Moreover, we also assumed to have complete knowledge about the underlying MDP. In cases where the transition and reward models remain uncertain, exact computations of the Bellman operators become infeasible, necessitating the sampling of the MDP to gather statistical insights. In this subsection we introduce policy-based, value-based and actor critic approximate methods, particularly focusing on those algorithms involved in this thesis: LSPI [60], REINFORCE [126] and DDPG [72].

2.2.2.1 Policy-based methods

Approximate policy-based methods can be split into two main family of methods: approximate policy iteration (API) and policy search algorithms.

Approximate policy iteration algorithms In API, the traditional policy iteration process, consisting of policy evaluation and policy improvement, is adapted to work with function approximators like neural networks, making it highly scalable. In policy evaluation, instead of precisely estimating the value function, we use an approximation $Q_{\boldsymbol{\omega}}$ parameterized by $\boldsymbol{\omega}$. The policy is then updated in the policy improvement step to maximize the approximate value function.

An important example of API algorithm is **least square policy iteration** (LSPI) [60]. LSPI is an off-policy approach operating through an iterative process that includes data collection, value

function approximation, and policy improvement. The algorithm starts with an arbitrary initial policy π that interacts with the environment to collect data. Using this data, LSPI approximates Q^π through a linear function approximation $Q_{\boldsymbol{\omega}}(s,a) = \boldsymbol{\psi}(s,a)^\top \boldsymbol{\omega}$ where $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{d_{\boldsymbol{\omega}}}$ is a vector of linearly independent basis functions $\boldsymbol{\psi}(s,a) = (\psi_0(s,a), \psi_1(s,a), ..., \psi_{d_{\boldsymbol{\omega}}-1}(s,a))^\top$. To find the value of $\boldsymbol{\omega}$ let's assume to have a complete knowledge about the underlying MDP. Let's consider $\mathbf{P}$ the stochastic matrix of size $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ where $\mathbf{P}((s,a),s') = P(s'|s,a)$, $\boldsymbol{\Pi}$ the stochastic matrix of size $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ where $\boldsymbol{\Pi}(s',(s',a')) = \pi(a'|s')$, the vector $\mathcal{R}$ of size $|\mathcal{S}||\mathcal{A}|$, where $\mathcal{R}(s,a) = r(s,a)$ and lets define the matrix of basic functions $\boldsymbol{\Psi}$ of size $|\mathcal{S}||\mathcal{A}| \times d_{\boldsymbol{\omega}}$ as:

$$\boldsymbol{\Psi} = \begin{pmatrix} \boldsymbol{\psi}(s_1,a_1)^\top \\ ... \\ \boldsymbol{\psi}(s,a)^\top \\ ... \\ \boldsymbol{\psi}(s_{|\mathcal{S}|},a_{|\mathcal{A}|})^\top \end{pmatrix}. \tag{2.28}$$

By doing so we can rewrite the Bellman equation in a concise way, as:

$$Q^\pi = \mathcal{R} + \gamma \mathbf{P}\boldsymbol{\Pi} Q^\pi. \tag{2.29}$$

Now, what we want to do is to find a linear approximation $Q_{\boldsymbol{\omega}}$ that is a fixed point for the Bellman operator:

$$T_\pi Q_{\boldsymbol{\omega}} \approx Q_{\boldsymbol{\omega}}. \tag{2.30}$$

While we are sure that $Q_{\boldsymbol{\omega}}$ lies on the space spanned by the basis functions, we can't be sure $T_\pi Q_{\boldsymbol{\omega}}$ does the same. For this reason we apply an orthogonal projection $\boldsymbol{\Psi}(\boldsymbol{\Psi}^\top\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^\top$, thus obtaining:

$$\begin{aligned} \boldsymbol{\Psi}(\boldsymbol{\Psi}^\top\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^\top T_\pi Q_{\boldsymbol{\omega}} &= Q_{\boldsymbol{\omega}} \\ \boldsymbol{\Psi}(\boldsymbol{\Psi}^\top\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^\top(\mathcal{R} + \gamma\mathbf{P}\boldsymbol{\Pi} Q_{\boldsymbol{\omega}}) &= Q_{\boldsymbol{\omega}} \\ \boldsymbol{\Psi}(\boldsymbol{\Psi}^\top\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^\top(\mathcal{R} + \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Psi}\boldsymbol{\omega}) &= \boldsymbol{\Psi}\boldsymbol{\omega} \\ \underbrace{\boldsymbol{\Psi}^\top(\boldsymbol{\Psi} - \gamma\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Psi})}_{A}\boldsymbol{\omega} &= \underbrace{\boldsymbol{\Psi}\mathcal{R}}_{b} \\ \boldsymbol{\omega} &= A^{-1}b \end{aligned} \tag{2.31}$$

This linear approximation is the best we can do while knowing the MDP in its totality. What happens if our knowledge is limited to just n transitions (s_i, a_i, r_i, s_i') sampled by the aforementioned policy π? LSPI's policy evaluation algorithm (namely LSTDQ) make an empirical estimation of matrices A and b through the data collected from π as:

$$\hat{A} = \frac{1}{n}\hat{\boldsymbol{\Psi}}^\top(\hat{\boldsymbol{\Psi}} - \gamma\widehat{\mathbf{P}\boldsymbol{\Pi}\boldsymbol{\Psi}}), \tag{2.32}$$

$$\hat{b} = \frac{1}{n}\hat{\boldsymbol{\Psi}}^\top\hat{\mathcal{R}}, \tag{2.33}$$

where:

$$\hat{\mathbf{\Psi}} = \begin{pmatrix} \boldsymbol{\psi}(s_0, a_0)^\top \\ ... \\ \boldsymbol{\psi}(s_i, a_i)^\top \\ ... \\ \boldsymbol{\psi}(s_{L-1}, a_{L-1})^\top \end{pmatrix}, \quad \widehat{\mathbf{P\Pi\Psi}} = \begin{pmatrix} \boldsymbol{\psi}(s'_0, \pi(s'_0))^\top \\ ... \\ \boldsymbol{\psi}(s'_i, \pi(s'_i))^\top \\ ... \\ \boldsymbol{\psi}(s'_{L-1}, \pi(s'_{L-1}))^\top \end{pmatrix}, \quad \hat{\mathcal{R}} = \begin{pmatrix} r_0 \\ ... \\ r_i \\ ... \\ r_{L-1} \end{pmatrix}, \tag{2.34}$$

Summarizing, LSPI iteratively improves a policy repeating the following three steps: (i) collect n samples from the environment with its current policy π, (ii) make an estimation $Q_{\boldsymbol{\omega}}$ of the current policy's Q-function by using LSTDQ with policy π, (iii) improve π by greedily selecting actions on $Q_{\boldsymbol{\omega}}$. Similarly to the policy iteration exact algorithm, LSPI can stop when two policies (or two Q-function parameters) of consecutive iterations are close enough to each other.

Policy search algorithms Differently from API, policy search approaches have been introduced as approximate methods and lack an equivalent in the category of exact methods. The central element of a policy search approach involves representing the policy using an approximation framework and subsequently seeking the optimal policy according to some optimality criterion. Policy search approaches encompasses a broad spectrum of techniques, with notable distinctions between gradient-free and gradient-based approaches.

Gradient-free methods aim to optimize policies without directly computing gradients of the expected return. Instead, they often employ black-box optimization techniques like particle swarm [41] or evolutionary strategies [39]. These methods explore the policy space through trial-and-error, evaluating policies and selecting the best-performing ones.

Gradient-based methods focus on the idea of calculating the gradient of the expected return with respect to the policy parameters (also assuming it is differentiable), and then employing this gradient to iteratively update the policy through gradient ascent thus finding parameters leading to maximal returns. Let $\Pi_\eta = \{\pi_{\boldsymbol{\eta}} : \boldsymbol{\eta} \in \mathbb{R}^d\}$ be a class of parameterized policies where d is a positive integer. Let's now consider a trajectory τ (defined as in equation (2.4)) and, with an abuse of notation, let's define the total reward over τ like:

$$r(\tau) \triangleq \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \tag{2.35}$$

A policy $\pi_{\boldsymbol{\eta}}$ then induces a probability distribution $\rho_{\boldsymbol{\eta}}(\tau)$ over trajectories reflecting the probability of obtaining a given trajectory, given an initial state distribution μ and following $\pi_{\boldsymbol{\eta}}$:

$$\rho_{\boldsymbol{\eta}}(\tau) \triangleq \mu(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi_{\boldsymbol{\eta}}(a_t|s_t). \tag{2.36}$$

The expected return $J(\boldsymbol{\eta})$ of $\pi_{\boldsymbol{\eta}}$ over all possible trajectories can be defined as:

$$J(\boldsymbol{\eta}) \triangleq \int_\tau r(\tau)\rho_{\boldsymbol{\eta}}(\tau). \tag{2.37}$$

Given this premise, starting from an arbitrary parameter $\boldsymbol{\eta}_0$, a policy gradient method focuses on iteratively updating the policy by gradient ascent:

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \alpha \nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}), \tag{2.38}$$

where α is the learning rate, and $\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta})$ represent the gradient of the expected return with respect to the policy parameter $\boldsymbol{\eta}$.

Due to the unknown state transition density of the underlying MDP, the precise computation of the policy gradient, much like expected returns, can't be analytically computed. **REINFORCE** algorithm [126] solves this problem by observing that:

$$\nabla_{\boldsymbol{\eta}} \rho_{\boldsymbol{\eta}}(\tau) = \rho_{\boldsymbol{\eta}}(\tau) \nabla_{\boldsymbol{\eta}} \log \rho_{\boldsymbol{\eta}}(\tau). \tag{2.39}$$

Thanks to equation (2.39) we can derive:

$$\begin{aligned} \nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) &= \nabla_{\boldsymbol{\eta}} \int_{\tau} r(\tau) \rho_{\boldsymbol{\eta}}(\tau) = \int_{\tau} r(\tau) \nabla_{\boldsymbol{\eta}} \rho_{\boldsymbol{\eta}}(\tau) \\ &= \int_{\tau} r(\tau) \rho_{\boldsymbol{\eta}}(\tau) \nabla_{\boldsymbol{\eta}} \log \rho_{\boldsymbol{\eta}}(\tau) \\ &= \mathbb{E}_{\tau \sim \rho_{\boldsymbol{\eta}}} \Big[r(\tau) \nabla_{\boldsymbol{\eta}} \log \rho_{\boldsymbol{\eta}}(\tau) \Big]. \end{aligned} \tag{2.40}$$

We can observe, by applying gradient and logarithm to equation (2.36), that:

$$\begin{aligned} \nabla_{\boldsymbol{\eta}} \log \rho_{\boldsymbol{\eta}}(\tau) &= \nabla_{\boldsymbol{\eta}} \Big(\log \mu(s_0) + \sum_{t=0}^{T-1} \log P(s_{t+1}|s_t, a_t) + \log \pi_{\boldsymbol{\eta}}(a_t|s_t) \Big) \\ &= \nabla_{\boldsymbol{\eta}} \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\eta}}(a_t|s_t) \\ &= \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\eta}} \log \pi_{\boldsymbol{\eta}}(a_t|s_t). \end{aligned} \tag{2.41}$$

Finally plugging equation (2.41) into equation (2.40) we obtain:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \mathbb{E}_{\tau \sim \rho_{\boldsymbol{\eta}}} \Big[r(\tau) \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\eta}} \log \pi_{\boldsymbol{\eta}}(a_t|s_t) \Big]. \tag{2.42}$$

As obtaining an analytical computation for equation (2.42) is typically impractical, the gradient can be approximated by averaging n i.i.d. trajectories collected under policy $\pi_{\boldsymbol{\eta}}$:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) \approx \frac{1}{n} \sum_{i=0}^{n-1} \sum_{t=0}^{T-1} r(\tau_i) \nabla_{\boldsymbol{\eta}} \log \pi_{\boldsymbol{\eta}}(a_{i,t}|s_{i,t}), \tag{2.43}$$

where $\tau_i = (s_{i,0}, a_{i,0}, r_{i,0}, s_{i,1}, ..., s_{i,T-1})$ is the i-th trajectory.

2.2.2.2 Value Based methods

The central concept of approximate value-based algorithms, such as Q-learning [123] and Deep Q-Networks (DQN) [80], involves expressing value functions within a functional space denoted as $\mathcal{F} = \{Q_{\boldsymbol{\omega}} : \boldsymbol{\omega} \in \mathbb{R}^{d_{\boldsymbol{\omega}}}\}$. This approach offers a significant advantage, as it facilitates the efficient handling of large and/or infinite state and action spaces when an appropriate $\mathcal{F}$ is chosen. The primary objective is to identify a function, represented as $f \in \mathcal{F}$, that effectively approximates the optimal value function Q^* or V^*.

Q-learning Most of approximate Q-learning algorithms are based on Temporal-Difference (TD) learning [105]. Instead of waiting the end of trajectories (as it is needed for Montecarlo estimation methods like REINFORCE), TD algorithms allows updates after each transition minimizing the discrepancy between the predicted value of the state-action pair and the sum between the received reward and the estimated value of the next state-action pair. Q-learning can be defined also in its tabular version; in this case, all state-action entries are updated following the rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{2.44}$$

In its, more popular, approximate version, Q-learning algorithms update the approximate state-action value function's parameter $\boldsymbol{\omega}$, assuming to be differentiable, in a gradient descent manner:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha(r_t + \gamma \max_{a \in A} Q_{\boldsymbol{\omega}}(s_{t+1}, a) - Q_{\boldsymbol{\omega}}(s_t, a_t))\nabla_{\boldsymbol{\omega}} Q_{\boldsymbol{\omega}}(s_t, a_t), \tag{2.45}$$

where α, in both equations, represents the learning rate.

Deep Q-Networks DQN [80] brings the Q-learning approach to a next step mixing the TD methodology with deep learning. While neural networks offer considerable representational capacity, their integration with an algorithm like Q-learning introduces numerous issues, which require the introduction of some deep learning techniques to stabilize the learning process. The approximate Q-learning's process of minimizing the mean TD error resembles a supervised learning problem, except for the fact that samples are not independent and identically distributed due to the sequential correlation of states. To address this issue, a DQN agent stores all observed transitions in a replay buffer $\mathcal{D}$. During each step of training, a subset of n transitions (s_i, a_i, r_i, s'_i) is uniformly sampled from this buffer and used to estimate the average squared TD error:

$$L(\boldsymbol{\omega}) = \frac{1}{n} \sum_{i=0}^{n-1} (r_i + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\omega}'}(s'_i, a') - Q_{\boldsymbol{\omega}}(s_i, a_i))^2, \tag{2.46}$$

thus deriving the gradient

$$\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}) = -\frac{1}{n} \sum_{i=0}^{n-1} \Big(r_i + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\omega}'}(s'_i, a') - Q_{\boldsymbol{\omega}}(s_i, a_i)\Big) \nabla_{\boldsymbol{\omega}} Q_{\boldsymbol{\omega}}(s_i, a_i). \tag{2.47}$$

Note that equations (2.47) and (2.46) depends on two parameters, namely $\boldsymbol{\omega}$ and $\boldsymbol{\omega}'$. The Q-network $Q_{\boldsymbol{\omega}'}$ is known as *target network*. At its core $Q_{\boldsymbol{\omega}'}$ is a delayed replica of the Q-network that provide more consistent and stable target Q-values preventing rapid changes during the training ultimately stabilizing the learning. By updating the target network's parameters at

a slower rate than the Q-network $Q_{\boldsymbol{\omega}}$, DQN mitigates issues related to moving target values, thereby improving the convergence and stability of the Q-learning process.

Once the Q estimation step is computed, new data must be collected and stored in $\mathcal{D}$. Usually this data collection is performed via an ϵ-greedy policy π_t, i.e.:

$$\pi_t(s) = \begin{cases} a \in \arg\max_{a' \in \mathcal{A}} Q_{\boldsymbol{\omega}}(s, a') & \text{with probability } 1 - \epsilon \\ a \sim \mathcal{U}_{\mathcal{A}} & \text{with probability } \epsilon \end{cases} \tag{2.48}$$

where $\mathcal{U}_{\mathcal{A}}$ denote the uniform distribution over the action space.

2.2.2.3 Actor critic methods

Actor-Critic (AC) algorithms [37] constitute a significant class of TD reinforcement learning techniques that combine the strengths of both policy-based and value-based methods. In these algorithms, an agent employs two essential components: the *actor*, responsible for selecting actions, and the *critic*, tasked with estimating the value of states or state-action pairs. The actor is determined by the agent's parameterized policy $\pi_{\boldsymbol{\eta}}$. Meanwhile, the critic is a parametrized value function ($Q_{\boldsymbol{\omega}}$ or $V_{\boldsymbol{\omega}}$) which evaluates the current agent's policy.

AC methods share a similarity with policy iteration, as both enhance policies based on value functions. However, the key distinction lies in their approach. In policy iteration, the improved policy is entirely greedy with respect to the value function, fully maximizing it over action variables. Conversely, AC methods employ gradient-based rules for policy updates. But differently from policy gradient methods, like the aforementioned REINFORCE, where the expected return is evaluated through an empirical estimation of the policy value function (equation (2.37)), in AC this long-run evaluation is performed via a parametric one.

Let's recall, for example, equation (2.37). The same equation can be rewritten as:

$$J(\boldsymbol{\eta}) = \int_{\mathcal{S}} \rho_{\boldsymbol{\eta}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\eta}}(a|s) r(s, a) da ds, \tag{2.49}$$

where $\rho_{\boldsymbol{\eta}}(s) = \lim_{t \to \infty} Pr(s_t = s | \mu, \pi_{\boldsymbol{\eta}})$ is the state distribution induced by policy $\pi_{\boldsymbol{\eta}}$ and initial state distribution μ. The *policy gradient theorem* [106] proved that:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \mathbb{E}_{s \sim \rho_{\boldsymbol{\eta}}} \Big[\int_{\mathcal{A}} \nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(a|s) Q^{\pi_{\boldsymbol{\eta}}}(s, a) da \Big]. \tag{2.50}$$

The problem here is to determine $Q^{\pi_{\boldsymbol{\eta}}}$. AC algorithms, instead of the unknown action-value function $Q^{\pi_{\boldsymbol{\eta}}}$, adopts an approximation $Q_{\boldsymbol{\omega}}$ of parameter $\boldsymbol{\omega}$:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \mathbb{E}_{s \sim \rho_{\boldsymbol{\eta}}} \Big[\int_{\mathcal{A}} \nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(a|s) Q_{\boldsymbol{\omega}}(s, a) da \Big]. \tag{2.51}$$

Regarding the critic, any update algorithm can be used. For example employing Q-learning TD error update after collecting a transition (s, a, r, s'):

$$\delta_t = r(s,a) + \gamma Q_{\boldsymbol{\omega}}(s', a') - Q_{\boldsymbol{\omega}}(s,a), \tag{2.52}$$

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_{\boldsymbol{\omega}} \delta_t \nabla_{\boldsymbol{\omega}} Q_{\boldsymbol{\omega}}(s,a), \tag{2.53}$$

where a' is not the greedy policy like in equation (2.45), but $a' \sim \pi_{\boldsymbol{\eta}}(\cdot|s')$.

At its core, a generic AC algorithm's learning loop is structured as a sequence of actor and critics updates via gradients estimation and interleaved dependencies. It is important to highlights how AC algorithms can be online or off-policy. In the former case transitions are observed in real-time and used for updates. In the latter, the algorithm iterates through a replay buffer $\mathcal{D}$ (made of samples obtained from previous iterations policies) to update actor and critic parameters before collecting additional data.

Deterministic policy gradient In Deterministic policy gradient (DPG) , the policy is represented as a *deterministic* parametric function $\pi_{\boldsymbol{\eta}}$, which maps states to continuous actions. DPG uses the *deterministic policy gradient theorem*[102] to rewrite the gradient of the expected return of equation (2.51) as:

$$\begin{aligned} \nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) &= \int_{\mathcal{S}} \rho_{\boldsymbol{\eta}}(s) \nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(s) \nabla_a Q_{\boldsymbol{\omega}}(s,a)|_{a=\pi_{\boldsymbol{\eta}}(s)} ds \\ &= \mathbb{E}_{s\sim\rho_{\boldsymbol{\eta}}} \Big[\nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(s) \nabla_a Q_{\boldsymbol{\omega}}(s,a)|_{a=\pi_{\boldsymbol{\eta}}(s)} \Big]. \end{aligned} \tag{2.54}$$

It is important to observe that, in practical scenarios, the stationary distribution $\rho_{\boldsymbol{\eta}}$ is replaced with the state visitation frequency. Consequently, for single-step cases, we can derive the subsequent update rules for both the actor and critic:

$$\delta_t = r_t + \gamma Q_{\boldsymbol{\omega}}(s'_t, \pi_{\boldsymbol{\eta}}(s'_t)) - Q_{\boldsymbol{\omega}}(s_t, a_t), \tag{2.55}$$

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_{\boldsymbol{\omega}} \delta_t \nabla_{\boldsymbol{\omega}} Q_{\boldsymbol{\omega}}(s_t, a_t), \tag{2.56}$$

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta} + \alpha_{\boldsymbol{\eta}} \nabla_{\boldsymbol{\eta}} \pi(s_t) \nabla_a Q_{\boldsymbol{\eta}}(s_t, a_t)|_{a=\pi_{\boldsymbol{\eta}}(s_t)}. \tag{2.57}$$

Unfortunately, despite the simplification of the expected return gradient calculation (which no longer needs integration with respect to the entire action space), ensuring adequate exploration becomes challenging because of the policy's deterministic nature, unless there is a substantial level of environmental noise. We can either add noise into the policy (making it non-deterministic) or learn the policy in an off-policy context, so as to follow a nondeterministic behavioral policy.

Deep deterministic policy gradient Deep deterministic policy gradient (DDPG) [72] is an off-policy actor critic algorithm that extends the work of DPG by introducing deep learning and borrowing techniques from DQN. As with DQN, DDPG features target networks (in this case one for the actor and one for the critic) to stabilize learning and prevent sudden changes. Another similarity lies in the use of experience replay to cope with the fact that samples are

not identically distributed due to sequential correlation of states. From DPG, instead, DDPG adopts the use of deterministic policies and gradients.

The algorithm starts arbitrarily initializing the four networks, one for the actor $\pi_{\boldsymbol{\eta}}$, one for the critic $Q_{\boldsymbol{\omega}}$ and two for their respective target networks $\pi_{\boldsymbol{\eta}'}$ and $Q_{\boldsymbol{\omega}'}$. At each time step t, a transition (s_t, a_t, r_t, s'_t) is collected, following the deterministic policy network $\pi_{\boldsymbol{\eta}}$ corrupted by random noise $\mathcal{N}_t$, and stored in a buffer $\mathcal{D}$. Note that the introduction of noise copes with the exploration issues of DPG. At this point, the algorithm proceeds by sampling n transitions from $\mathcal{D}$. These samples are used to compute the average squared TD error $L(\boldsymbol{\omega})$ that is used to update the critic network:

$$y_i = r_i + \gamma Q_{\boldsymbol{\omega}'}(s_{i+1}, \pi_{\boldsymbol{\eta}'}(s_{i+1})), \tag{2.58}$$

$$L(\boldsymbol{\omega}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - Q_{\boldsymbol{\omega}}(s_i, a_i))^2. \tag{2.59}$$

A similar computation is performed to obtain the sampled policy gradient $\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta})$ in order to update the actor network:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) \approx \frac{1}{n}\sum_{i=0}^{n-1} \nabla_a Q_{\boldsymbol{\omega}}(s, a)|_{s=s_i, a=a_i} \nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(s_i). \tag{2.60}$$

Another strong difference with respect to DQN stands on the update of the target networks. While in DQN they are slowly updated, i.e. frozen for some iteration before updating them, in DDPG the target networks are "softly" updated, i.e. the target networks parameters are adjusted gradually to follow the learned networks' progress:

$$\boldsymbol{\omega}' = \epsilon\boldsymbol{\omega} + (1-\epsilon)\boldsymbol{\omega}', \tag{2.61}$$

$$\boldsymbol{\eta}' = \epsilon\boldsymbol{\eta} + (1-\epsilon)\boldsymbol{\eta}', \tag{2.62}$$

with $\epsilon << 1$. This slow update ensures that the target values experience gradual changes, leading to a substantial enhancement in learning stability.

DDPG has proven its efficacy in addressing complex tasks with continuous action spaces [125, 128, 132]. The incorporation of deep neural networks, experience replay, target networks, and exploration strategies has made DDPG a versatile and robust tool for training agents in high-dimensional state and action spaces. Its theoretical foundations, coupled with its practical success in real-world applications, underscore its importance as a powerful and widely applicable algorithm in the realm of reinforcement learning.

2.3 Sample Complexity

In reinforcement learning, the efficient acquisition of knowledge and the ability to make informed decisions are pivotal. However, learning optimal policies is often constrained by a critical factor: sample complexity [48]. Sample complexity refers to the number of interactions or experiences an RL agent needs with its environment to acquire an effective policy or achieve "learning".

A high sample complexity implies a substantial requirement for gathering and utilizing a large amount of data during the learning process, which can result in slower convergence and increased computational costs, while a low sample complexity allows for more efficient and faster learning, often making it more practical and cost-effective. Understanding sample complexity is imperative because it unveils the practical feasibility and efficiency of RL algorithms in real-world scenarios. How many samples does it take for an RL agent to make informed decisions, adapt to dynamic environments, and achieve its goals? This question lies at the heart of sample complexity analysis.

In the context of this thesis, sample complexity is only approached indirectly by leveraging IRL and TL techniques. Nonetheless, this section will furnish a preliminary overview of the fundamental concepts and results, offering the reader a concise introduction to the subject.

2.3.1 Sampling Models

Sample complexity can be interpreted as the quantity of interactions with a sampling model necessary to attain a level of "learning." it is evident that this concept relies on the specific *sampling model* assumed (i.e. a model describing how samples or interactions are generated from the environment).

To characterize the sample complexity, the simplest sampling model is the vanilla **online simulation model** of an MDP. Within this framework, the agent starts from an initial state s_0 and follows an umbroken sequence of experiences. In practical terms, this means that the agent can select any action, represented as a, and will subsequently transition to a new state, denoted as $s' \sim P(\cdot|s,a)$, based on the probability distribution associated with state transitions. Importantly, the agent lacks the option to "reset" the MDP to a different state at will. This limitation emphasizes the critical importance of deliberate exploration within this particular context.

A widely adopted model weakening the need for explicit exploration is the *generative model*.

Definition 2.10 (Generative Model[51])**.** Given an MDP $\mathcal{M}$, a generative model $G(\mathcal{M})$ is a randomized algorithm that, on input of a state-action pair (s,a), outputs a couple $r(s,a)$, $s' \sim P(\cdot|s,a)$.

Because it is possible to acquire samples from any state at will, the exploration challenge in generative models is transformed into the task of determining the states from which to gather samples.

Between the online simulation model and the generative model lies the μ-reset model [47]. In this model, the agent possesses the ability to reset the present state to a state sampled from the initial state distribution μ. However, aside from this reset option, the model follows an online simulation approach.

2.3.2 Sample Complexity in main RL algorithms

Similarly to sampling models, the concept of sample complexity is intricately linked with the concept of "learning." In dynamic programming (or approximate dynamic programming) methods like Value/Policy iterations (or Q-learning/Approximate Policy Iteration), learning can be measured through the error metric $||Q^* - Q_k||_\infty$. On the other hand, for policy gradient methods the sample complexity is commonly considered [48] as the number of samples needed to obtain an accurate estimation of the gradient itself.

Value Iteration For the value iteration algorithm, assuming the existence of an upper bound for the maximum reward, denoted as $R_{\max} = \max_{s\in\mathcal{S},a\in\mathcal{A}} |r(s,a)|$, it is possible to establish an upper bound [18] for the difference between optimal value function and the value function at a given iteration k through:

$$||V^* - V_k||_\infty \leq 2\frac{\gamma^k}{(1-\gamma)^2}R_{\max}. \tag{2.63}$$

This means that, to achieve a desired level of accuracy ζ in the estimation of V^*, the algorithm must perform I iterations given from:

$$I = \left\lceil \log_\gamma \frac{\zeta(1-\gamma)^2}{2R_{\max}} \right\rceil. \tag{2.64}$$

Given that the algorithm iterates over all the state-action pairs for each iteration, the number of total table updates is given by $N = I|\mathcal{S}||\mathcal{A}|$.

Policy Iteration For the policy iteration algorithm, it can be proven [97] that at iteration k, if we assume a normalization of the value-function in an interval $[0, 1]$, it holds:

$$\begin{aligned} ||V^* - V^{\pi_k}||_\infty &\leq \gamma||V^* - V^{\pi_{k-1}}||_\infty \\ &\leq \gamma^k||V^* - V^{\pi_0}||_\infty \\ &\leq \gamma^k \end{aligned} \tag{2.65}$$

At this point we can obtain a desired accuracy ζ after I iterations, where:

$$I = \lceil \log_\gamma \zeta \rceil. \tag{2.66}$$

Similarly to the value-iteration case, the total number of updates to reach a given level of accuracy is given by $N = I|\mathcal{S}||\mathcal{A}|$.

Approximate Policy Iteration The authors in [14] expanded the exact policy-iteration upper bound for API using the subsequent lemma, assuming they possess a priori knowledge of the estimation error bound, denoted as $||Q_{\boldsymbol{\omega}} - Q^{\pi_k}||_\infty$:

Lemma 2.11. *Assume the sequence of approximate state-action values $Q_{\boldsymbol{\omega}}$ generated by approximate policy iteration satisfies $||Q_{\boldsymbol{\omega}} - Q^{\pi_k}||_\infty \leq \epsilon$. Then it holds that:*

$$||V^* - V^{\pi_k}||_\infty \leq \gamma^k + \frac{2\epsilon}{(1-\gamma)^2}. \tag{2.67}$$

Least Square Policy Iteration Authors of [65] provided two important results in the determination of the sample complexity of LSPI. In particular, Theorem 5 of [65] provides a PAC upper bound for the approximation error of LSTD. In our work [22], in section 3.4.2.4, we reformulated their theorem to fit in the LSTDQ context, thus obtaining:

$$\begin{aligned}\|Q^{\pi} - Q_{\boldsymbol{\omega}}\|_{\rho^{\pi}} \leq \frac{2}{\sqrt{1-\gamma^2}}\left(2\sqrt{2}\|Q^{\pi} - \Pi Q^{\pi}\|_{\rho} + \epsilon^{(1)}\right) + \epsilon^{(2)} \\ + \frac{2}{1-\gamma}\left[\gamma Q_{\max} L \sqrt{\frac{d}{\nu}}\left(\sqrt{\frac{8\log\left(8d_{\boldsymbol{\omega}}/\delta\right)}{n}} + \frac{1}{n}\right)\right],\end{aligned} \tag{2.68}$$

where $d_{\boldsymbol{\omega}}$ is the dimension of parameter $\boldsymbol{\omega}$, $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are $\mathcal{O}(Q_{\max}/\sqrt{N})$, $Q_{\max} = \frac{R_{\max}}{(1-\gamma)}$, and L is the upper bound of each basis function ψ_j of a feature vector $\boldsymbol{\psi}$, i.e. $\|\psi_j\|_{\infty} \leq L$.

Moreover, in [65], Theorem 8 offers a PAC upper bound for the approximation error of the value function associated with the policy π_k, which is the greedy policy with respect to the Q-function $Q_{\boldsymbol{\omega}}$, during the k-th iteration of LSPI:

$$\begin{aligned}||V^{*} - V^{\pi_k}||_{\sigma} \leq \frac{4\gamma}{(1-\gamma)^2}\Big\{(1+\gamma)\sqrt{CC_{\sigma,\mu}}\Big[\frac{2}{\sqrt{1-\gamma^2}}(2\sqrt{2}E_0(\mathcal{F}) + E_2) + \\ + \frac{2}{1-\gamma}(\gamma V_{\max} L\sqrt{\frac{d}{v_{\mu}}}(\sqrt{\frac{8\log(8dk/\delta)}{n}} + \frac{1}{n})) + E_1\Big] + \gamma^{\frac{k-1}{2}} R_{\max}\Big\}.\end{aligned} \tag{2.69}$$

While a detailed explanation of this bound goes beyond the scope of this thesis (we refer to [65] for more details), we noticed that the bound depends on the available number of samples, denoted as n, for each iteration k. At this point, similarly to previous bounds, one can derive the number of iterations, denoted as I, to attain an accuracy ζ and subsequently obtain the overall number of samples, which equals $N = In$.

Policy Gradient For the sample complexity of policy gradient methods, we will examine the quantity of samples required to achieve a precise estimation of the gradient. A crucial aspect to consider is that the gradient formula in equation (2.50) is expressed as an expectation over the state space, allowing for the use of sampling techniques for estimation. The gradient in equation (2.50) can be rewritten as:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \mathbb{E}_{s\sim\rho_{\boldsymbol{\eta}}, a\sim\pi_{\boldsymbol{\eta}}(\cdot|s)}\Big[\frac{\nabla_{\boldsymbol{\eta}}\pi_{\boldsymbol{\eta}}(a|s)}{\pi_{\boldsymbol{\eta}}(a|s)} Q^{\pi_{\boldsymbol{\eta}}}(s,a)\Big]. \tag{2.70}$$

Previous equation suggests an on-policy importance sampling procedure where trajectories generated by the learning policy are followed to estimate the gradient. However, this approach is not highly recommended, as the factor $\frac{\nabla_{\boldsymbol{\eta}}\pi_{\boldsymbol{\eta}}(a|s)}{\pi_{\boldsymbol{\eta}}(a|s)}$ can become arbitrarily large, especially in cases with nearly deterministic policies. One can also assume to have a bound for this factor, but it is unnatural for practical nearly deterministic policies.

Instead, for T-epoch MDPs [47] with discrete and finite action spaces it is possible to make an estimation by uniformly sampling actions. Thus writing the following form for the gradient:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = |\mathcal{A}| T \mathbb{E}_{s\sim\rho_{\boldsymbol{\eta}}, a\sim\mathcal{U}_{\mathcal{A}}}[\nabla_{\boldsymbol{\eta}}\pi_{\boldsymbol{\eta}}(a|s) Q^{\pi_{\boldsymbol{\eta}}}(s,a)]. \tag{2.71}$$

A way to obtain an unbiased estimate for equation (2.71) is through the following procedure:

1. compute m trajectories τ_j of length T where $a_{j,0}$ is uniformly sampled.
2. compute estimation of $Q^{\pi_{\boldsymbol{\eta}}}$ as $\hat{Q}_j = \sum_{t=0}^{T-1} \gamma^t r_{j,t}$
3. compute estimation of $\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta})$ as $\widehat{\nabla_{\boldsymbol{\eta}} J_{\boldsymbol{\eta}}} = \frac{AT}{m} \sum_{j=0}^{m-1} \nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}}(a_{j,0}|s_{j,0}) \hat{Q}_j$

Previous procedure constructs m estimates and returns their average, resulting in the algorithm observing a total of mT samples. It can be proven [47] that, given a policy parameter $\boldsymbol{\eta} \in \mathbb{R}^k$, a bound on the policy gradient $\nabla_{\boldsymbol{\eta}} \pi_{\boldsymbol{\eta}} \leq B$ and a desired accuracy ζ for the gradient estimation, if we examine $\mathcal{O}(\frac{|\mathcal{A}|T^3B^2}{\zeta^2} \log \frac{k}{\delta})$ samples, then with probability greater than $1 - \delta$ we have:

$$|\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) - \widehat{\nabla_{\boldsymbol{\eta}} J_{\boldsymbol{\eta}}}| \leq \zeta. \tag{2.72}$$

Final Remarks

Reinforcement learning's research area is vast, encompassing a wide array of literature and methodologies. While the subsequent chapters will introduce specific settings and algorithms pertinent to this thesis, it is essential to first establish the fundamental terminology and taxonomy of RL approaches that will recur throughout the whole document.

In conclusion, understanding the sample complexity in reinforcement learning is pivotal to assess the efficiency and feasibility of various algorithms. It sheds light on the number of interactions or experiences an agent needs to accumulate to learn a robust policy effectively. A lower sample complexity implies faster convergence and more data-efficient learning.

Chapter 3

Inverse Reinforcement Learning

This chapter focuses on the comprehensive exploration of inverse reinforcement learning (IRL). To start, we will provide an introductory overview and formalize the IRL problem. Next, we will delve into the state-of-the-art techniques in the field of IRL. Finally, our novel IRL formulation will be presented. This unique formulation aims to trade-off the identification of a reward function aligning with the expert behavior and its sample efficiency in the subsequent reinforcement learning phase. This algorithm serves as a bridging gap between IRL and RL's sample complexity, offering a promising avenue for future research and development.

3.1 Introduction to Inverse Reinforcement Learning

As we saw in the previous chapter, the power of reinforcement learning lies in the reward function, as it serves as the guiding light for the completion of decision-making tasks. However, RL agents can exhibit unforeseen behaviors when the reward function fails to capture all aspects of the task. In many real-world scenarios, manually crafting reward functions proves to be a daunting task [3, 86], often requiring an extensive domain knowledge. Take the case of autonomous driving, for instance, where various factors like obstacles, pedestrians, traffic lights, weather conditions, road gradients, and more influence the driving process. A failure to consider all these aspects in the reward function could lead to severe accidents or unsafe behaviors. Conversely, designing a comprehensive reward function that takes all these variables into account is a formidable challenge. Hence, the quest for a method that can autonomously discover a fitting reward function arises.

The hope of automatically acquiring a suitable reward function is rooted in the observation that, in many real-world scenarios, humans instinctively respond to various situations based on their experiences. However, the explicit understanding of how these experiences guide these reactions often eludes us. In simpler terms, it is often easier to access to a huge amounts of valuable human expert behaviors than to directly access the underlying reward signals that drive these behaviors. In the case of autonomous driving, we can readily collect driving data from a skilled driver who knows how to skillfully navigate obstacles, avoid pedestrians, halt at red lights, and reduce speed

on rainy days. This naturally raises the question of whether we can harness these demonstrated human expert behaviors for learning a policy. This inquiry brought to the development of a family of methods known as imitation learning [40]. As the name suggests, "imitation" refers to the method's aim to emulate human expert behaviors, while "learning" signifies the use of machine learning techniques to achieve this emulation. Formally, imitation learning takes as input an optimal policy or sampled states and actions from that policy (*demonstrated behaviors*) and produces an approximation of the optimal policy. During the early stages of imitation learning, one widely explored method was behavioral cloning [11], a supervised learning technique that trains a model by mimicking the behaviors of the expert. However, a significant limitation of behavioral cloning is that the learned policy may exhibit inaccuracies when the environmental dynamics change. This stems from the fact that an optimal policy is strongly dependent on the environment dynamics, as it maximizes rewards while considering the uncertainties quantified by the environment model. When these dynamics shift, the expert's policy may no longer retain its optimality, leading to the learned expert-like policy also losing its optimality.

Given the challenges encountered with behavioral cloning, it becomes pertinent to inquire whether there exists an imitation learning approach capable of preserving optimality when confronted with dynamic environmental changes. The answer is affirmative, and the solution revolves around a factor that remains consistent and robust across varying scenarios: the **reward function**. Consider this: if we possess prior knowledge of the reward function, we retain the ability to compute an optimal policy under new environmental dynamics, even when we face changing circumstances. This approach deviates from direct supervised learning, where the objective is to train a policy by minimizing the disparity between the policy and expert demonstrations. Instead, we opt for an indirect route, wherein we first deduce the underlying reward function from expert demonstrations and subsequently acquire a policy by optimizing the recovered reward function. The first step of this approach takes the name of **inverse reinforcement learning**.

The IRL problem can be mathematically formalized as follows:

Definition 3.1 (IRL [7])**.** Let $\mathcal{M} \backslash r^E = (\mathcal{S}, \mathcal{A}, P, \mu, \gamma)$ be a rewardless MDP Let $\mathcal{D} = \{\tau_0, \tau_1, ..., \tau_n\}$ be a dataset of n demonstrated trajectories τ_i of length H with the form

$$\tau_i = \Big((s_{i,0}, a_{i,0}), (s_{i,1}, a_{i,1}), ..., (s_{i,H-1}, a_{i,H-1})\Big),$$

where $s_{i,j} \in \mathcal{S}$, $a_{i,j} \in \mathcal{A}$. Then, the central task of IRL is to identify (at least) one reward function r, that best explains either the **expert policy** π^E (if provided, but usually unknown) or the observed behavior encapsulated in the form of demonstrated trajectories in $\mathcal{D}$.

In IRL, the modeling of rewards r is a pivotal aspect of the process. These representations can take various forms, depending on the particular approach used. One common approach is to model the reward function as a linear combination of feature functions [2, 84, 89, 137], where the choice of feature functions (usually encoding prior knowledge about the task, if available) and their corresponding weights defines the reward landscape. Alternatively, rewards may even be expressed through neural networks [127] or as probability distributions [10, 67, 93].

3.2 Inverse Reinforcement Learning Algorithms

In this section we will focus on the most significant IRL algorithms. By examining these algorithms, we aim to provide a comprehensive understanding of the IRL landscape, offering insights into how it navigates the challenge of determining the most suitable reward function to guide autonomous agents in modeling expert-like behavior.

3.2.1 Ng-Russell IRL

One of the pioneering algorithms in the field of IRL was the one introduced by Ng and Russell [84]. This algorithm can successfully reconstruct a reward function when provided with full knowledge of a rewardless discrete MDP $\mathcal{M}\backslash r^E$ and an expert policy that is an optimal deterministic policy $\pi^E = \pi^*$. Authors assume the transition model P to be a $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ tensor, and they define P_a to be the transition kernel, a $|\mathcal{S}| \times |\mathcal{S}|$ matrix, associated to action $a \in \mathcal{A}$ and P_{sa} to be a vector of dimension $|\mathcal{S}|$ reflecting the probability distribution of being in each state after applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. In their work, authors' major contribution is the characterization of the set of reward solutions through the following theorem.

Theorem 3.2. *Let $\mathcal{M}\backslash r^E = \{\mathcal{S}, \mathcal{A}, P, \mu, \gamma\}$ be a rewardless MDP with a finite action space $\mathcal{A} = \{a_0, ..., a_{|\mathcal{A}|-1}\}$ and a finite state space $\mathcal{S}$. Assuming, without loss of generality, $\pi^*(s) = a_0$, $\forall s \in \mathcal{S}$, then π^* is an optimal if and only if it holds:*

$$(P_{a_0} - P_a)(I - \gamma P_{a_0})^{-1} r \succeq 0, \quad \forall a \in \mathcal{A}\backslash\{a_0\}. \tag{3.1}$$

Here $\succeq$ represents a component-wise vectorial inequality.

We can notice that $V^{\pi^*} = (I - \gamma P_{a_0})^{-1} r$ is a $|\mathcal{S}|$-dimensional vector containing the value function for each state. So, equation (3.1) can be rewritten as:

$$P_{a_0} V^{\pi^*} \succeq P_a V^{\pi^*}, \quad \forall a \in \mathcal{A}\backslash\{a_0\} \tag{3.2}$$

which means that a_0 must be better to all other actions a, thus indicating that π^* is optimal Moreover, although it characterizes the solution space, in theorem 3.2 we can observe one of the biggest challenge of IRL: *reward ambiguity*. Given two reward functions, r_1 and r_2, which satisfy theorem 3.2, it is notable that any linear combination with non-negative coefficients would still yield a reward that is optimized by π^*. Furthermore, this set includes trivial reward functions, like constant or even zero functions. To break this ambiguity, authors propose an intuitive approach by selecting a reward that not only is optimized by π^* (thus solving the IRL problem) but also makes deviations from π^* as costly as possible. In other words, they seek for a reward in the set spanned by inequality (3.1) maximizing:

$$\sum_{s \in \mathcal{S}} Q^{\pi^*}_{r,\gamma}(s, a_0) - \max_{a \in \mathcal{A}\backslash\{a_0\}} Q^{\pi^*}_{r,\gamma}(s, a), \tag{3.3}$$

where with $Q^{\pi^*}_{r,\gamma}$ we denote the Q-function induced by π^* with reward r and discount factor γ. Furthermore, they introduce a regularization term λ to prefer reward functions with a limited

number of non-zero entries (L1-norm), thus obtaining the follow linear programming problem:

$$\max_{r} \sum_{s\in\mathcal{S}} \min_{a\in\mathcal{A}\setminus\{a_0\}} (P_{sa_0} - P_{sa})(I - \gamma P_{a_0})^{-1} r - \lambda ||r||_1 \tag{3.4a}$$

$$\text{s.t. } (P_{a_0} - P_a)(I - \gamma P_{a_0})^{-1} r \succeq 0, \quad \forall a \in \mathcal{A}\setminus\{a_0\} \tag{3.4b}$$

$$|r_i| \leq r_{\max} \quad i = 1, ..., |S|, \tag{3.4c}$$

where $r_{\max} < +\infty$ serves as an upper limit on the magnitude of the recovered reward[1].

Despite the assumption of having a finite state space $\mathcal{S}$, authors also propose an extension to infinite state spaces by the adoption of a linear approximation for the reward function:

$$r_{\boldsymbol{\theta}}(s) = \sum_{i=0}^{d_\theta - 1} \theta_i \phi_i(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s), \tag{3.5}$$

where ϕ_i are fixed and bounded basis functions. By denoting $V_i^{\pi^*}$ as the value function of the policy π^* in the MDP under the reward function's feature ϕ_i. Using the linearity of expectations, the value function under the reward function r defined by equation (3.5) can be expressed as follows:

$$V^{\pi^*} = \sum_{i=0}^{d_\theta - 1} \theta_i V_i^{\pi^*}. \tag{3.6}$$

In this context, theorem 3.2 (and so equation (3.2)) can be rewritten as:

$$\mathbb{E}_{s'\sim P(\cdot|s,a_0)}[V^{\pi^*}(s')] \geq \mathbb{E}_{s'\sim P(\cdot|s,a)}[V^{\pi^*}(s')], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}\setminus\{a_0\} \tag{3.7}$$

The current formulation faces two main challenges. Firstly, when dealing with infinite state spaces, there is an issue related to the infinite number of constraints from equation (3.7). Checking all these constraints becomes impractical or even impossible. To tackle this, the authors take an algorithmic solution by working only with a large but finite subset of states, denoted as $\mathcal{S}_0$, and applying constraints (3.7) only to states $s \in \mathcal{S}_0$. The second challenge is more intricate. By restricting the use of the linear function approximator in equation (3.5) to represent r, it may happens that there's no linear approximation of reward functions for which π^* is optimal. However, in order to prevent ending up with trivial solutions, authors permit violations of constraints (3.7), and accepting paying penalties for any violation, thus obtaining the final optimization problem:

$$\max_{\boldsymbol{\theta}\in\mathbb{R}^{d_{\boldsymbol{\theta}}}} \sum_{s\in\mathcal{S}_0} \min_{a\in\mathcal{A}\setminus\{a_0\}} c(\mathbb{E}_{s'\sim P(\cdot|s,a_0)}[V^{\pi^*}(s')] - \mathbb{E}_{s'\sim P(\cdot|s,a)}[V^{\pi^*}(s')]) \tag{3.8a}$$

$$\text{s.t. } |\boldsymbol{\theta}_i| \leq r_{\max} \quad i = 0, ..., d_{\boldsymbol{\theta}} - 1, \tag{3.8b}$$

where $c(\cdot)$ represent a penalty function.

[1] Early approaches defined reward function to be function of the states. Later these function have been defined for state-action couples.

3.2.2 Feature Expectation Algorithms

At the core of these algorithms, firstly introduced by [2], lies the concept of *feature expectation*. For any policy π, one can calculate the so-called feature expectations $\boldsymbol{\varphi}(\pi)$ by considering the expectation of features based on the trajectory distribution induced by the policy itself. For a finite length trajectory $\tau = (s_0, a_0, ..., s_{H-1})$, we can write the feature expectation as:

$$\boldsymbol{\varphi}(\pi) = \mathbb{E}_{\substack{s_{t+1}\sim P(\cdot|a_t,s_t) \\ a_t\sim\pi(\cdot|s_t)}}\Big[\sum_{t=0}^{H-1}\gamma^t\boldsymbol{\phi}(s_t)\Big]. \tag{3.9}$$

Similarly to the Ng-Russel approach, the reward is represented as a linear combination of a feature functions vector $\boldsymbol{\phi}$ and a weight vector $\boldsymbol{\theta}$. For this reason the expected return can be expressed using the feature expectations, thus characterizing the performance of a given policy π in the linear approximation space as:

$$J(\boldsymbol{\varphi}) = \mathbb{E}_{\substack{s_{t+1}\sim P(\cdot|a_t,s_t) \\ a_t\sim\pi(\cdot|s_t)}}\Big[\sum_{t=0}^{H-1}\gamma^t r(s_t)\Big] = \mathbb{E}_{\substack{s_{t+1}\sim P(\cdot|a_t,s_t) \\ a_t\sim\pi(\cdot|s_t)}}\Big[\sum_{t=0}^{H-1}\gamma^t\boldsymbol{\theta}^\top\boldsymbol{\phi}(s_t)\Big] = \boldsymbol{\theta}^\top\boldsymbol{\varphi}(\pi). \tag{3.10}$$

We denote by $\pi^{\boldsymbol{\theta}}$ the optimal policy that arises from the reward function $\boldsymbol{\theta}^\top\boldsymbol{\phi}$. Our objective is to find a reward function that results in an optimal policy which is as similar as possible to the expert's policy π^E. In other words, we seek to identify $\boldsymbol{\theta}$ that minimize the dissimilarity (e.g., Euclidean distance) between the expert's feature expectation $\boldsymbol{\varphi}(\pi^E)$ and the feature expectations $\boldsymbol{\varphi}(\pi^{\boldsymbol{\theta}})$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\mathbb{R}^{d_{\boldsymbol{\theta}}}}{\arg\min}\,||\boldsymbol{\varphi}(\pi^E) - \boldsymbol{\varphi}(\pi^{\boldsymbol{\theta}})||_2. \tag{3.11}$$

Given the available n trajectories τ_i in $\mathcal{D}$, we can estimate the expert's feature expectations as follows:

$$\boldsymbol{\varphi}(\pi^E) \approx \boldsymbol{\varphi}^E = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{t=0}^{H-1}\gamma^t\boldsymbol{\phi}(s_{i,t}). \tag{3.12}$$

Authors of [2] propose two iterative algorithms to solve the problem in equation (3.11): *max-margin algorithm* and *projection algorithm*. Both these algorithms iteratively alternates two phases. First, a reward parameter estimation is computed, then a policy is learnt employing the new reward in a standard RL algorithm.

In *max-margin* algorithm, at each iteration k, the reward parameter $\boldsymbol{\theta}_k$ is computed by solving the following optimization problem:

$$(m_k, \boldsymbol{\theta}_k) = \underset{m\in\mathbb{R},\boldsymbol{\theta}\in\mathbb{R}^{d_\theta}}{\arg\max}\; m \tag{3.13}$$

$$\text{s.t. } \boldsymbol{\theta}^\top\boldsymbol{\varphi}^E \geq \boldsymbol{\theta}^\top\boldsymbol{\varphi}(\pi_{k'}) + m, \quad k' = 0, ..., k-1 \tag{3.14}$$

$$||\boldsymbol{\theta}||_2 \leq 1 \tag{3.15}$$

Once the algorithm computes the margin m_k (i.e. a measure of the difference between the feature expectation of the expert and the feature expectation associated to π_{k-1}) and the reward parameter $\boldsymbol{\theta}_k$, if the margin is lower than a given threshold, the algorithm stops returning $\boldsymbol{\theta}_k$.

Otherwise, the algorithm proceeds with the computation of the optimal policy π_k optimizing reward $r_{\boldsymbol{\theta}_k}(s) = \boldsymbol{\theta}_k^\top \boldsymbol{\phi}(s)$ via any standard RL algorithm.

Projection algorithm proceeds similarly to max-margin but avoid the task of solving an optimization problem. This algorithms compute the weight vector as $\boldsymbol{\theta}_k = \boldsymbol{\varphi}^E - \bar{\boldsymbol{\varphi}}_{k-1}$ where $\bar{\boldsymbol{\varphi}}_{k-1}$ is the orthogonal projection of $\boldsymbol{\varphi}^E$ onto the line through $\bar{\boldsymbol{\varphi}}_{k-2}$ and $\boldsymbol{\varphi}_{k-1}$:

$$\bar{\boldsymbol{\varphi}}_{k-1} = \bar{\boldsymbol{\varphi}}_{k-2} + \frac{(\boldsymbol{\varphi}_{k-1} - \bar{\boldsymbol{\varphi}}_{k-2})^\top (\boldsymbol{\varphi}^E - \bar{\boldsymbol{\varphi}}_{k-2})}{(\boldsymbol{\varphi}_{k-1} - \bar{\boldsymbol{\varphi}}_{k-2})^\top (\boldsymbol{\varphi}_{k-1} - \bar{\boldsymbol{\varphi}}_{k-2})} (\boldsymbol{\varphi}_{k-1} - \bar{\boldsymbol{\varphi}}_{k-2}). \tag{3.16}$$

Then, the margin m_k is computed as:

$$m_k = ||\boldsymbol{\varphi}_E - \bar{\boldsymbol{\varphi}}_{k-1}||_2. \tag{3.17}$$

Both these algorithm are proven to terminate [2] and the set of all the policies π_k (such that $r_{\boldsymbol{\theta}_k}(s) = \boldsymbol{\theta}_k^\top \boldsymbol{\phi}(s)$ makes them optimal) contains a policy that, when evaluated under the unknown reward function r_E, performs at least as well as the expert's performance minus a small margin ϵ.

3.2.3 Entropy-based Algorithms

One common critical point of feature expectation algorithms is the reliance to a RL algorithm to iteratively solve the IRL problem. Entropy-based approaches seek to reduce dependency on RL by directly incorporating entropy-based strategies into feature expectation algorithms, thereby reducing the necessity for iterative problem-solving in IRL.

Maximum Entropy Inverse Reinforcement Learning Maximum entropy inverse reinforcement learning [137] resolves the reward ambiguity by adopting the maximum entropy principle [43]. The principle of maximum entropy is relevant in the context of estimating a probability distribution with constraints based on data. For instance, it is used to estimate the distribution of expert trajectories while ensuring the matching of the expert's feature expectations. This principle states that the most suitable probability distribution for accurately modeling the available data is the one characterized by the highest entropy. In simpler terms, when no constraints are in place, one may opt for a uniform distribution, which boasts the maximum achievable entropy. Conversely, when specific constraints are present, it has been demonstrated that adopting the maximum likelihood Boltzmann distribution is necessary [43].

Like previous algorithms, in this method the true reward function is assumed to be approximated by a linear combination of state features $r_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s)$. Given a trajectory $\tau_i \in \mathcal{D}$, the feature count and the discounted return over τ_i can be, respectively, defined as:

$$\boldsymbol{\varphi}(\tau_i) = \sum_{t=0}^{H-1} \gamma^t \boldsymbol{\phi}(s_{i,t}), \tag{3.18}$$

$$J_{\boldsymbol{\theta}}(\tau_i) = \sum_{t=0}^{H-1} \gamma^t \boldsymbol{\theta}^\top \boldsymbol{\phi}(s_{i,t}) = \boldsymbol{\theta}^\top \boldsymbol{\varphi}(\tau_i). \tag{3.19}$$

Disposing of n trajectories, the objective is to infer the probability distribution of these trajectories, denoted as $\rho_{\boldsymbol{\theta}}$, while ensuring that the feature count's expectation align with the average feature count $\bar{\boldsymbol{\varphi}} \triangleq \frac{1}{n}\sum_{i=0}^{n-1}\boldsymbol{\varphi}(\tau_i)$, ultimately obtaining:

$$\mathbb{E}_{\tau\sim\rho_{\boldsymbol{\theta}}}[\boldsymbol{\varphi}(\tau)] = \bar{\boldsymbol{\varphi}}. \tag{3.20}$$

In line with the maximum entropy principle [43], this task is equivalent to identify the maximum likelihood Boltzmann distribution from the sampled trajectories conditioned by the state transition density P, such that it aligns with the average feature count $\bar{\boldsymbol{\varphi}}$:

$$\rho_{\boldsymbol{\theta}}(\tau_i) = \frac{e^{\boldsymbol{\theta}^\top\boldsymbol{\varphi}(\tau_i)}}{\sum_{j=0}^{n-1} e^{\boldsymbol{\theta}^\top\boldsymbol{\varphi}(\tau_j)}} \prod_{t=0}^{H-1} P(s_{i,t+1}|a_{i,t}, s_{i,t}), \quad i = 0, ..., n-1. \tag{3.21}$$

Consequently, $\rho_{\boldsymbol{\theta}}$ induces a stochastic policy, where the probability of selecting an action is determined by the expected exponentiated rewards of all trajectories starting with that action:

$$\pi^{\boldsymbol{\theta}}(a|s) = \sum_{\substack{i:(s_{i,0},a_{i,0})=(s,a)\\ i=0,\dots,n-1}} \rho_{\boldsymbol{\theta}}(\tau_i). \tag{3.22}$$

In case of a deterministic MDP, equation (3.21) can be rewritten as:

$$\rho_{\boldsymbol{\theta}}(\tau_i) = \frac{e^{\boldsymbol{\theta}^\top\boldsymbol{\varphi}(\tau_i)}}{\sum_{j=0}^{n-1} e^{\boldsymbol{\theta}^\top\boldsymbol{\varphi}(\tau_j)}}, \quad i = 0, ..., n-1 \tag{3.23}$$

According to equation (3.23), trajectories sharing the same discounted return have same probabilities while trajectories exhibiting an higher reward are exponentially preferred. Moreover, by maximizing the entropy of the trajectory distribution while subject to the feature constraints observed in the data, we are essentially maximizing the likelihood of the observed data when applying the maximum entropy distribution mentioned earlier:

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \sum_{i=0}^{n-1} \log \rho_{\boldsymbol{\theta}}(\tau_i). \tag{3.24}$$

When applied to deterministic MDPs, problem in equation (3.24) is convex, and its optimal solutions can be obtained through gradient-based optimization techniques, where the gradient of the likelihood function $L(\boldsymbol{\theta}) = \log \rho_{\boldsymbol{\theta}}(\tau_i)$, is intuitively the difference between the expert feature expectation $\bar{\boldsymbol{\varphi}}$ and the feature expectation induced by the current estimate of the linear reward function $r_{\boldsymbol{\theta}}$:

$$\nabla_{\boldsymbol{\theta}} \sum_{i=0}^{n-1} \log p_{\boldsymbol{\theta}}(\tau_i) = \bar{\boldsymbol{\varphi}} - \sum_{i=0}^{n-1} \rho_{\boldsymbol{\theta}}(\tau_i)\boldsymbol{\varphi}(\tau_i) = \bar{\boldsymbol{\varphi}} - \sum_{s\in\mathcal{S}} d_{\boldsymbol{\varphi}}^{\pi^{\boldsymbol{\theta}}}(s)\boldsymbol{\phi}(s), \tag{3.25}$$

where $d_{\boldsymbol{\varphi}}^{\pi^{\boldsymbol{\theta}}}$ represents the expected state visitation frequency, i.e. the probability of being in a given state under policy $\pi^{\boldsymbol{\theta}}$ (equation (3.22)). Algorithm 1 from [137] provides a dynamic programming procedure for the computation of $d_{\boldsymbol{\varphi}}^{\pi^{\boldsymbol{\theta}}}$ which takes, as input, the reward parameter $\boldsymbol{\theta}$ and the state transition density P.

Deep Maximum Entropy Inverse Reinforcement Learning The selection of the approximation model significantly influences the algorithm's ability to capture the connection between feature expectations and rewards. Approaches like feature expectation algorithms and maximum entropy IRL presume that the mapping from state to reward can be represented as a linear combination of weighted features.

This choice, while computationally straightforward and convenient, may not be suitable when the true reward function's characteristics are best explained by complex nonlinear models. Deep Maximum Entropy Inverse Reinforcement Learning [127] employs the maximum entropy principle while using nonlinear approximators based on deep neural networks. In this approach, the reward function is approximated in a nested nonlinear function space, allowing for more complex modeling:

$$\begin{aligned} r_{\boldsymbol{\theta}}(s) &= g(\boldsymbol{\theta}, \phi(s)) \\ &= g_{0,\boldsymbol{\theta}_0}(g_{1,\boldsymbol{\theta}_1}(...(g_{d_\theta-1,\boldsymbol{\theta}_{d_\theta-1}}(\boldsymbol{\phi}(s))))). \end{aligned} \tag{3.26}$$

Importantly, when utilizing backpropagation, neural networks naturally align with the training process in the maximum entropy IRL framework. Moreover, the network structure can be customized to match specific tasks without introducing complexities or undermining the core IRL learning mechanism [127]. In the context of deep maximum entropy IRL, we gain access to a diverse array of architectural choices. Different problem domains can leverage distinct network structures, for instance, convolutional layers can eliminate the reliance on manually crafted spatial features.

It is worth noting that the aforementioned linear maximum entropy IRL can be considered as a simplification of the more comprehensive deep approach. This simplification can be achieved by applying the principles of backpropagation to a network with a single linear output connected to all inputs and zero bias terms.

Relative Entropy Inverse Reinforcement Learning Maximum entropy inverse reinforcement learning relies on the assumption of knowing the probability density function of the MDP. A model-free extension based on importance sampling, overcoming the necessity of a transition model, has been proposed in [16]. This approach aim to minimize the relative entropy, namely the Kullback-Leibler divergence [59], between the empirical distribution of the state-action trajectories under a baseline policy and the distribution of the trajectories under a policy that matches the reward feature counts of the expert. The resulting trajectory distribution $\rho_{\boldsymbol{\theta}}$ is given by:

$$\rho_{\boldsymbol{\theta}}(\tau_i) = U(\tau_i) \frac{e^{\boldsymbol{\theta}^\top \boldsymbol{\varphi}(\tau_i)}}{\sum_{j=0}^{n-1} e^{\boldsymbol{\theta}^\top \boldsymbol{\varphi}(\tau_j)}} \mu(s_{i,0}) \prod_{t=0}^{H-1} P(s_{i,t+1}|s_{i,t}, a_{i,t}), \tag{3.27}$$

where $U(\tau_i)$ is the joint probability of the actions conditioned on the states in τ_i. Despite equation (3.27) depends on the MDP's transition density, authors of [16] provides an importance sampling procedure for the estimation of the trajectory distribution; we refer to section 4.3 of [16] for such a procedure.

Despite being a model-free approach, this method requires additional interaction with the environment to sample trajectories with a non-expert exploratory policy. The following method overcomes this limitations.

3.2.4 Gradient Inverse Reinforcement Learning

Gradient inverse reinforcement learning (GIRL) [89] is a model-free IRL approach that, given a parametric stochastic expert policy $\pi^E_{\boldsymbol{\eta}}$, optimal w.r.t. the unknown reward r^E, and a dataset $\mathcal{D}$ of n trajectories collected by the expert, aims to directly recover the expert reward exploiting policy-gradient RL formulations.

To begin, let's recall the result of the policy gradient theorem in equation (2.50) and let's consider the case we want to compute the policy gradient of the expert under the unknown reward function r^E:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \mathbb{E}_{s\sim\rho_{\boldsymbol{\eta}}}\Big[\int_{\mathcal{A}} \nabla_{\boldsymbol{\eta}} \pi^E_{\boldsymbol{\eta}}(a|s) Q^{\pi^E_{\boldsymbol{\eta}}}_{r^E}(s,a) da\Big], \tag{3.28}$$

where, $Q^{\pi^E_{\boldsymbol{\eta}}}_{r^E}$ is the Q-function induced by the expert policy $\pi^E_{\boldsymbol{\eta}}$ and reward r^E.

If the expert policy is optimal with respect to r^E, equation (3.28) must be equal to 0. The idea behind GIRL involves a reformulation of equation (3.28), taking into account a parametric reward represented as $r_{\boldsymbol{\theta}}$:

$$\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}, \boldsymbol{\theta}) = \mathbb{E}_{s\sim\rho_{\boldsymbol{\eta}}}\Big[\int_{\mathcal{A}} \nabla_{\boldsymbol{\eta}} \pi^E_{\boldsymbol{\eta}}(a|s) Q^{\pi^E_{\boldsymbol{\eta}}}_{r_{\boldsymbol{\theta}}}(s,a) da\Big]. \tag{3.29}$$

Notice that J can be estimated offline through any policy-gradient algorithm and exploiting the trajectories dataset $\mathcal{D}$.

The goal, now, is to find a parameter $\boldsymbol{\theta}^*$ that minimize the gradient in a way that ensures the expert's optimality remains intact. In other words, to find $\boldsymbol{\theta}^*$ that is a stationary point for $J(\boldsymbol{\eta}, \cdot)$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{y} \|\nabla_{\boldsymbol{\eta}} J(\boldsymbol{\eta}, \boldsymbol{\theta})\|^y_x, \tag{3.30}$$

for any $x, y \geq 1$.

One of the algorithm's notable characteristics is the convex nature of the objective function when the parametric reward model (which is a design choice) is convex with respect to $\boldsymbol{\theta}$, thus resulting to be easily solvable using any standard convex optimization approach.

In case the expert's policy is not optimal for any reward function in the chosen class, the solution $\boldsymbol{\theta}^*$ corresponds to the minimum norm gradient. This means it represents the reward that leads to the smallest possible change in the policy parameters. In essence, GIRL aims to determine the reward weights that most effectively explain the expert's policy trajectories.

3.2.5 Bayesian Methods

Bayesian IRL (BIRL) methods offer a probabilistic framework to address the challenge of inferring the underlying reward function. While aforementioned IRL approach seeks a single "optimal" reward function, Bayesian IRL extends this concept by treating the reward function as a random variable and capturing its uncertainty. This is achieved through the application of Bayesian principles. In Bayesian IRL, the reward function is assigned a prior distribution, which encodes prior beliefs about the possible forms of the reward function. Then, through a process of posterior inference, this prior distribution is updated with information from the expert's demonstrations to yield a posterior distribution over reward functions.

Let R be a reward function chosen from a (known) prior distribution P_R. Let $Pr(\tau|R)$ be the trajectory likelihood of the reward. Then, the Bayesian update is applied to derive a posterior distribution employing the candidate reward functions through the following process:

$$Pr(R|\tau) = Pr(\tau|R)\underbrace{P_R(R)}_{\text{known}}. \tag{3.31}$$

The trajectory likelihood can be factored as:

$$Pr(\tau|R) = \prod_{(s,a)\in\tau} Pr((s,a)|R), \tag{3.32}$$

and it is performed for each $\tau \in \mathcal{D}$. These Bayesian IRL methods can be categorized based on the implementation of the trajectory likelihood in equation (3.32).

Boltzmann Distribution The earliest Bayesian IRL work, BIRL [93], models the trajectory likelihood as the Boltzmann distribution with the Q-value of the state-action pair as the energy function:

$$Pr((s,a)|R) = e^{\alpha Q^*_{R,\gamma}(s,a)}, \tag{3.33}$$

where α represents the degree of confidence we have in the expert's ability to choose actions with high value.

Concerning the prior distribution for reward functions, authors of BIRL propose a wide range of alternatives. While opting for the uniform density function is an agnostic choice, it is important to note that many real-world MDPs possess sparse reward structures, making Gaussian or Laplacian priors more suitable [93]. In cases where the MDP represents a planning problem with significant disparities in rewards, such as goal states carrying substantial positive rewards, a Beta density may be a better fit. However, working within the continuous space of reward functions poses the analytical challenge of deriving the posterior distribution. To overcome this obstacle, BIRL introduces a random walk-based Markov Chain Monte Carlo algorithm operating in the policy space. This approach allows for the generation of sample-based empirical approximations of the posterior distribution [93].

Finally, once the posterior distribution is defined, one can determine the reward with an estimation task. Common losses for this tasks are the linear and squared error loss functions:

$$L_{\text{linear}}(R, \hat{R}) = ||R - \hat{R}||_1 \tag{3.34}$$

$$L_{\text{SE}}(R, \hat{R}) = ||R - \hat{R}||_2, \tag{3.35}$$

where R and $\hat{R}$ are the actual and estimated rewards, respectively. When the reward R is sampled from the posterior distribution as defined in equation (3.31), it can be demonstrated [12] that the expected value of $L_{\text{SE}}(R, \hat{R})$ is minimized when $\hat{R}$ is assigned the mean of the posterior distribution. Likewise, optimizing the expected linear loss involves setting $\hat{R}$ to the median of the distribution.

Other works *Gaussian Process IRL* (GPIRL) [67] is an influential Bayesian approach, developed after BIRL, allowing the reward function to take the form of a nonlinear function of features, denoted as $r_{\boldsymbol{\theta}} = f_{\boldsymbol{\theta}}(r, \boldsymbol{\phi})$. In this approach, the reward function is modeled as a Gaussian process, with the underlying structure determined by a kernel function parameterized by $\boldsymbol{\theta}$. Consequently, the posterior distribution, which was originally represented as $Pr(R|\tau)$ in equation (3.31), is now given by $Pr(r_{\boldsymbol{\theta}}|\tau, \boldsymbol{\phi})$, where $r_{\boldsymbol{\theta}}$ corresponds to the rewards associated with the feature functions $\boldsymbol{\phi}$ at specific states and actions. The GPIRL technique, computes this posterior through a Bayesian update, where the likelihood term is represented as $Pr(\tau|r_{\boldsymbol{\theta}})Pr(r_{\boldsymbol{\theta}}|\tau, \boldsymbol{\phi})$. The first factor of this expression is computed as previously shown in equation (3.33). However, the second factor, often referred to as the Gaussian process posterior, is a Gaussian distribution with analytically derived mean and covariance matrices. GPIRL extends its knowledge from a limited subset of reward values, specifically those present in the observed trajectories and a few additional rewards assigned to random states. To optimize the likelihood and determine the most likely r and, consequently, the most probable reward functions, GPIRL employs L-BFGS [73] optimization with restarts.

Maximum Likelihood IRL (MLIRL) [10], differently from the posterior estimation techniques used in BIRL methods, proposes an alternative approach that directly maximize the likelihood of the expert data in equation (3.33). The conventional formulation of data likelihood involves the policy and the Bellman operator, which, unfortunately, is non-differentiable. To address this, a softmax Boltzmann exploration strategy is employed to modify the policy expression from a maximization problem to an analytical one as:

$$\pi^{\boldsymbol{\theta}}(a|s) = \frac{e^{\beta Q_R(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\beta Q_R(s,a')}}. \tag{3.36}$$

This modification renders the likelihood $Pr(\tau|R)$ differentiable. Since the likelihood is now differentiable, the algorithm utilizes a standard gradient ascent approach to converge toward the (locally) optimal reward weights $\boldsymbol{\theta}$.

3.3 Challenges

So far, we have introduced various approaches to solve the reward inference problem. Each of these approaches comes with its own strengths and weaknesses.

For example, model-based algorithms, such as Ng-Russel or Maximum Entropy RL, are often challenging to apply in real-world scenarios because of their reliance to the transition model of the underlying MDP. One can think to estimate such a model, but it becomes particularly difficult when dealing with continuous state and/or action action spaces, especially when the only available data is provided by an expert. Indeed the expert, typically representing an optimal policy, explores only a small portion of the state space, which is insufficient to accurately estimate the MDP's transition model.

Other algorithms, like feature expectation methods, don't explicitly require the MDP's transition model, but they have their own set of limitations. These approaches heavily rely on the utilization of reinforcement learning to learn (temporary) policies and exploit them to generate new trajectories and their feature expectation. These methods can be unfeasible in real-world applications where access to an actual environment is limited, or collecting data is expensive. Additionally, the use of RL for data generation often necessitates a substantial amount of trial-and-error exploration, which may not always be practical, especially in contexts where mistakes can have serious consequences. Among described algorithms, GIRL is the only "pure" model-free approach. Moreover, it is the sole approach capable of inferring a reward function using only the trajectories of the expert. But, on the other hand, GIRL relies on the strong assumption of knowing the expert's policy and its parameters.

Another critical challenge, initially identified by Ng and Russell [84], is that a multitude of reward functions, even those with significant degeneracy, such as a uniform zero reward across all states, can effectively account for the provided expert's observations. This issue arises from the fact that the input data typically comprises a finite and restricted collection of trajectories, and within the entire spectrum of possible reward functions, numerous options exist that can yield policies consistent with the observed demonstrations. Consequently, IRL grapples with a substantial challenge, characterized by a lack of a unique or unambiguous solution to the reward function inference problem; typically, this difficulty is referred to in the literature as the "ill-posedness" of IRL [7, 19, 134]. Moreover the selection of a reward function from the multitude of feasible options is inherently linked to the definition of optimality chosen for the task at hand. In other words, the reward selection should also consider how it is employed and which specific characteristics it should possess, aside from ensuring the compatibility with the expert's behavior. Surprisingly, this issue hasn't received comprehensive treatment in the existing literature. In essence, the objective is to identify a reward that exhibits favorable learning characteristics, such as the ability to facilitate rapid learning of the optimal policy. Nonetheless, the speed of learning is contingent on the specific algorithm employed. Only maximum entropy IRL has explicitly tackled this challenge, introducing an optimality criterion grounded in the maximum entropy principle.

3.4 Sample-Aware Inverse Reinforcement Learning

In light of the above-mentioned challenges of IRL, in this section we propose a novel model-free IRL framework that seeks to address the challenge of reward ambiguity by taking into account two critical dimensions: the consistency of the identified reward with expert behavior and its impact on the traditional (or "forward") RL phase.

Its model-free nature sets it apart from many existing methods, making it more adaptable to real-world situations where precise models may be challenging to construct. This adaptability enables our approach to be employed even in complex, dynamic environments, making it suitable for a wide range of practical applications. Moreover, and foremost, our framework takes a comprehensive approach to the IRL problem. It doesn't just aim to match the expert's behavior with the reward but also ensures that this alignment translates into a more efficient learning. This dual focus on reward consistency and learning efficiency is a distinctive feature of our approach and directly addresses the issue of reward ambiguity. Furthermore, it addresses the practical concern of sample budget limitations in the subsequent forward learning phase, ensuring that learning remains feasible even with constrained resources. By doing so, our approach considerably reduces the sample complexity and expedites the forward learning process, making it a valuable option for real-world scenarios where efficient learning is crucial.

In practical terms, our new model-free IRL method autonomously balances the potential error induced on the learned policy by choosing a potentially sub-optimal reward and the estimation error arising from finite samples in the forward learning phase. This balance is achieved by explicitly optimizing both a parameterized reward and the discount factor of the associated learning problem.

3.4.1 Theoretical Formulation

The challenge of efficient learning is closely intertwined with the notion of sample complexity, which can be framed as the inquiry of *how much data must we collect in order to achieve "learning"?* [48]. Within the RL context, the quantity of sampling model invocations typically depends on various problem parameters. These include the discount factor γ, which correlates with the effective number of decision epochs, as well as the reward (and its variance), alongside an accuracy parameter ϵ [2] concerning the chosen performance criteria, and a confidence parameter δ. As the discount factor decreases, the number of samples needed to achieve a near-optimal estimate of the optimal value function diminishes. This phenomenon is substantiated by numerous theoretical findings related to sample complexity [9, 28, 65, 83], often relying on a power of $1/(1-\gamma)$ as a key factor. However, it is crucial to note that the reward function derived through IRL must align with the expert's policy, which is estimated from a finite set of demonstrations and is known with a certain level of accuracy and confidence.

Our IRL formulation integrates these various elements together, and explicitly accounts for the impact of the learned IRL reward on the subsequent forward learning phase. Let's consider a

[2] In chapter 2 we adopted the symbol ζ to avoid misunderstandings with other variables.

forward RL algorithm that, given a reward function r, a discount factor γ, and a sample budget of $M \geq 0$, can produce an $\epsilon^\star(M, \gamma)$-approximation denoted as $\widehat{Q}^\star_M$ for the optimal Q-function $Q^\star_{r,\gamma}$ with a probability of at least $1 - \delta^\star$. Then, the impact of the IRL reward r and discount factor γ on the distance between the expert's policy π^E and the (greedy) policy learned in the subsequent forward learning task, considering the presence of a finite and potentially limited number M of samples, can be encapsulated through the following adversarial min-max optimization program:

$$\min_{r \in \mathcal{R}, \gamma \in [0,1)} \max_{\pi \in \mathcal{G}\left[\widehat{Q}^\star_M\right]} \left\| Q^{\pi^E}_{r^E,\gamma^E} - Q^{\pi}_{r^E,\gamma^E} \right\| \tag{3.37a}$$

$$\text{s.t.} \left\| \widehat{Q}^\star_M - Q^\star_{r,\gamma} \right\| \leq \epsilon^\star(M, \gamma), \tag{3.37b}$$

where $\mathcal{R}$ is a set of available reward functions and $\|\cdot\|$ is a suitably defined norm.

Equation (3.37a) represents a worst-case guarantee regarding the sub-optimality of the learned policy π compared to the expert's policy π^E, as assessed using the actual (and unknown) reward r^E and discount factor γ^E. This also implies the *compatibility* of the learned reward r with the expert's policy π^E, which is the primary condition in IRL. Additionally, by explicitly optimizing the learned discount factor γ, it becomes possible to balance the trade-off between the reward and the optimality of the learned policy π, thus allowing for the adjustment of the sample complexity in the subsequent forward RL task. For this purpose, we introduce the confidence region of the future optimal Q-function $\widehat{Q}^\star_M$ in equation (3.37b). This region is influenced by the optimized reward r and discount factor γ and is determined by the specific forward RL algorithm employed. This set defines the viable space within which we can search for a greedy policy that emulates the expert's one, within the degree of accuracy $\epsilon^\star$, a confidence level $\delta^\star$ and depending on the quantity of data M accessible during the subsequent forward learning phase.

Essentially, our IRL formulation in equation (3.37) introduces a novel element that prior IRL approaches have overlooked. This element involves striking a balance between optimality (in the IRL context, signifying compatibility) and the sample complexity required to learn a policy in the ensuing forward RL phase. The obtained reward is designed to minimize the most extreme error scenario in the forward RL context, as defined in objective (3.37a), where a limited sample budget M is available, as specified in constraint (3.37b). This does not suggest that learning the policy will be straightforward in all cases. However, the optimized pair (r, γ)[3] will maximize the efficiency of learning the forward RL policy.

It is important to note that the formulation in equation (3.37) is not immediately solvable due to its dependence on the unknown variables r^E and γ^E within its objective function. In subsection 3.4.2, we will show how we can eliminate the need for the state-action value functions $Q^{\pi^E}_{r^E,\gamma^E}$ and $Q^{\pi}_{r^E,\gamma^E}$ and provide an approximation of the confidence region outlined in equation (3.37b).

[3]The reason for representing (r, γ) under the min operator is that they serve as the optimization variables that indirectly affect constraint (3.37b) and consequently influence the objective function.

3.4.2 Construction of a Solvable Formulation

To address the optimization problem described earlier, several numerical components must be introduced. Initially, we need to establish a parameterization scheme to approximate the reward, Q-function, and the forward learning policy π. Subsequently, we should eliminate the norm between the unknown Q-functions by introducing a distance metric between the two policies, π and π^E. Additionally, we should replace the unavailable forward optimal Q-function $Q^*_{r,\gamma}$, which cannot be accessed during the IRL task, with a computable equivalent that can be obtained within the IRL context, denoted as $Q_N^{\pi^E}$, assuming access to a IRL budget of N samples (as opposed to M). Finally, we estimate the IRL Q-function $\widehat{Q}_N^{\pi^E}$ and define its associated confidence interval.

3.4.2.1 Parameterization

First, we introduce certain commonly accepted approximations. We assume the existence of feature vectors $\boldsymbol{\phi} : \mathcal{S} \times \mathcal{A} \to [0,1]^{d_\theta}$ and $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to [0,1]^{d_\omega}$. These vectors are respectively employed for linear parameterization of the reward function and the action-value function. In this parameterization, the reward function is represented as $r_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\phi}(s,a)^\top \boldsymbol{\theta}$, and the action-value function is expressed as $\widehat{Q}_{\boldsymbol{\omega}}(s,a;\boldsymbol{\omega}) = \boldsymbol{\psi}(s,a)^\top \boldsymbol{\omega}$, where $\boldsymbol{\theta} \in \mathbb{R}^{d_\theta}$ and $\boldsymbol{\omega} \in \mathbb{R}^{d_\omega}$. Additionally, our search for the greedy policy within the inner maximization is constrained to a class of parameterized policies denoted as $\Pi_{\boldsymbol{\eta}} = \{\pi_{\boldsymbol{\eta}} : \boldsymbol{\eta} \in \mathbb{R}^{d_\eta}\}$. The only constraints for policies within this class is that they must be differentiable with respect to the parameter vector $\boldsymbol{\eta}$. Instead, for $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, we require them to be linearly independent to guarantee the absence of redundant parameters and ensure that subsequent computations involve full-rank matrices.

3.4.2.2 Wasserstein Distance on Expert's Policy π^E

The resolution of the optimization problem (3.37) is not straightforward because it incorporates the expert's reward r^E and discount factor γ^E, which are inherently unknown. Therefore, we adopt a surrogate objective function and circumvent the value-function discrepancy by introducing a policy divergence. In particular, a result from [90, 92] provides an upper bound on the Q-function distance using policy divergence.

Theorem 3.3 (Rachelson and Lagoudakis [92])**.** *If the MDP is Lipschitz continuous with constants (L_r, L_P) and π is Lipschitz continuous with constant L_π. Then, it holds that:*

$$\left\| Q^{\pi^E}_{r^E,\gamma^E} - Q^{\pi}_{r^E,\gamma^E} \right\|_\mu \leq L_Q \mathop{\mathbb{E}}_{s \sim d^{\pi^E}_{\mu,\gamma^E}} [W_2(\pi^E(\cdot|s), \pi(\cdot|s))],$$

where $L_Q = \frac{\gamma^E L_r L_\pi}{(1-\gamma^E)(1-\gamma^E L_P(1+L_\pi))}$, W_2 is the L_2-Wasserstein distance and $d^{\pi^E}_{\mu,\gamma^E}$ is the γ^E-discounted state occupancy [106] induced by policy π^E and initial state distribution μ.

Given our focus on continuous action spaces and deterministic policies (typical of expert policies), the Wasserstein distance [117] emerges as a suitable choice for measuring distributional

divergence.[4] Formally, when provided with two deterministic policies $\pi_{\boldsymbol{\eta}}$ and π^E, as well as a state $s \in \mathcal{S}$, we can calculate the L_2-Wasserstein distance as follows:

$$W_2^2\big(\pi^E(s), \pi_{\boldsymbol{\eta}}(s)\big) = \big(\pi^E(s) - \pi_{\boldsymbol{\eta}}(s)\big)^2. \tag{3.38}$$

For the sake of clarity in our explanation, we focus on deterministic policies. However, it is important to note that the formulation, especially the Wasserstein distance, can be extended to accommodate stochastic policies.

We can observe that, thanks to theorem 3.3, we can eliminate the reliance on the expert's reward r^E and discount factor γ^E in the formulation (3.37). Henceforth, unless explicitly specified, all references to Q-functions will pertain to the optimized pair (r, γ), which will be omitted from the subscripts for compactness.

3.4.2.3 Dealing with the Forward Q-function $Q^\star_{r,\gamma}$

Eliminating the reliance on the expert's reward and discount factor in the definition of the objective function (3.37a) is an essential step, but it is not enough to convert the program (3.37) into a tractable one. The primary challenge that remains is that the confidence region associated with constraint (3.37b) includes an unknown variable, $Q^\star_{r,\gamma}$, which represents the optimal Q-function calculated using the optimized pair of parameters, (r, γ).

Replacing $Q^\star_{r,\gamma}$ with $Q^{\pi^E}_{r,\gamma}$ In theory, one could implement forward reinforcement learning as an internal process for each candidate pair, (r, γ), to generate an estimate of $Q^\star_{r,\gamma}$ and then use this approximation to satisfy constraint (3.37b), albeit in an imprecise manner. However, this approach is not feasible in our specific scenario, as we are limited to a fixed budget of only N samples available during IRL, and, more importantly, we aim to avoid the execution of forward RL altogether. As a solution, we substitute the optimal Q-function $Q^\star_{r,\gamma}$ in constraint (3.37b) with the expert Q-function $Q^{\pi^E}_{r,\gamma}$ to address this issue. The reason behind employing this approximation is that, when the values of (r, γ) are compatible with the expert's policy π^E, it follows that $Q^\star_{r,\gamma} = Q^{\pi^E}_{r,\gamma}$. This approximation offers a notable advantage in that it eliminates the necessity for forward RL, requiring only "policy evaluation". In essence, our task involves only estimating the expert's Q-function $Q^{\pi^E}_{r,\gamma}$ for each candidate pair (r, γ). Consequently, constraint (3.37b) is replaced with:

$$\left\| \widehat{Q}^{\pi^E}_M - Q^{\pi^E}_{r,\gamma} \right\| \leq \epsilon_1(M, \gamma), \tag{3.39}$$

where the $\epsilon_1(M, \gamma)$ depends on the policy evaluation algorithm employed to estimate $\widehat{Q}^{\pi^E}_M$ and the bound holds with probability at least $1 - \delta_1$. Finally, the policy evaluation task is easily performed leveraging the N samples available at IRL time.

Relaxing the Greedy Constraint A second challenge arises in relation to the calculation of the greedy policy for the inner maximization. To address this issue, we initiate the process by

[4] Other commonly used divergences, such as Total Variation or Kullback-Leibler, are ill-suited for deterministic distributions as they yield the maximum distance value in such cases.

explicitly expressing the greedy policy constraint within the inner maximization of the objective function (3.37a) and then proceed with two relaxation steps:

$$\begin{aligned} &\pi_{\boldsymbol{\eta}} \in \mathcal{G}\left[\widehat{Q}_M^{\pi^E}\right] \\ &\Rightarrow \widehat{Q}_M^{\pi^E}(s, \pi_{\boldsymbol{\eta}}(s)) \geq \widehat{Q}_M^{\pi^E}(s, \pi^E(s)) \quad \forall s \in \mathcal{S} \\ &\Rightarrow \sum_{s \in \mathcal{D}_{\text{IRL}}} \widehat{Q}_M^{\pi^E}(s, \pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi^E}(s, \pi^E(s)) \geq 0. \end{aligned} \tag{3.40}$$

The initial relaxation entails transitioning from a strict greedy policy to all policies that exhibit at least a *performance improvement*, characterized by a positive advantage. This adaptation ensures that the learner policy, denoted as $\pi_{\boldsymbol{\eta}}$, explicitly relies on $\widehat{Q}_M^{\pi^E}$. The second relaxation involves the constraint being applicable on average across a finite selection of states, denoted as $\mathcal{D}_{\text{IRL}} \subseteq \mathcal{S}$. This adjustment is made because enforcing such a constraint in an infinite state space is impractical (a similar relaxation is also applied, for example, in [98] for the KL-divergence).

Enforcing Constraint in equation (3.39) Finally, to precisely define constraint (3.39), we illustrate the process of deriving a more relaxed version, in conjunction with the policy improvement inequality (3.40), aiming to eliminate the reliance on the expert's Q-function $Q_{r,\gamma}^{\pi^E}$ in problem (3.37). The concept revolves around formulating a less strict constraint than (3.39) that doesn't involve the unknown Q-function $Q_{r,\gamma}^{\pi^E}$.

Proposition 3.4. *Suppose that, simultaneously for all $r \in \mathcal{R}$ and $\gamma \in [0, 1)$, we have that the Q-function $\widehat{Q}_M^{\pi^E}$ is known within some accuracy level $\epsilon_1(M, \gamma)$ (with probability $1 - \delta_1$), and let us introduce a new Q-function $\widehat{Q}_N^{\pi^E}$ which can be estimated with N samples during the IRL task within some accuracy $\epsilon_2(N, \gamma)$ (with probability $1 - \delta_2$), i.e.:*

$$\left\| \widehat{Q}_M^{\pi^E} - Q_{r,\gamma}^{\pi^E} \right\|_\infty \leq \epsilon_1(M, \gamma), \tag{3.41}$$

$$\left\| \widehat{Q}_N^{\pi^E} - Q_{r,\gamma}^{\pi^E} \right\|_\infty \leq \epsilon_2(N, \gamma). \tag{3.42}$$

Then, with probability at least $1 - \delta_1 - \delta_2$, the inequality (3.40) can be upper bounded by dropping the dependence on $\widehat{Q}_M^{\pi^E}$, obtaining the following inequality:

$$\sum_{s \in \mathcal{D}_{IRL}} \widehat{Q}_N^{\pi^E}(s, \pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi^E}(s, \pi^E(s)) + 2\epsilon_1(M, \gamma) + 2\epsilon_2(N, \gamma) \geq 0. \tag{3.43}$$

Proof. The concept presented in this proposition starts with the constraint (3.40):

$$\sum_{s \in \mathcal{S}} \widehat{Q}_M^{\pi^E}(s, \pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi^E}(s, \pi^E(s)) \geq 0, \tag{3.44}$$

which includes the unknown quantity $\widehat{Q}_M^{\pi^E}$. We now compute a new looser inequality involving only the known quantity $\widehat{Q}_N^{\pi^E}$. To do so, we need to take a larger left hand side by considering an upper bound to $\widehat{Q}_M^{\pi^E}(s, \pi_{\boldsymbol{\eta}}(s))$ and a lower bound to $\widehat{Q}_M^{\pi^E}(s, \pi^E(s))$. Starting with the former,

we can write:

$$\widehat{Q}_M^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) = \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) + \left(\widehat{Q}^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s))\right) + \tag{3.45}$$
$$+ \left(\widehat{Q}_M^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s))\right)$$
$$\leq \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) + \left|\widehat{Q}^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s))\right| + \tag{3.46}$$
$$+ \left|\widehat{Q}_M^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s))\right|$$
$$\leq \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) + \epsilon_2(N,\gamma) + \epsilon_1(M,\gamma), \tag{3.47}$$

where in the last step we applied assumptions in equations (3.41) and (3.42). Similarly, we can derive:

$$\widehat{Q}_M^{\pi^E}(s,\pi^E(s)) = \widehat{Q}_N^{\pi^E}(s,\pi^E(s)) + \left(\widehat{Q}^{\pi^E}(s,\pi^E(s)) - \widehat{Q}_N^{\pi^E}(s,\pi^E(s))\right) + \tag{3.48}$$
$$+ \left(\widehat{Q}_M^{\pi^E}(s,\pi^E(s)) - \widehat{Q}^{\pi^E}(s,\pi^E(s))\right)$$
$$\geq \widehat{Q}_N^{\pi^E}(s,\pi^E(s)) - \left|\widehat{Q}^{\pi^E}(s,\pi^E(s)) - \widehat{Q}_N^{\pi^E}(s,\pi^E(s))\right| + \tag{3.49}$$
$$- \left|\widehat{Q}_M^{\pi^E}(s,\pi^E(s)) - \widehat{Q}^{\pi^E}(s,\pi^E(s))\right|$$
$$\geq \widehat{Q}_N^{\pi^E}(s,\pi^E(s)) - \epsilon_2(N,\gamma) - \epsilon_1(M,\gamma), \tag{3.50}$$

where again in the last step we applied assumptions in equations (3.41) and (3.42). Putting back together upper and lower bounds we obtain:

$$\sum_{s\in\mathcal{S}} \widehat{Q}_M^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_M^{\pi^E}(s,\pi^E(s)) \leq \sum_{s\in\mathcal{S}} \widehat{Q}_N^{\pi^E}(s,\pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi^E}(s,\pi^E(s)) + 2\epsilon_2(N,\gamma) + 2\epsilon_1(M,\gamma). \tag{3.51}$$

If the original constraint is satisfied, also the looser one holds. □

Regarding the parameters $\epsilon_1(M,\gamma)$ and $\epsilon_2(N,\gamma)$, we now consider their general structures as follows:

$$\epsilon_1(M,\gamma) = \frac{\gamma c_1}{(1-\gamma)\sqrt{M}},\ \ \epsilon_2(N,\gamma) = \frac{\gamma c_2}{(1-\gamma)\sqrt{N}}, \tag{3.52}$$

These structures generalize the majority of sample complexity bounds found in the existing literature, often converging asymptotically to a power of $1/(1-\gamma)$ [9, 28, 65, 83]. The constants c_1 and c_2 represent suitable factors, and they are determined through detailed finite-sample analysis. Typically, these constants encapsulate the intrinsic features of the underlying MDP, in conjunction with the selection of a specific estimation algorithm.

We remark that the assumption (3.41) essentially mirrors the constraint (3.39) itself when instantiated with the L_∞-norm. This direct instantiation allows us to establish the new relation solely involving $\widehat{Q}_N^{\pi^E}$. Notably, inequality (3.43) involves all the uncertainties associated with both $\widehat{Q}_M^{\pi^E}$ and $\widehat{Q}_N^{\pi^E}$. It relies on the outer optimization variables, namely γ (influenced by parameters ϵ_1 and ϵ_2 and the Q-function $\widehat{Q}_N^{\pi^E}$), and r (via the Q-function $\widehat{Q}_N^{\pi^E}$), as well as the number of samples N available for the IRL task, as opposed to the number of samples M for the forward RL problem. The dependency on the policy $\pi_{\boldsymbol{\eta}}$ in (3.43) is, however, straightforward.

3.4.2.4 Expert's Policy Evaluation with $\widehat{Q}_N^{\pi^E}$

The last step in our development of a solvable IRL formulation involves the estimation of the new Q-function $\widehat{Q}_N^{\pi^E}$. While, in theory, any policy evaluation technique could be applied for this task, we have chosen to utilize the Least-Squares Temporal Difference (LSTDQ) algorithm, as it provides a confidence region, as expressed in the form (3.42), through finite-sample analysis. As described in section 2.2.2.1, LSTDQ returns the parameter vector $\boldsymbol{\omega} \in \mathbb{R}^{d_\omega}$ that minimizes the mean-squared projected Bellman error (MSPBE), equation (2.31). We can rewrite in a more coincise way:

$$\text{MSPBE}(\boldsymbol{\omega}) \triangleq \left\| \widehat{Q}_{\boldsymbol{\omega}} - \text{proj}_\Psi \, T^{\pi^E} \widehat{Q}_{\boldsymbol{\omega}} \right\|_{\rho^{\pi^E}}^2, \tag{3.53}$$

where proj_Ψ is the projection operator onto the linear space Ψ spanned by the basis functions ψ, $\text{proj}_\Psi f = \arg\min_{\boldsymbol{\omega}} \left\| \boldsymbol{\psi}^\top \boldsymbol{\omega} - f \right\|^2$, and ρ^{π^E} is the stationary distribution induced by π^E. As a result of the linear parameterization utilized, the projection operator also assumes a linear form. It can be represented by the matrix expression $\boldsymbol{\Psi}(\boldsymbol{\Psi}^\top \boldsymbol{\Psi})^{-1} \boldsymbol{\Psi}^\top$, where $\boldsymbol{\Psi} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}| \times d_\omega}$ is a matrix consisting of feature vectors $\boldsymbol{\psi}(s,a)^\top$ as its rows, for each $(s,a) \in \mathcal{S} \times \mathcal{A}$.

LSTDQ operates with a batch of transitions, usually collected off-policy, denoted as $\mathcal{D}_{\text{LSTD}} = (s_i, a_i, r_i, s'i)_{i=1}^N$. In this context, $s'_i \sim P(\cdot|s_i, a_i)$ represents the subsequent state, and $r_i = \boldsymbol{\phi}(s_i, a_i)^\top \boldsymbol{\theta}$ corresponds to the reward. The objective here is to numerically evaluate the Bellman operator, thereby avoiding the need for knowledge about the underlying MDP. Additionally, it aims to sample the complete feature matrix $\boldsymbol{\Psi}$, which is typically unattainable in a continuous environment. In our IRL scenario, the dataset $\mathcal{D}_{\text{LSTD}}$ can either be the same as the one used for IRL resolution, which is $\mathcal{D}_{\text{IRL}}$, or, if accessible, a freshly gathered dataset.

As anticipated in section 3.4.2.4, the choice of LSTDQ allows us to provide a specific confidence region for $\widehat{Q}_N^{\pi^E}$ of the form in equation (3.42). A valuable contribution in this regard is presented in theorem 5 of [65], where the authors conducted a finite-sample analysis of the LSTD algorithm.

Below we rephrase this theorem to provide a bound on the prediction error of the LSTDQ solution $\widehat{Q}_{\widehat{\boldsymbol{\omega}}} = \widehat{Q}_N^{\pi^E}(s,a;\widehat{\boldsymbol{\omega}})$ w.r.t. the true value function $Q_{r,\gamma}^{\pi^E}(s,a)$. This theorem is applicable under the condition that, with a probability of $1-\delta$, the sample-based Gram matrix $\frac{1}{N}\tilde{\boldsymbol{\Psi}}^\top \tilde{\boldsymbol{\Psi}}$ is invertible, and its smallest eigenvalue ν_N is positive. This condition is guaranteed if the number of samples N meets the requirement (up to constant and logarithmic factors) $N \geq \tilde{\mathcal{O}}\left(288 L^2 / \nu (d+1) \log(N/\delta)^2\right)$, where ν represents the smallest eigenvalue of the exact Gram matrix (we refer to lemma 4 of [65] for further details).

Theorem 3.5 (Lazaric et al. [65])**.** *Let $(s_1, a_1), \ldots, (s_N, a_N)$ be a path generated by a stationary β-mixing process with parameters $\bar{\beta}, b, \kappa$ (that is, its β-mixing coefficients satisfy $\beta_i \leq \bar{\beta}\exp(-bi^\kappa)$) and stationary distribution ρ^{π^E}. With probability at least $1-\delta_2$ we have:*

$$\begin{aligned} \left\| Q_{r,\gamma}^{\pi^E} - \widehat{Q}_N^{\pi^E} \right\|_{\rho^{\pi^E}} \leq \epsilon_2(N,\gamma) = \\ \frac{2}{\sqrt{1-\gamma^2}} \left(2\sqrt{2} \left\| Q_{r,\gamma}^{\pi^E} - \text{proj}_\Psi \, Q_{r,\gamma}^{\pi^E} \right\|_\rho + \epsilon^{(1)} \right) + \epsilon^{(2)} \\ + \frac{2}{1-\gamma} \left[\gamma Q_{\max} L \sqrt{\frac{d}{\nu}} \left(\sqrt{\frac{8 \log(8K/\delta_N)}{N}} + \frac{1}{N} \right) \right], \end{aligned} \tag{3.54}$$

with $\epsilon^{(1)}$ *and* $\epsilon^{(2)}$ *that are* $\mathcal{O}(Q_{\max}/\sqrt{N})$*, where* $Q_{\max} = \|r\|_\infty/(1-\gamma)$*, and* L *is the upper bound of each basis function* ψ_j *of a feature vector* $\boldsymbol{\psi}$*, i.e.* $\|\psi_j\|_\infty \leq L$*.*

3.4.2.5 Optimization Algorithm

After presenting all the essential components for establishing our new IRL formulation, we now delve into the optimization algorithm used to tackle the min-max optimization problem. Our first step involves recasting the ultimate optimization problem with respect to the optimization variables $(\boldsymbol{\theta}, \gamma, \boldsymbol{\eta})$ as follows:

$$\begin{aligned} \min_{\substack{\boldsymbol{\theta}\in\mathbb{R}^{d_\theta}\\ \gamma\in[0,1)}} \max_{\boldsymbol{\eta}\in\mathbb{R}^{d_\eta}} & \underbrace{\sum_{s\in\mathcal{D}_{\text{IRL}}} W_2\big(\pi^E(s), \pi_{\boldsymbol{\eta}}(s)\big)}_{\triangleq f(\boldsymbol{\eta})} \\ \text{s.t.} & \underbrace{\sum_{s\in\mathcal{D}_{\text{IRL}}} \widehat{Q}_N^{\pi_{\boldsymbol{\eta}}^E}(s, \pi_{\boldsymbol{\eta}}(s)) - \widehat{Q}_N^{\pi_{\boldsymbol{\eta}}^E}(s, \pi^E(s)) + 2\epsilon_M + 2\epsilon_N \geq 0}_{\triangleq -g(\boldsymbol{\theta},\gamma,\boldsymbol{\eta})}, \end{aligned} \tag{3.55}$$

In this context, the dataset $\mathcal{D}_{\text{IRL}}$ is employed for the sample-based approximation of the Wasserstein distance $f(\cdot)$ and for the constraint $g(\cdot)$.

In cases where min-max problems exhibit concavity in the inner variables (i.e., $\boldsymbol{\eta}$) and convexity in the outer variables (i.e., $\boldsymbol{\theta}, \gamma$), numerous algorithms have been introduced in the literature to address such scenarios [122]. However, solving problems like (3.55) can prove exceedingly challenging in a non-convex context, where established optimization algorithms are lacking. For instance, a straightforward extension of a gradient-like descent-ascent algorithm to the min-max setting may readily fail to converge to a meaningful solution [94].

In this context, we look at the min-max optimization as a competitive game involving two players and aim to discover a stationary solution to the problem. In accordance with [94], we reconfigure problem (3.55) using the potential function $F(\boldsymbol{\theta}, \gamma) \triangleq \max_{\boldsymbol{\eta}\in\mathbb{R}^{d_\eta}: g(\boldsymbol{\eta},\boldsymbol{\theta},\gamma)\leq 0} f(\boldsymbol{\eta})$, resulting in:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d_\theta},\, \gamma\in[0,1)} F(\boldsymbol{\theta}, \gamma).$$

Assuming the existence of an implicit function $\boldsymbol{\eta}^\star(\boldsymbol{\theta}, \gamma) \triangleq \arg\max_{\boldsymbol{\eta}\in\mathbb{R}^{d_\eta}: g(\boldsymbol{\eta},\boldsymbol{\theta},\gamma)\leq 0} f(\boldsymbol{\eta})$, we can calculate the gradient of $F(\cdot)$ using the chain rule in the following manner: $\boldsymbol{\nabla}_{\boldsymbol{\theta},\gamma} F(\boldsymbol{\theta}, \gamma) = \boldsymbol{\nabla}_{\boldsymbol{\theta},\gamma} f(\boldsymbol{\eta}^\star(\boldsymbol{\theta}, \gamma)) = \boldsymbol{\nabla}_{\boldsymbol{\theta},\gamma} \boldsymbol{\eta}^\star(\boldsymbol{\theta}, \gamma) \boldsymbol{\nabla}_{\boldsymbol{\eta}} f(\boldsymbol{\eta})|_{\boldsymbol{\eta}=\boldsymbol{\eta}^\star(\boldsymbol{\theta},\gamma)}$ [23].

It is important to exercise caution in this context because $\boldsymbol{\eta}^\star$ is an implicit function of $(\boldsymbol{\theta}, \gamma)$, as it is determined by satisfying the constraint $g(\cdot) = 0$. Instead of employing partial derivatives, we should use the following selection of total differential forms:

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial f}{\partial \boldsymbol{\eta}^\star} \frac{\mathrm{d}\boldsymbol{\eta}^\star}{\mathrm{d}\boldsymbol{\theta}}, \quad \text{with} \quad \frac{\mathrm{d}\boldsymbol{\eta}^\star}{\mathrm{d}\boldsymbol{\theta}} = -\frac{\partial g}{\partial \boldsymbol{\theta}} \Big/ \frac{\partial g}{\partial \boldsymbol{\eta}^\star}, \tag{3.56a}$$

$$\frac{\mathrm{d}F}{\mathrm{d}\gamma} = \frac{\partial f}{\partial \boldsymbol{\eta}^\star} \frac{\mathrm{d}\boldsymbol{\eta}^\star}{\mathrm{d}\gamma}, \quad \text{with} \quad \frac{\mathrm{d}\boldsymbol{\eta}^\star}{\mathrm{d}\gamma} = -\frac{\partial g}{\partial \gamma} \Big/ \frac{\partial g}{\partial \boldsymbol{\eta}^\star}, \tag{3.56b}$$

wherethe differentials and divisions apply on a component-wise basis, and the differentiation of $\boldsymbol{\eta}$ is determined by the application of the *implicit function theorem* [58] to the equation $g(\cdot) = 0$. We can now finally solve problem (3.55) by running the following iterative procedure, for $t \in \mathbb{N}$:

$$\boldsymbol{\eta}_{t+1} = \underset{\boldsymbol{\eta} \in \mathbb{R}^{d_\eta} : g(\boldsymbol{\eta}, \boldsymbol{\theta}, \gamma) \leq 0}{\arg\max} f(\boldsymbol{\eta}), \tag{3.57a}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_\theta \left. \frac{\mathrm{d}F(\boldsymbol{\eta}_{t+1}, \boldsymbol{\theta}, \gamma_t)}{\mathrm{d}\boldsymbol{\theta}} \right|_{\boldsymbol{\theta} = \boldsymbol{\theta}_t},$$

$$\gamma_{t+1} = \mathrm{proj}_{[0,1)} \left[\gamma_t - \alpha_\gamma \left. \frac{\mathrm{d}F(\boldsymbol{\eta}_{t+1}, \boldsymbol{\theta}_t, \gamma)}{\mathrm{d}\gamma} \right|_{\gamma = \gamma_t} \right],$$

where $\alpha_\theta > 0$ and $\alpha_\gamma > 0$ are suitable learning rates.

The previous algorithm would necessitate finding the exact solution to the maximization problem in (3.57a). However, this task can be computationally challenging if the function $f(\cdot)$ is not concave. Fortunately, we can replace the exact computation of the point $\boldsymbol{\eta}_{r+1}$ with an approximation, ensuring that it satisfies the condition $f(\boldsymbol{\eta}_{r+1}) \geq \max_{\boldsymbol{\eta} \in \mathbb{R}^{d_\eta} : g(\boldsymbol{\eta}, \boldsymbol{\theta}, \gamma) \leq 0} f(\boldsymbol{\eta}) - \epsilon'$, thus relaxing the concavity assumption. In this scenario, the algorithm is guaranteed [94] to find an approximate stationary point (given appropriate choices for α_θ and α_γ), with the level of accuracy determined by the value of $\epsilon' > 0$.

3.4.3 Experiments

As a proof of concept of the need for a reward function that is aware of the subsequent forward RL phase, we present three experiments in the LQG context [24]. Two of them takes as case study the simplest scalar problem (in which all the dynamics' matrices are scalars equal to 1). We have opted to use this simple problem as a first benchmark because the availability of a closed form expression for the control gain can work as a fundamental reference to compare our results with. Nevertheless, we also propose a slightly more complex LQG setting in order to validate our approach in a more "realistic" environment in which we control the pitch of the longitudinal axis of an aircraft modelled by a 3-dimensional LQG problem [118]. Finally an additional experiment has been conducted in the Mountain Car domain [81] to prove the generality of the method in a different context.

3.4.3.1 Experiments on Scalar LQG

We now examine a scalar Linear Quadratic Gaussian (LQG) problem (refer to appendix A.1), in which all matrices are constants and scalars ($A = B = Q = R = 1$). Consequently, the reward function in the environment (kept hidden during the IRL process) can be expressed as: $r^E(s, a) = -s^2 - a^2$. We calculate the parameter for the expert policy π^E in a closed-form, making use of Riccati equation (A.7). This parameter is optimized for the aforementioned reward

function and a discount factor of $\gamma^E = 0.9$, thus providing the expert's policy:

$$\pi^E(s) \approx \underbrace{-0.58}_{\eta^*} s. \tag{3.58}$$

To perform a linear estimation of the Q-function, we employ a feature vector $\boldsymbol{\psi}(s,a) = [s^2, a^2, sa]$ in order to cover the space of the exact quadratic Q-function $Q^{\pi^E}_{r^E,\gamma^E}$. Meanwhile, the reward features are defined as $\boldsymbol{\phi}(s,a) = [-s^2 - a^2, Q^{\pi^E}_{\bar{s}}(s,a)]$. Here, $Q^{\pi^E}_{\bar{s}}$ denotes the expert's Q-function in a shifted Linear Quadratic Gaussian (LQG) problem, which is designed to stabilize the state at $\bar{s} \neq 0$ (meaning the expert is optimal with respect to the reward $r_{\bar{s}}(s,a) = -(s-\bar{s})^2 - (a-\bar{a})^2$, with $\bar{a}$ being the equilibrium control action corresponding to $\bar{s}$). Finally, for these experiments, the policy of the forward learning agent is parameterized linearly in the state as $\pi_\eta(s) = \eta s$, and the reward weights $\boldsymbol{\theta}$ are normalized to sum to 1.

Reward analysis for *infinite* number of samples. Our approach relies on two datasets: one for the estimation of $\widehat{Q}^{\pi^E}_N$, denoted as $\mathcal{D}_{\text{LSTD}}$, and another for the resolution of the IRL formulation, referred to as $\mathcal{D}_{\text{IRL}}$. $\mathcal{D}_{\text{LSTD}}$ is created by initiating from 40 uniformly sampled states within the range of $[-1, 1]$, and by following the expert policy for $H = 5$ steps, with the expert's actions being perturbed by white noise $v \sim \mathcal{N}(0, 0.05)$. On the other hand, $\mathcal{D}_{\text{IRL}}$ is comprised of 200 states sampled randomly from the interval $[-1, 1]$, making $N = 200$. It is assumed that there are an infinite number of samples available for solving the forward learning problem, and thus M is set to ∞. This assumption is made with the expectation that a sufficiently high number of samples in the forward learning phase will approach asymptotic behavior $\epsilon(M, \gamma) \to 0$. Furthermore, the parameters M and N can be interchanged numerically, and for the sake of simplicity in figures 3.1 and 3.3, we have set a specific value for N while allowing M to be ∞.

For a comprehensive numerical analysis of the novel min-max formulation, we illustrate in figure 3.1 how the maximum Wasserstein distance value $f(\boldsymbol{\eta}^\star)$ in (3.55) changes with variations in the discount factor γ and the reward weights $\boldsymbol{\theta}$ for $r_{\boldsymbol{\theta}}$[5]. Different images refers to increasing values of the shifted goal state $\bar{s}$. As expected, when $\bar{s} = 0$, the formulation identifies the optimal min-max solution as $\gamma^\star = 0$, thereby minimizing the sample complexity for the forward learning phase. Additionally, it leads to $r_{\boldsymbol{\theta}^\star} = Q^{\pi^E}_{\bar{s}}$, which mirrors the expert's reward r^E. Interestingly, as the goal $\bar{s}$ transitions from 0 (the expert's goal) to a higher value, the formulation strategically balances the sample complexity resulting from a higher γ against the error introduced in the learned policy when opting for a sub-optimal reward. It moves towards selecting the unbiased expert reward, with $\gamma^\star = \gamma^E$ and $r_{\boldsymbol{\theta}^\star} = r^E$ when the goal $\bar{s} = 0.4$ significantly deviates (sub-optimally) from the expert's goal of 0.

Reward analysis for *finite* number of samples. To emphasize the impact of utilizing a potentially suboptimal reward in scenarios where the forward RL phase has limited available samples, we devised an additional experiment in the scalar LQG setting. Specifically, we examined two reward functions learned in prior experiments: $-s^2 - a^2$ and $Q^{\pi^E}_0(s,a)$. In figure 3.2, we present the learned parameters (top row) and the average discounted return (bottom row) achieved through RL using the REINFORCE algorithm [126] in two distinct LQG environments.

[5] In figure 3.1, only one component θ_0 of the weights is plotted. The second component is determined as: $\theta_1 = 1 - \theta_0$

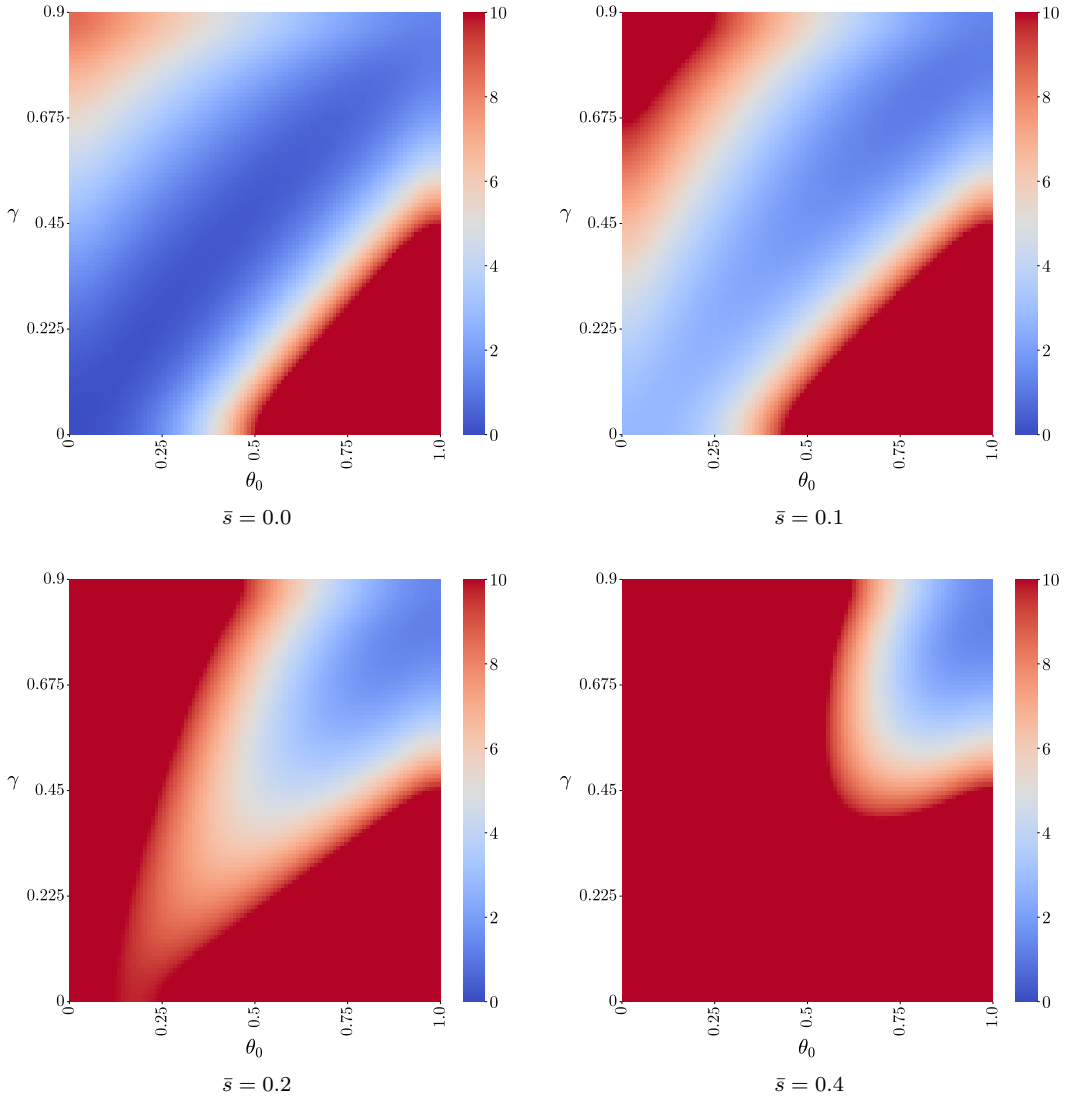


FIG. 3.1: Value of the objective function $f(\boldsymbol{\eta}^\star)$ in (3.55) related to the change of the outer variables $(\gamma, \boldsymbol{\theta})$, with $N = 200, c_1 = 0.01$ and $M = \infty$. Each plot refers to different values of the goal $\bar{s}$.

On the left, we explore the same environment used during the IRL phase. On the right, we consider an altered LQG environment with a modified dynamical matrix (scaled by 0.85 compared to the original setup). As expected, on the left, both reward functions, $Q_0^{\pi^E}(s,a)$ (*Controller with IRL reward*) and $-s^2 - a^2$ (*Controller with Real reward*), effectively recover the optimal parameter. However, $Q_0^{\pi^E}(s,a)$ requires a smaller number of samples. This behavior is observed on the right. While the original reward function $-s^2 - a^2$ successfully retrieves the correct parameter, the *biased* reward $Q_0^{\pi^E}(s,a)$, learned in a different environment, proves to be more effective in achieving reasonable performance when the number of available samples is limited. Clearly, as the sample size increases, the impact of bias becomes more prominent.

The impact of utilizing the optimized IRL reward on the sample complexity of the forward learning problem is also illustrated in figure 3.3. After solving problem (3.55) with parameters $N = 200, c_1 = 0.01, M = \infty, \bar{s} = 0$, we employ the resulting pair $(\gamma^\star, \boldsymbol{\theta}^\star)$ to learn the optimal policy parameters while varying the number of available samples. Specifically, we randomly select

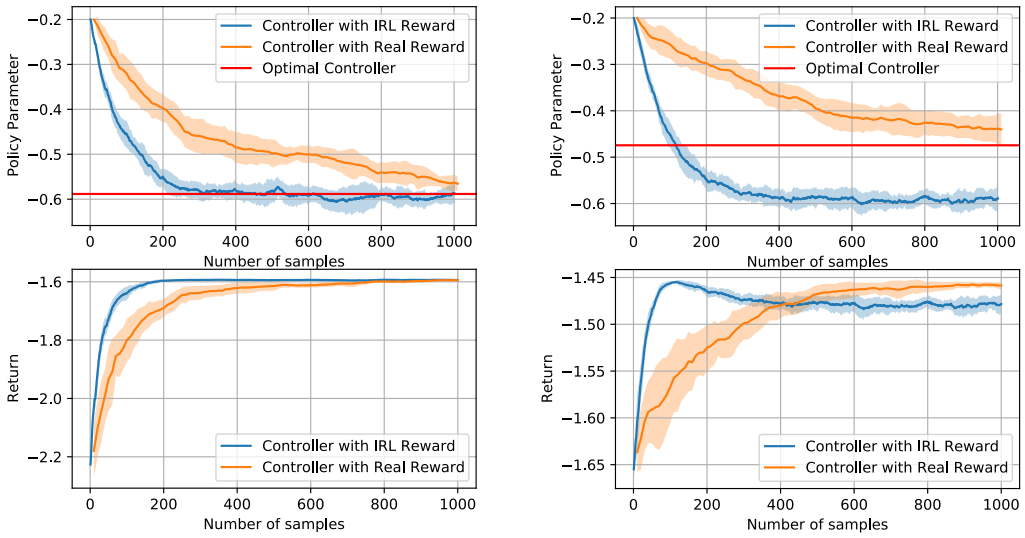


FIG. 3.2: Comparison of the learned policy parameter and average return when learning in the same environment used for IRL (left) and when changing the environment (right) (10 runs, 95% c.i.).

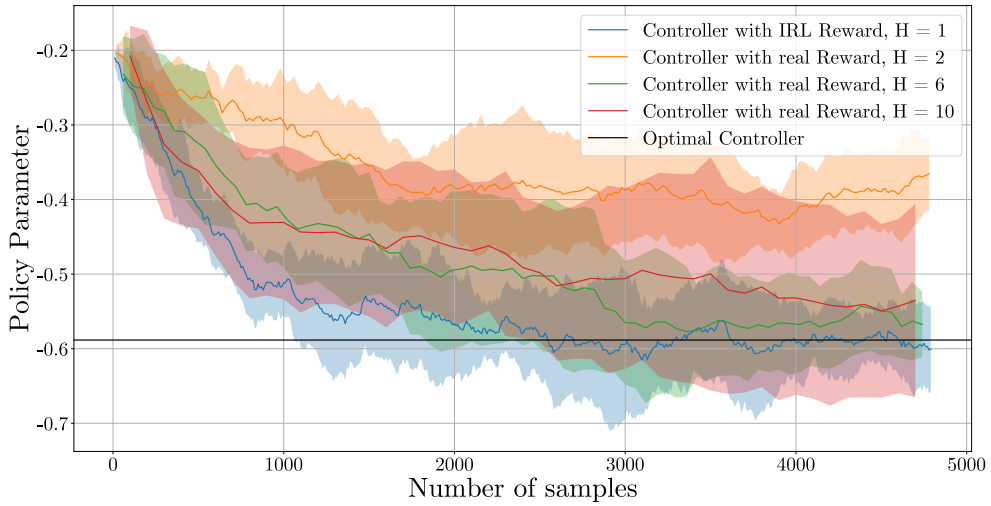


FIG. 3.3: Impact of the optimized IRL reward on the sample efficiency of the forward learning task, and convergence to the expert's policy parameter (10 runs, 95% c.i.).

20 initial states, and then estimate the gradient direction within the REINFORCE algorithm [126] through a Monte Carlo evaluation of the reward along trajectories of different lengths, H. We utilize $H = 1$ with the IRL reward and $H \in 2, 6, 10$ with the real reward. The plot clearly demonstrates how the IRL reward and discount factor enable the RL algorithm to converge to the optimal policy parameters much more quickly than when using the LQG reward. In other words, the number of samples required during the learning process is significantly reduced when our proposed IRL formulation's solution is employed in place of the expert's (and exact) one (γ^E, r^E).

3.4.3.2 Experiment on Multi-dimensional LQG

In order to prove our approach in a more complex environment, we employed it in a Pitch Control of an Aircraft modeled as a 3-dimensional LQG problem.

Pitch control is an integral part of the flight control system of an airplane. It is used to control the angle of the airplane's nose relative to the horizon, and is a key factor in maintaining level flight and maneuverability. It is usually accomplished through the use of either ailerons or elevators. Ailerons are typically placed on the trailing edge of the wing, and are used to move the wing up or down. Elevators are usually located on the tail of the aircraft, and are used to control the pitch of the aircraft by moving the tail up or down. In this context, we consider the model of an aircraft equipped only with the tail elevator.

Wahid and Rahmat [118] derived a continuous LQG problem of the control under some specific assumption. They assumed that the aircraft is initially in a state of equilibrium, maintaining a constant altitude and velocity, with the thrust and drag forces canceling out, and the lift and weight forces being in equilibrium. Furthermore, they suppose the alteration of the pitch angle of the aircraft does not affect its speed.

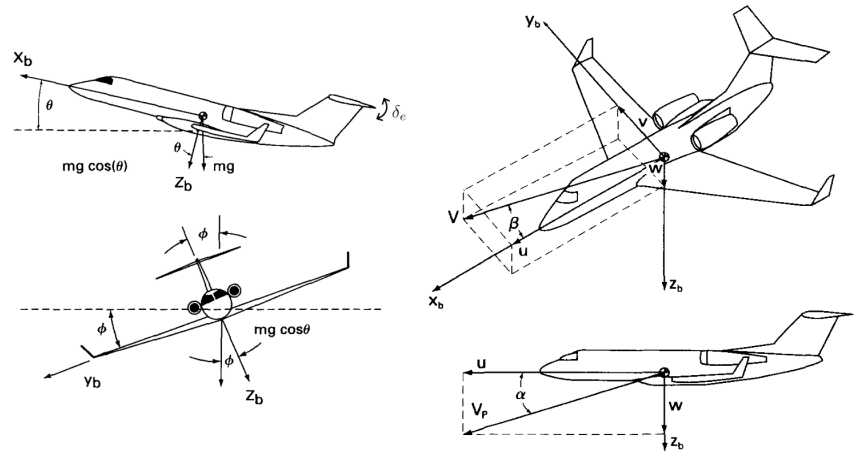


FIG. 3.4: Visual description of the Pitch Control System

Under these circumstances, authors managed to derive state-space representation of the system in which the state is represented by $s_t = [\Delta\alpha_t, \Delta q_t, \Delta\theta_t]^\top$, where α_t is the angle of attack (i.e. the aircraft orientation with respect to the air-relative velocity), θ_t is the pitch angle and q_t is the pitching rate (i.e. the derivative of the pitching angle w.r.t. the time). Finally, $a_t = [\Delta\delta_{et}]$ is the control input representing the elevator angle of the aircraft's tail. Here Δ represents a perturbation of each variable according to the disturbance theory that authors adopted to linearize the system.

The final continuous state-space representation derived is given by:

$$\begin{bmatrix} \Delta\dot{\alpha}_t \\ \Delta\dot{q}_t \\ \Delta\dot{\theta}_t \end{bmatrix} = \underbrace{\begin{bmatrix} -2.02 & 0 & 1 \\ -6.9868 & 2.9476 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{A} \begin{bmatrix} \Delta\alpha_t \\ \Delta q_t \\ \Delta\theta_t \end{bmatrix} + \underbrace{\begin{bmatrix} 0.16 \\ 11.7304 \\ 0 \end{bmatrix}}_{B} \Delta\delta_{et} \tag{3.59}$$

Finally they defined the LQG matrices as:

$$Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 500 \end{bmatrix} \quad R = 1 \tag{3.60}$$

The goal here is to stabilize the pitch during the flight in order to have the nose of the plane parallel to the air-relative velocity and to the ground: $s_{\text{goal}} = [0, 0, 0]^\top$.

As a first step we discretize the system dynamic with a sampling period $T = 1\ s$ obtaining the following matrices for system:

$$A_d = \begin{bmatrix} -0.0639 & 0.0165 & 0 \\ -0.1150 & -0.0792 & 0 \\ -0.5565 & 0.1773 & 1 \end{bmatrix} \quad B_d = \begin{bmatrix} 0.9745 \\ 1.9913 \\ 1.9599 \end{bmatrix} \tag{3.61}$$

To show how the sample budget affects the performance of the forward RL process, we consider two distincts reward functions and use them as features:

$$r_0(s, a) = -s^\top Q s - a^\top R a \tag{3.62}$$

$$r_1(s, a) = Q_{\bar{s}}^{\pi^E}(s, a) \tag{3.63}$$

$$\boldsymbol{\phi}(s, a) = [r_0(s, a), r_1(s, a)]^\top \tag{3.64}$$

Where $Q_{\bar{s}}^{\pi^E}(s, a)$ represents the Q-function of the expert in the shifted pitch-control problem where the agent aims to stabilize the plane to the state $\bar{s} = [0.7854, 0, 0.5236]^\top$ representing the plane in a constant angle of attack and pitch angle. In the following, we refer to equations (3.62) and (3.64) respectively as "Real Reward" and "IRL reward".

As for the scalar LQG experiments, we adopted REINFORCE to underline the effect of employing a possibly suboptimal reward when the forward RL phase has a limited sample budget for the forward learning. As we can see from figure 3.5, although $r_0(s, a)$ leads to higher cumulative rewards in the long run, the *biased* reward $r_1(s, a)$, acquired from a distinct environment, demonstrates greater effectiveness in achieving satisfactory performance, especially when the sample size is limited (less than 16000). As for the scalar case, as the number of samples grows, the influence of bias becomes increasingly significant.

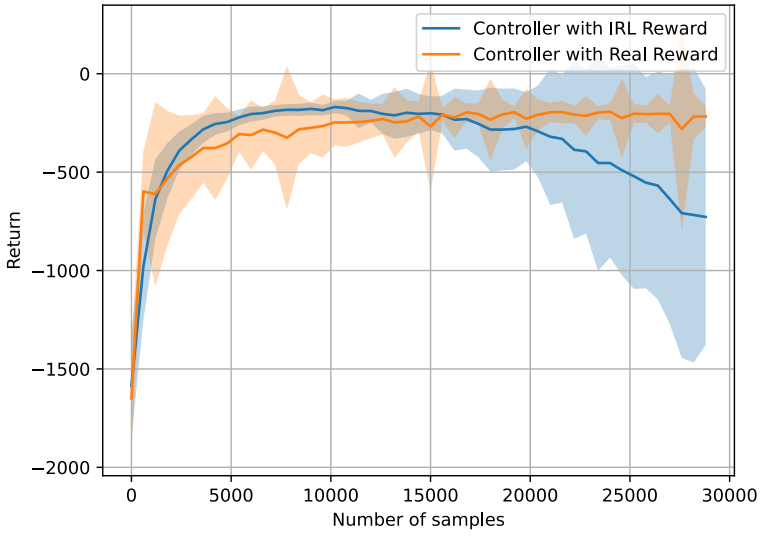


FIG. 3.5: Comparison of the effect of two different reward functions on the sample complexity of REINFORCE on the Pitch Control problem (5 runs, 95% c.i.).

3.4.3.3 Experiment on Mountain Car

To further emphasize the importance of having a reward function that accounts for the available budget in the forward RL phase, we present an additional experiment in the Mountain Car domain (refer to appendix A.2). Similar to the LQG cases, we explore two different reward functions:

$$r_0(s,a) = 200 \cdot \mathbf{1}_{[s=\text{goal}]}(s) - a^2 \tag{3.65}$$

$$r_1(s,a) = -(a - \tilde{\pi}(s))^2 \tag{3.66}$$

$$\boldsymbol{\phi}(s,a) = [r_0(s,a), r_1(s,a)]^\top \tag{3.67}$$

We call "Real Reward" the former reward function, an almost *sparse* reward recompensing the agent when reaching the goal while penalizing large actions (essentially a slighly changed version of the environment reward described in A.2). Similarly to LQG cases, we also call "IRL Reward" the latter reward function, a *dense* reward that induces the agent to *mimic* a sub-optimal policy $\tilde{\pi}$, obtained by perturbing the parameters of the optimal policy.

While the "Real Reward", leading to the optimal policy, requires a large γ (because of the action penalization $-a^2$) and it is preferred by our IRL objective when many samples are available (i.e. large value for M), the "IRL Reward" leads to a sub-optimal policy but it is very easy to learn (aiming to imitate $\tilde{\pi}$), admitting very small γ, and it is preferred for small values of M. Figure 3.6 shows the forward learning results obtained by REINFORCE on the two rewards and confirms the properties discussed in the previous subsections.

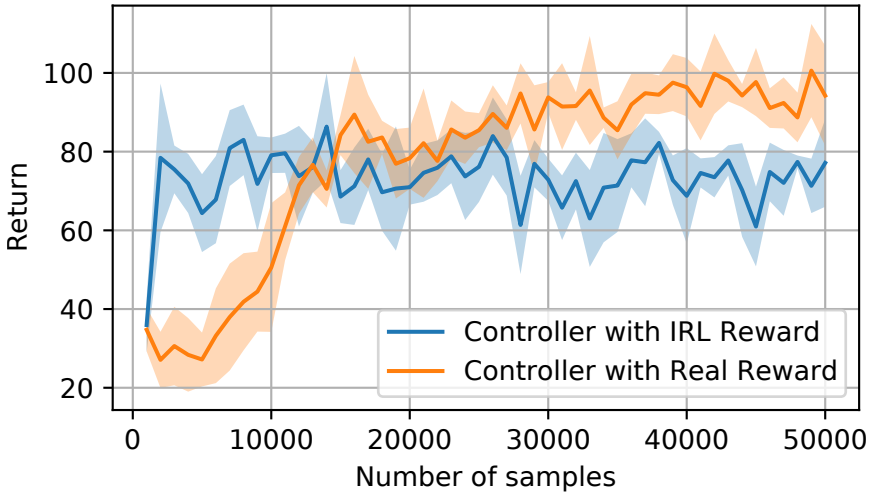


FIG. 3.6: Comparison of the effect of two different reward functions on the sample complexity of REINFORCE on the Mountain Car problem (5 runs, 95% c.i.).

3.5 Partial Conclusions

In this chapter, we began an investigative exploration into IRL, commencing with a foundational introduction that provided a contextual understanding of this intricate domain.

Subsequently, a presentation of the primary IRL algorithms was given, highlighting their unique characteristics and inherent limitations. This overview served to offer a panoramic glimpse into the diverse landscape of IRL methodologies.

Central to this chapter was the proposal of a novel approach to address a new facet within the IRL problem. This innovative methodology introduces a nuanced strategy that takes into account, during the reward selection phase, the sample budget of the subsequent forward RL phase. The essence of this approach lies in the simultaneous selection of reward parameters and the discount factor, accomplished through the formulation of a min-max problem. The primary objective is to minimize the distance between the expert's policy and the policy learned in the subsequent forward learning task. This approach aims to delicately balance potentially sub-optimal rewards with the errors stemming from the utilization of a limited sample size during the forward learning phase.

The empirical validation through numerical simulations highlighted the crucial necessity for a tailored reward function that accommodates the constraints posed by having a limited amount of samples.

Chapter 4

Transfer Learning

This chapter extensively explores Transfer Learning (TL) within the context of reinforcement learning. We initiate by offering an introductory overview and establishing the formalization of the TL problem. Following this, we examine categorization and the latest techniques within the TL domain. Ultimately, we introduce our innovative TL framework. This methodology leverages deep learning techniques, specifically employing the feature extraction capabilities of autoencoders and harnessing the generative power of GANs to facilitate the transfer of data samples between MDPs, even in scenarios where these MDPs do not share state and/or action spaces or possess any common prior knowledge.

4.1 Introduction to Transfer Learning

Transfer learning in the context of reinforcement learning has gained significant attention due to its potential to revolutionize the field by addressing one of its primary challenges: sample complexity [63, 109]. Reinforcement learning, although powerful, often demands an extensive amount of experience and data to train agents effectively. The ability to leverage knowledge from related but not necessarily identical tasks can substantially alleviate the burden of data collection. This is where transfer learning steps in as a promising solution. By allowing an agent to transfer its acquired knowledge and skills from one task to another, it not only accelerates learning but also makes RL more practical in domains where data scarcity or sample inefficiency has previously limited its applicability.

The advantages of transfer learning extend beyond just speeding up the learning process. By building upon the expertise developed in different yet related domains, transfer learning can improve the generalization capabilities of RL agents. It facilitates the efficient exploration of the state-action space, enabling faster convergence towards optimal policies, and ultimately enhancing an agent's performance [109]. Furthermore, transfer learning has the potential to foster agent adaptability to dynamic environments. Agents equipped with knowledge from diverse domains can better handle changes and uncertainties, which is critical for real-world applications where

conditions can be unpredictable. In essence, the integration of transfer learning into reinforcement learning holds the promise of reducing the data requirements, enhancing the adaptation to various scenarios, and improving the overall efficiency and effectiveness of RL algorithms, making them more practical and applicable to a wide range of complex tasks.

Transfer learning, although increasingly prevalent in RL, was originally a concept defined primarily within the realm of supervised learning [116, 124, 136]. In the supervised learning paradigm, the primary objective is to learn a mapping from inputs to outputs, and transfer learning often involved adapting a model trained on one task to perform better on a related task. In the context of RL, a task is represented by a MDP. Transfer learning in RL goes beyond adapting mere input-output mappings; it involves transferring knowledge about how to make decisions and navigate in an environment efficiently. This often requires learning the underlying dynamics, causal relationships, and strategies that are task-specific. By successfully transferring such knowledge, RL agents can start new tasks with a head start, leveraging previous experience to expedite learning, improve generalization, and adapt more effectively to diverse challenges and scenarios.

More formally, following the notation from [63], we define the *space of tasks* involved in the transfer learning problem as $\mathscr{M}$. Let Ω represent a probability distribution spanning the task space $\mathscr{M}$. We can denote the environment that sets the context for the transfer problem as $\mathscr{E} = (\mathscr{M}, \Omega)$. The tasks the learner faces are sampled from this task distribution $\mathcal{M} \sim \Omega$. This general definition draws parallels with traditional supervised learning, wherein training samples are obtained from a given distribution. Consequently, similar to the principles of classification and regression, transfer learning operates on the premise that when tasks originate from the same distribution, an algorithm that performs well on an average basis across a finite set of source tasks (or training tasks) will also exhibit strong generalization capabilities when applied to the target tasks within $\mathscr{M}$, which share the same underlying distribution Ω.

From an high level of abstraction, a learning algorithm can be seen as a function that takes as input some kind of knowledge and returns a solution in a set of possible hypotheses. More formally we can define:

$$\mathscr{A}_{\text{learn}} : \mathscr{K} \to \mathscr{S}, \tag{4.1}$$

where $\mathscr{K}$ is the set of possible knowledge the algorithm exploits (e.g. set of all possible samples, set of options, set of features, ...) and $\mathscr{S}$ represents the set of solutions (or hypotheses) the algorithm can return (e.g. for value-iteration algorithms is the space of all possible Q-functions $\mathscr{S} = \mathcal{Q}$).

Given the definitions of $\mathscr{A}_{\text{learn}}, \mathscr{K}$, and $\mathscr{S}$, we can now outline the abstract form of transfer learning algorithms. Let L be the number of source tasks we have at disposal, a transfer learning algorithm can be seen as the result of two subsequent phases. The first one, the *transfer phase* $\mathscr{A}_{\text{transfer}}$, maps knowledge from all the source tasks $\mathscr{K}_S^L$ and available knowledge from target (if any) $\mathscr{K}_T$ to a final knowledge $\mathscr{K}_{\text{transfer}}$:

$$\mathscr{A}_{\text{transfer}} : \mathscr{K}_S^L \times \mathscr{K}_T \to \mathscr{K}_{\text{transfer}} \tag{4.2}$$

The second one, the *learning phase* exploits such a final knowledge along with additional knowledge from the target (e.g. new samples, ...) to produce a solution in the solution space $\mathscr{S}$:

$$\mathscr{A}_{\text{learn}} : \mathscr{K}_{\text{transfer}} \times \mathscr{K}_T \rightarrow \mathscr{S} \tag{4.3}$$

To provide an example, our proposed approach deals with transferring samples from one or more source tasks to a target task within an offline context. In this scenario, $\mathscr{A}_{\text{transfer}}$ takes a large dataset of samples from L source tasks $\mathcal{D}_{S_i}$ and a small dataset of samples from the target task $\mathcal{D}_T$ as inputs. The transfer algorithm then produces an extensive dataset of synthetic samples $\hat{\mathcal{D}}_T$ coherent with the state and action space of the target task. During the learning phase, we utilize $\mathcal{D}_T$ in combination with $\hat{\mathcal{D}}_T$ to estimate the transition model of the target task. The DDPG algorithm [72], subsequently, exploit this transition model as a generative model to learn a policy. To sum up, in this particular scenario, we have $\mathscr{K}_S = \prod_{i=0}^{L-1} \{\mathcal{S}_{S_i}, \mathcal{A}_{S_i}, \mathcal{R}_{S_i}, \mathcal{S}_{S_i}\}^{|\mathcal{D}_{S_i}|}$, $\mathscr{K}_T = \{\mathcal{S}_T, \mathcal{A}_T, \mathcal{R}_T, \mathcal{S}_T\}^{|\mathcal{D}_T|}$ and $\mathscr{K}_{\text{transfer}} = \{\mathcal{S}_T, \mathcal{A}_T, \mathcal{R}_T, \mathcal{S}_T\}^{|\hat{\mathcal{D}}_T|}$. Here $\mathcal{S}_{S_i}, \mathcal{A}_{S_i}, \mathcal{R}_{S_i}$ represents the state space, the action space and the reward space for the i-th source task, and $\mathcal{S}_T, \mathcal{A}_T, \mathcal{R}_T$ the target task's counterpart.

4.2 Transfer Learning Algorithms

An RL problem is characterized by various components: state space, action space, dynamics, reward and so on. Tasks in $\mathscr{M}$ may differ in several ways, depending on these elements. For example, in [64] source and target tasks are characterized by the same state, action and reward space but different dynamics, while in [5] source and target tasks completely differ from each other. Depending on the differences between tasks' *domain* (i.e. state and action spaces) we can distinguish two major groups of transfer settings: *intra-domain* or *cross-domain*. Additionally, another way to classify the transfer algorithms is through the type of knowledge that is transferred.

In terms of the knowledge transferred, three prominent categories of algorithms emerge. Firstly, there's *instance transfer*, which aim to reuse data sampled in source tasks for the RL in the target task, whether for estimating the MDP model in model-based methods or directly approximating the value function or policy in RL model-free approaches. Second, *representation transfer*. This category of transfer algorithms exploits the source learned representation (e.g. value functions, feaures, ...) to support the learning in the target task offering various approaches like reward shaping or basis function extraction. Finally, similar to representation transfer, there is the *options and policies transfer*. In this case, whole policies or sequences of primitive actions are transferred to guide or jumpstart the learning in the target task.

In the following subsection, we provide an overview of the main transfer learning techniques for both intra-domain and cross-domain scenarios, ultimately categorizing these algorithms according to the specific type of knowledge they intend to transfer.

4.2.1 Intra-Domain TL

Formally, in the intra-domain setting we have one or more source tasks $\mathcal{M}_S = (\mathcal{S}, \mathcal{A}, P_S, r_S)$ and a target task $\mathcal{M}_T = (\mathcal{S}, \mathcal{A}, P_T, r_T)$ sharing the same domain $\mathcal{S} \times \mathcal{A}$

The early literature on transfer in reinforcement learning predominantly concentrated on scenarios where the domain is shared and only two tasks are involved in the transfer: a source task and a target task. In this context, the transfer algorithm may or may not have informations about the target task during the transfer phase. When the target task is not accessible, certain transfer algorithms employ a shallow transfer approach, utilizing the knowledge gathered in the source task (e.g., the policy) directly in the target task [31]. Conversely, when some information about the target task is available at the time of transfer, it is leveraged to adapt the knowledge from the source task to the specific requirements of the target task [64]. Moreover, during the past decade, intra-domain algorithms have begun to explore the concept of transferring knowledge from multiple source tasks [30, 64, 119]. In this context, we expect that, increasing the number of source tasks, the transfer algorithm can enhance the overall performance on the target task when compared to a single-task transfer learning algorithm.

4.2.1.1 Options and Policy Transfer

One of the commonly adopted strategies to address the transfer challenge between one source and one target task involves altering the representation of the MDP by introducing *options* [107], i.e. sequences of primitive actions, into the available action set. In discrete MDPs, the introduction of options doesn't fundamentally impact the potential to achieve an optimal solution, as all the primitive actions remain accessible, allowing for the representation of any policy. However, options can significantly enhance the speed of learning when they explore regions within the state space that are valuable for mastering the target task. The methodologies for transferring options predominantly pertain to discrete MDPs, employ a tabular representation of action-value functions, and involve source and target tasks that primarily differ only in their reward functions. The concept behind these approaches revolves around capitalizing on the shared dynamics structure between the two tasks, Consequently, most of these methods follow a similar underlying structure. Initially, a dataset $\mathcal{D}_S = \{s_i, a_i, r_i, s'_i\}_{i=0}^{N_S}$, is gathered from the source task, and an approximation of the MDP $\hat{\mathcal{M}}_S$ is estimated. Utilizing the properties of these estimated dynamics, the idea is to identify a selection of subgoals, and subsequently, a collection of d options is defined to facilitate the reaching of these subgoals. During the learning phase, the algorithm is encouraged to employ this newly extended action space to effectively acquire a solution for the target task via option Q-learning [107]. This subfamily of option transfer algorithms share this same structure. The distinctive aspect of each option-transfer algorithm [78, 79, 103] is the way the subgoals are identified and how to define options from the estimated dynamics. [13] introduces the novel concept known as the "reuse option", which becomes an integral part of the available action set. This reuse option is fashioned as a composite of optimal policies acquired from L source tasks, and its primary purpose is to expedite the learning process in the target task. Another approach, as exemplified by [85], employs options to enhance learning within a framework reminiscent of partially observable MDPs. In this context, it is assumed that the entire set of

possible tasks $\mathscr{M}$ is known in advance, and the target task is randomly selected from this set. Subsequently, authors computes, for each task, the belief that task is the target one. The value of an option in the target task is then determined as an average of its values across various tasks, weighted by the current belief of the target task. Notably, although this last two approaches presume that tasks originate from a shared distribution, they do not offer an analysis concerning which options would yield the most significant improvements. In contrast, [50] tackle the challenge of identifying the set of options that minimizes the iterations required for value iteration to reach convergence in the target task. Unlike previous approaches, this method guarantees the identification of the optimal set of options for the transfer algorithm. Using the notation from [63], the option-based approaches can be described by: $\mathscr{K}_S = \prod_{i=0}^{L-1}(\mathcal{S}_S, \mathcal{A}_S, \mathcal{R}_{S_i}, \mathcal{S}_S)^{N_{S_i}}$ and for any $K \in \mathscr{K}_S$, any option transfer algorithm returns $\mathscr{A}_{\text{transfer}}(K) = (\mathcal{O}, \mathscr{H})$, where L is the number of source tasks and $\mathscr{H} = \{\mathcal{Q} : \mathcal{S} \times \{\mathcal{A} \times \mathcal{O}\} \to \mathbb{R}\}$.

All previous options-transfer methods operate under the implicit assumption that the source and target tasks share sufficient similarity, enabling options extracted from source tasks to effectively improve learning the solution for the target task. However, it is essential to acknowledge that various interpretations of task similarity can be established. The only explicit effort to quantitatively measure the expected performance of transfer between source and target tasks in relation to the dissimilarity of the two MDPs is undertaken by [31] and [87]. Specifically, they examine scenarios where a policy π_S is transferred from a source task to a target task. In such cases, the learning process in the target task is initiated using π_S, and its initial performance is assessed. When the MDPs exhibit a considerable degree of similarity, it is expected that this policy-transfer method will yield initial performance improvements. In this context, following the previous formalism, $\mathscr{K}_S = \mathscr{K}_{\text{transfer}} = \Pi$, where Π denotes the set of all policies within the source (and target) domain, and there's no learning phase. Furthermore, [87] defines a state distance between $\mathcal{M}_S$ and $\mathcal{M}_T$ similar to the metrics presented in [31]. This state distance, denoted as $d : \mathcal{S} \to \mathbb{R}$, is defined as

$$d(s) = \max_{a \in \mathcal{A}}(|r_S(s,a) - r_T(s,a)| + \gamma \mathscr{J}(d)(P_S(\cdot|s,a), P_T(\cdot|s,a))) \tag{4.4}$$

where $\mathscr{J}$ corresponds to a Wasserstein distance, providing a measure of the difference between the two transition distributions $P_S(\cdot|s,a)$ and $P_T(\cdot|s,a)$, given the state distance d. Previous equation has been proved to have a stable fixed point, and so a solution d^*. [87] also proved that when a policy π_S is transferred from the source task to the target task, its performance can be upper bounded with respect to the optimal policy π_T^* by:

$$||V_T^{\pi_S} - V_T^{\pi_T^*}|| \leq \frac{2}{1-\gamma} \max_{s \in \mathcal{S}} d^*(s) + \frac{1+\gamma}{1-\gamma} ||V_S^{\pi_S} - V_S^{\pi_S^*}|| \tag{4.5}$$

In [30], under assumption of having source(s) and target task differing only in the reward, source policies are employed as probabilistic biases for the exploration in the target task. Essentially, the learning agent in the target task, during the exploration will either select a random action or following the source policies.

A distinct approach involving the action space is introduced in [100]. This method employs random task perturbation to create a set of tasks derived from a single source task, resulting in

a new action set. This new action set is obtained by excluding from the original action space $\mathcal{A}$ any actions that are not optimal in any of the artificially generated source tasks. A reduced action space $\mathcal{A}' \subseteq \mathcal{A}$ leads to a significant improvement in the learning speed for the target task, although it may come at the cost of a less optimal learned policy. This is because some of the actions that are removed based on the artificially generated source tasks might be optimal in the target task. However, when the source and target tasks are similar, and the perturbed tasks are sufficiently different from the source one, the method is more likely to retain the majority of actions required to effectively find the optimal policy.

4.2.1.2 Representation Transfer

An approach similar to option-transfer is presented in [29, 32, 76]. However, there is a notable distinction: instead of options, the transfer algorithm extracts a set of d value-function features denoted as $\{\phi_i\}_{i=1}^{d}$ from a collection of samples acquired from the source MDP $\mathcal{M}_S$. In other words, this can be expressed as $\mathscr{K}_S = (\mathcal{S}_S, \mathcal{A}_S, \mathcal{R}_S, \mathcal{S}_S)^{N_S}$, and for a given realization $K \in \mathscr{K}_S$, $\mathscr{A}_{\text{transfer}}(K) = \mathscr{F}^d$, where $\phi_i \in \mathscr{F}$. Similar to the option-transfer methodologies, it is assumed that the tasks share the same dynamics, and an estimated transition model $\hat{P}_S$ is utilized to extract these features. However, whereas option-transfer seeks to enhance learning speed, feature-transfer is geared toward achieving a more accurate approximation of the target value function. Importantly, option-transfer approaches are primarily designed for online algorithms like option Q-learning, whereas feature-transfer algorithms can be integrated with any RL algorithm employing a linear function approximation framework. In [76], features are derived via spectral analysis of the Laplacian of the estimated graph from the source MDP. These features capture the inherent structure of the underlying dynamics manifold, making them well-suited to approximate the value function of tasks that share the same dynamics. [32] extends this approach to a more generalized setting where both dynamics and reward functions may differ between source and target tasks. They propose an alternative method to construct the source graph and extract features that are particularly suitable for approximating functions arising from similar dynamics and reward functions. In the scenario involving multiple source tasks, [119] employ a straightforward state-aggregation function approximation approach. Here, the goal is to discover an aggregation of states that can effectively approximate the optimal value functions for all the tasks at hand.

4.2.1.3 Instance Transfer

Instance transfer in intra-domain involves reusing data samples or experiences collected from source tasks to assist in learning a target task within the same domain. By transferring instances such as state-action pairs, rewards, and transitions, the algorithm can benefit from the information accumulated in the source tasks. This method is particularly valuable when the source and target tasks share similarities in terms of state and action spaces, as it allows the model to draw upon relevant experiences to enhance performance in the new task. From this point of view, [64] presents an approach that selectively transfer samples based on similarity metrics between involved tasks. The algorithm start by collecting N_S samples in the form of

(s, a, r, s') for each source task $\mathcal{M}_{S_i}$ and N_T from the target, with $N_T << N_S$. In this scenario, we have $\mathscr{K}_S = \prod_{i=0}^{L-1}\{\mathcal{S}_S, \mathcal{A}_S, \mathcal{R}_S, \mathcal{S}_S\}^{N_S}$, $\mathscr{K}_T = \{\mathcal{S}_T, \mathcal{A}_T, \mathcal{R}_T, \mathcal{S}_T\}^{N_T}$ and $\mathscr{K}_{\text{transfer}} = \mathscr{K}_S$. For each source task $\mathcal{M}_{S_i}$, an approximation $\hat{\mathcal{M}}_{S_i}$ is computed and employed, along with the N_T target samples, to compute a similarity metric Λ_{S_i} between $\mathcal{M}_{S_i}$ and $\mathcal{M}_T$. This similarity metric is, essentially, the average probability that target samples are generated by the i-th source task:

$$\Lambda_{S_i} = \frac{1}{N_T}\sum_{t=0}^{N_T-1} Pr((s_t, a_t, r_t, s'_t)|\hat{\mathcal{M}}_{S_i}) \tag{4.6}$$

Finally, source samples are transferred proportionally to Λ_{S_i} to the target task. To enhance the process, an additional relevance measure is applied to each source sample. This ensures that only those samples from the source tasks, which are more likely to contribute to the learning performance in the target task, are taken into consideration. For more details on the relevance measure we refer to section 3.2 of [64]. Transferred samples are then employed, along with target task's ones in an offline RL context. In a similar batch setting, [61] operate under the assumption that tasks share identical state-transition dynamics but have different reward. Their approach involves the direct incorporation of all samples equipped by an supervised learned reward to enhance the dataset for an offline reinforcement learning algorithm.

4.2.2 Cross-Domain TL

Cross-domain transfer is the most general case of transfer learning. It involves transferring knowledge between tasks having domains with substantial differences. These differences can manifest in diverse state spaces, action spaces, and even entirely different environments. As a result, the domain gap between the source and target tasks can be quite large, making direct knowledge transfer challenging.

Formally, we can describe the cross-domain transfer as the setting where exist one or multiple source tasks $\mathcal{M}_S = (\mathcal{S}_S, \mathcal{A}_S, P_S, r_S)$, and a distinct target task, $\mathcal{M}_T = (\mathcal{S}_T, \mathcal{A}_T, P_T, r_T)$ where, usually, $\mathcal{S}_S \neq \mathcal{S}_T$, $\mathcal{A}_S \neq \mathcal{A}_T$, $P_S \neq P_T$, $r_S \neq r_T$.

In cross-domain transfer scenarios, domain adaptation techniques become crucial to bridge the gap and enable effective knowledge transfer. This domain adaptation is translated with establishing an inter-task mapping function linking source state and/or action variables to their counterparts in the target task and viceversa. This mapping serves as a key component for the effective transfer of knowledge [38, 111, 112, 129, 131]. During the earlier stages of cross-domain transfer learning in RL, it was common to assume the existence of an appropriate mapping function *a priori* [111, 112]. Some exceptions involved extensive iterations to identify all possible one-to-one state-action mappings, rendering the method impractical for environments with large domains. Alternatively, empirical methods relied on the similarities between tasks to define the inter-task mapping [108]. Over the past decade, research in transfer learning within RL has shifted towards automating the learning of the inter-task mapping. This mapping is subsequently utilized either to reuse a policy from a source task to the target task [131], jumpstart the reinforcement learning process in the target task with a transfer-based initialization of the policy [6], guide the reinforcement learning process in the target task via reward shaping [38],

or integrate policies from multiple source tasks with the learning policy [129]. These approaches that automatically search for an inter-task mapping can be broadly categorized into two methodological groups. The first group focuses on projecting the data from source and target tasks into a shared space, enhancing the discovery of relationships in a more informative domain [5, 38]. The second family of algorithms concentrates on direct data mapping between source and target tasks, optimizing for various criteria such as ensuring dynamics cycle-consistency or maximizing mutual information between source and mapped states [129, 131].

As there is no universally accepted notation for the inter-task mapping, in the subsequent sections of this document, we will represent any inter-task mapping function from the source task to the target task as M and its inverse from the target to the source as M^{-1}. If the mapping pertains only particular variables (such as state, action, or state-action couples), it will be specified in the subscript.

4.2.2.1 Policy Transfer

Among all the cross-domain transfer learning approaches, policy transfer is the most recent one. As far as our knowledge, one of the first work in this area is provided by [6]. In their research, the authors outline two primary contributions: the automatic identification of an inter-task mapping for states $\mathrm{M}_s : \mathcal{S}_S \rightarrow \mathcal{S}_T$ and $\mathrm{M}_s^{-1} : \mathcal{S}_T \rightarrow \mathcal{S}_S$, and the reuse of an optimal source policy (along with the inter-task mapping) to initialize the learning policy for the target task. To establish the inter-state mapping, authors utilize Unsupervised Manifold Alignment (UMA) [120], a versatile framework designed to reveal a shared latent representation capturing the intrinsic relationships between datasets, regardless of their dimensional disparities. Originally devised for aligning datasets in supervised learning tasks, UMA is adapted by the authors to an RL context, aligning the state spaces of both the source and target tasks. To learn such a mapping, as a first step authors collects a large source dataset of states $\mathcal{D}_S$ using the optimal source policy π_S^* and a smaller target dataset $\mathcal{D}_T$ using a randomly initialized policy π_T. From these datasets, authors construct two feature state datasets $\mathbf{\Phi}_S = \{\boldsymbol{\phi}_S(s_i) | s_i \in \mathcal{D}_S\}$ and $\mathbf{\Phi}_T = \{\boldsymbol{\phi}_T(s_i) | s_i \in \mathcal{D}_T\}$ where $\boldsymbol{\phi}_S$ and $\boldsymbol{\phi}_T$ represents a feature function for source and target states. Subsequently, UMA is employed to derive the mapping between tasks. For a more in-depth understanding of UMA, additional details can be found in [6, 120]. Once M_s and M_s^{-1} are derived, a small set $S_T = \{s_0^T, ..., s_{m-1}^T\}$ of target task's states are selected and mapped in the source domain $S_S = \mathrm{M}^{-1}(S_T) = \{s_0^S = \mathrm{M}^{-1}(s_0), ..., s_{m-1}^S = \mathrm{M}^{-1}(s_{m-1})\}$. Source policy π_S^* is employed to generate a dataset $\mathcal{D}'_S$ of m states trajectories of length H starting from states in S_S:

$$\mathcal{D}'_S = \{\tau_i = (s_{i,0}^S, ..., s_{i,H-1}^S) | s_{i,0}^S = s_0^S \in S_S, s_{i,k+1}^S \sim P_S(s_{i,k}^S, \pi_S^*(s_{i,k}^S))\}_{i=0}^{m-1}. \tag{4.7}$$

Once $\mathcal{D}'_S$ is collected, it is mapped back in the target tasks, thus obtaining:

$$\hat{\mathcal{D}}'_T = \{\hat{\tau}_i = (\hat{s}_{i,0}^T = M_s(s_{i,0}^S), ..., \hat{s}_{i,H-1}^T = M_s(s_{i,H-1}^S))\}_{i=0}^{m-1}. \tag{4.8}$$

In the target task, on the other hand, let's consider a non-deterministic parametric policy $\pi_{\boldsymbol{\eta}}(s) = \boldsymbol{\phi}_T^\top(s)\boldsymbol{\eta} + \epsilon$. $\pi_{\boldsymbol{\eta}}$ induces a states trajectories distribution $\rho_{\boldsymbol{\eta}}(\cdot)$

The algorithm, at this point, aim at finding a parameter $\boldsymbol{\eta}$ minimizing:

$$J(\boldsymbol{\eta}) = \sum_{i=0}^{m-1} \rho_{\boldsymbol{\eta}}(\tau_i)\mathcal{R}(\tau_i, \hat{\tau}_i), \tag{4.9}$$

where the states trajectory $\tau_i = (s_{i,0}^T = s_i^T, ..., s_{i,H-1}^T) \sim \rho_{\boldsymbol{\eta}}(\cdot)$ starts from the i-th state $s_i^T \in S_T$, $\hat{\tau}_i \in \hat{\mathcal{D}}_T^{'}$ is the mapped states trajectory starting from $\hat{s}_i^T = M_s^{-1}(M_s(s_i^T))$ and $\mathcal{R}(\cdot,\cdot)$ is a cost function penalizing deviations between states of τ_i and source-optimal mapped states trajectory $\hat{\tau}_i$:

$$\mathcal{R}(\tau_i, \hat{\tau}_i) = \frac{1}{H}\sum_{j=0}^{H-1} ||s_{i,k}^T - \hat{s}_{i,k}^T||_2^2 \tag{4.10}$$

Finally, once they identify $\hat{\boldsymbol{\eta}}^*$ minimizing equation (4.9), the authors execute a traditional reinforcement learning phase using the REINFORCE algorithm [126] to acquire the optimal policy for the target task. They initialize this learning process with the policy $\pi_{\hat{\boldsymbol{\eta}}}^*$, resulting in significantly higher reward compared to scenarios where no transfer learning techniques are employed.

The research conducted by [131] indirectly tackles the policy transfer problem. Their central objective revolves around establishing a "correspondence" between two MDPs to control the second MDP while interacting in the first, sharing a similar dynamics (for instance, controlling a virtualized robotic arm through a physical robotic one). In other words, they aim to find an inter-task mapping between a source and a target task, ensuring that by applying an action in the source and mapping it to the target, the agent in the target domain moves in a manner akin to the agent in the source domain. To do so, they define three mapping functions:

$$\begin{aligned} \mathrm{M}_s^{-1} &: \mathcal{S}_T \to \mathcal{S}_S \\ \mathrm{M}_{sa\to a} &: \mathcal{S}_S \times \mathcal{A}_S \to \mathcal{A}_T \\ \mathrm{M}_{sa\to a}^{-1} &: \mathcal{S}_T \times \mathcal{A}_T \to \mathcal{A}_S \end{aligned} \tag{4.11}$$

Given a dataset of triplets (s, a, s') from the source task $\mathcal{D}_S$ and from the target $\mathcal{D}_T$, each of this mapping function is implemented and trained as a generator function in a Generative Adversarial Network (GAN) [36] configuration, minimizing:

$$\min_{\mathrm{M}_s^{-1}} \max_{\mathrm{D}_s^{-1}} J_{\text{adv}}(\mathrm{M}_s^{-1}, \mathrm{D}_s^{-1}) = \mathbb{E}_{s_S \in \mathcal{D}_S}\Big[\log \mathrm{D}_s^{-1}(s_S)\Big] + \mathbb{E}_{s_T \in \mathcal{D}_T}\Big[\log\big(1 - \mathrm{D}_s^{-1}(\mathrm{M}_s^{-1}(s_T))\big)\Big]. \tag{4.12}$$

In equation (4.12) we reported the adversarial loss related to the state mapping M_s^{-1}, where D_s^{-1} is its discriminator's network, but the same loss is also adapted to fit the others mapping functions in equation (4.11). By employing only generator networks, one can only generate data fitting the source state-action space and the target action space, no mapping is learned at this point. To achieve that, two additional losses are added. The first one aims to ensure that $\mathrm{M}_{sa\to a}$ and $\mathrm{M}_{sa\to a}^{-1}$ are each other's inverses. To do so, authors propose the minimization of the cross-domain cycle consistency:

$$\min_{\mathrm{M}_{sa\to a}, \mathrm{M}_{sa\to a}^{-1}} J_{\text{dom}}(\mathrm{M}_{sa\to a}, \mathrm{M}_{sa\to a}^{-1}) = \mathbb{E}_{(s_T, a_T)\in\mathcal{D}_T}\Big[\Big|\Big|\mathrm{M}_{sa\to a}(M_s^{-1}(s_T), \mathrm{M}_{sa\to a}^{-1}(s_T, a_T)) - a_T\Big|\Big|_1\Big] \tag{4.13}$$

which implies that a mapped action should be able to be translated back:

$$\mathrm{M}_{sa\to a}(M_s^{-1}(s_T), \mathrm{M}_{sa\to a}^{-1}(s_T, a_T)) \approx a_T \tag{4.14}$$

The second additional loss, on the other hand, aims to ensure dynamics cycle-consistency, guaranteeing that, given a target triplet (s_T, a_T, s'_T), once remapped, it remains consistent with the dynamics of the source task:

$$\min_{\mathrm{M}_s^{-1}, \mathrm{M}_{sa\to a}^{-1}} J_{\mathrm{dyn}}(\mathrm{M}_s^{-1}, \mathrm{M}_{sa\to a}^{-1}) = \mathbb{E}_{(s_T, a_T, s'_T)\in\mathcal{D}_T}\left[\left|\left|\mathrm{M}_s^{-1}(s'_T) - \mathrm{T}_S(\mathrm{M}_s^{-1}(s_T), \mathrm{M}_{sa\to a}^{-1}(s_T, a_T))\right|\right|_1\right], \tag{4.15}$$

where T_S is the estimation of the source transition model pre-trained using $\mathcal{D}_S$. When considering all elements collectively, the whole loss function is defined as:

$$\begin{aligned} J_{\mathrm{full}} = {} & \lambda_0 J_{\mathrm{dyn}}(\mathrm{M}_s^{-1}, \mathrm{M}_{sa\to a}^{-1}) + \\ & + \lambda_1 (J_{\mathrm{adv}}(\mathrm{M}_{sa\to a}^{-1}, \mathrm{D}_{sa\to a}^{-1}) + J_{\mathrm{adv}}(\mathrm{M}_{sa\to a}, \mathrm{D}_{sa\to a}) + J_{\mathrm{dom}}(\mathrm{M}_{sa\to a}^{-1}, \mathrm{M}_{sa\to a})) + \\ & + \lambda_2 J_{\mathrm{adv}}(\mathrm{M}_s^{-1}, \mathrm{D}_s^{-1}), \end{aligned} \tag{4.16}$$

where λ_0, λ_1, and λ_2 are weights that their optimization algorithm turns off (setting them to 0) or on (resetting them to their original value) to alternate the minimization of the various components of the loss in equation (4.16). Finally, to assess their approach, authors first train a policy within the source task using DDPG [72]. Subsequently, they evaluate the cumulative reward achieved when this policy is deployed in the target task, making comparisons with other policies, which could be random actions, optimal actions, or policies from different transfer methodologies.

A recent and innovative approach is proposed by [129]. This method aims to adapt policies and value functions from N source tasks to a single target task during its learning process. The approach necessitates that both the source and target policies are implemented using neural networks with an identical number and structure of hidden layers. The same requirement applies to the value functions. The inter-task mapping between the i-th source and the target task is defined by:

$$\mathrm{M}_{i,s} : \mathcal{S}_{S_i} \to \mathcal{S}_T \tag{4.17}$$

$$\mathrm{M}_{i,s}^{-1} : \mathcal{S}_T \to \mathcal{S}_{S_i} \tag{4.18}$$

$$\mathrm{M}_{i,a} : \mathcal{A}_{S_i} \to \mathcal{A}_T \tag{4.19}$$

The acquisition of this mapping takes place during online training and requires ongoing interaction with the target task, as well as access to pre-existing source datasets $\mathcal{D}_{S_i}$. Each learning iteration within the target task follows a structured sequence: (i) solving optimization problems to define the mapping, (ii) mixing policies (and Q-functions) from the source tasks with the target task's policy to determine the data collection policy, and (iii) minimizing the gradient in a policy gradient approach to update the learning policy.

In the first stage, trajectories from the source datasets $\tau_{S_i} \in \mathcal{D}_{S_i}$ are collected, and target task's trajectories from previous learning iterations τ_T are employed to minimize cycle consistency losses. This process aims to ensure that the mappings $\mathrm{M}_{i,s}$ and $\mathrm{M}_{i,s}^{-1}$ indeed act as mutual inverses:

$$L_{\text{cyc}}(\mathrm{M}_{i,s}, \mathrm{M}_{i,s}^{-1}) = \mathbb{E}_{s_T \in \tau_T}\Big[\Big|\Big|\mathrm{M}_{i,s}(\mathrm{M}_{i,s}^{-1}(s_T)) - s_T\Big|\Big|\Big] + \\ + \mathbb{E}_{s_{S_i} \in \tau_{S_i}}\Big[\Big|\Big|\mathrm{M}_{i,s}^{-1}(\mathrm{M}_{i,s}(s_{S_i})) - s_{S_i}\Big|\Big|\Big]. \tag{4.20}$$

Furthermore, to maintain the coherence of the mapping with the target task's dynamics, each triplet $(s_S, a_S, s'_S) \in \tau_{S_i}$ is paired with a corresponding triplet $(\mathrm{M}_{i,s}(s_S), \mathrm{M}_{i,a}(a_S), s'_T)$, which is sampled from the target task when starting from the mapped state $\mathrm{M}_{i,s}(s_S)$ and applying the mapped action $\mathrm{M}_{i,a}(a_S)$. These paired triplets are used to calculate the correction loss:

$$L_{\text{corr}}(\mathrm{M}_{i,s}, \mathrm{M}_{i,s}^{-1}, \mathrm{M}_{i,a}) = \mathbb{E}_{\substack{(s_S, a_S, s'_S) \in \tau_{S_i} \\ s'_T \sim P_T(\cdot|\mathrm{M}_{i,s}(s_S), \mathrm{M}_{i,a}(a_S))}}\Big[\Big|\Big|\mathrm{M}_{i,s}^{-1}(s'_S) - s'_T\Big|\Big|_2^2\Big]. \tag{4.21}$$

Upon obtaining the mapping by minimizing the two previous losses, the next step involves mixing policies and value functions from the source tasks with their counterparts in the target task. The authors assume that every policy (source policies and learning policy in the target task) are structured as neural networks, sharing the identical hidden layer architecture (this holds true for the value functions as well). Considering the j-th hidden layer, the combination of source policies π_{S_i} with the learning target policy $\pi_{\boldsymbol{\eta}}$ is performed as:

$$\hat{z}_\eta^j = p z_\eta^j + (1-p)\sum_{i=0}^{N-1} w_i z_{S_i}^j. \tag{4.22}$$

Here $\hat{z}_\eta^j$, z_η^j, $z_{S_i}^j$ respectively represent the pre-activation representation of the j-th layer of the obtained combined policy $\hat{\pi}_{\boldsymbol{\eta}}$, the learning policy $\pi_{\boldsymbol{\eta}}$, and the i-th source policy π_i. Additionally, w_i are weights assigned to the i-th source task, and $p \in [0,1]$ is a value that progressively increases over time to enable gradual adaptation of the mixed policy to the learning one $\hat{\pi}_{\boldsymbol{\eta}} \to \pi_{\boldsymbol{\eta}}$. The same combination process is also applied to the value function. Once this combined policy is obtained, it is used to sample new data in the target task to compute the gradient for the Proximal Policy Optimization (PPO) reinforcement learning algorithm [99]. The inter-task mapping, which has not been utilized thus far, is employed in an additional loss for the gradient optimization to ensure that the visited states and mappings are highly correlated, thus enabling the agent to receive the most appropriate guidance from source policies. This is achieved by maximizing the mutual information:

$$L_{\text{MI}}(\mathrm{M}_s^{-1}) = -\mathbb{E}_{s_T \sim \rho'_{\boldsymbol{\eta}}}\Big[\log q_\omega(s_T|\mathrm{M}_s^{-1}(s_T))\Big]. \tag{4.23}$$

Here $\rho'_{\boldsymbol{\eta}}$ is the state distribution induced by the combined policy $\hat{\pi}_{\boldsymbol{\eta}}$ and $q_\omega(\cdot|\cdot)$ is the variational distribution (for a better insight we refer to [129]). Gradient of equation 4.23 is finally added to the PPO's one and used to compute the update of the learning policy network $\pi_{\boldsymbol{\eta}}$.

4.2.2.2 Representation Transfer

One of the first cross-domain representation transfer approaches is the one proposed in [110]. In their work, authors aim to reuse a source pre-learned action-value function Q_S in the target task proposing two approaches. In the first one, Q_S is properly adapted via a mapping $\mathrm{M}_Q : \mathcal{Q}_S \to \mathcal{Q}_T$ to the target task in order to behave as a starting point for the target action-value function Q_T in a TD approach. Authors propose three different algorithms for computing M_Q, using ad-hoc transfer of Q-functions parameters based on how those functions are implemented, namely *cerebellar model arithmetic computer* [4], neural networks or radial basis functions. To compute such mappings, these three algorithms make also use of mapping functions for state and action variables $\mathrm{M}_s^{-1} : \mathcal{S}_T \to \mathcal{S}_S$ and $\mathrm{M}_a^{-1} : \mathcal{A}_T \to \mathcal{A}_S$ that are assumed to be given. Authors' second approach, instead, aim to reuse Q_S in a reward shaping [62] manner evaluating, in the TD approach, a target state-action pair as a combination of the two approximators Q_S and Q_T, where the TD Q-function's update only involves Q_T:

$$Q(s,a) = Q_S(\mathrm{M}_s^{-1}(s), \mathrm{M}_a^{-1}(a)) + Q_T(s,a) \tag{4.24}$$

As mentioned earlier, this method relies on the availability of mapping functions, denoted as M_s^{-1} and M_a^{-1}, which are assumed to be provided in advance. In [113], the authors propose a procedure for automatically determining these functions. The algorithm begins by collecting a small dataset of samples from the target MDP, denoted as $\mathcal{D}_T$, and a larger one from the source MDP, denoted as $\mathcal{D}_S$. The algorithm proceeds by approximating the transition model $\hat{\mathrm{T}}_\mathrm{T} : \mathcal{S}_T \times \mathcal{A}_T \to \mathcal{S}$ of the target MDP using $\mathcal{D}_T$. Once this is done, for every possible 1-to-1 state mapping M_s and action mapping M_a, the algorithm creates a dataset $\hat{\mathcal{D}}_T = (\mathrm{M}_s(s), \mathrm{M}_a(a), \mathrm{M}_s(s'))|(s,a,s') \in \mathcal{D}_S$, which is then used to evaluate the Mean-Squared Error (MSE):

$$MSE = \frac{1}{|\hat{\mathcal{D}}_T|} \sum_{(s,a,s') \in \hat{\mathcal{D}}_T} \left|\left| \hat{\mathrm{T}}_\mathrm{T}(s,a) - s' \right|\right|_2^2. \tag{4.25}$$

Finally, the algorithm identifies the best mapping[1] as the one that create a dataset providing the minimum MSE of equation (4.25), and derives the inverse mappings M_s^{-1} and M_a^{-1} which are ultimately used to perform the Q-function transfer as in the second approach of [110].

[74] expanded these concepts by demonstrating the automatic construction of inter-task mappings using qualitative dynamic Bayes networks [35]. [77] have also explored the application of value transfer to enhance jumpstart. Their approach involves creating an optimistic initialization for the value function of the target task , using a manually provided inter-task mapping, by employing the source value functions. Additionally, [75] have investigated reusing single-agent knowledge in a deep multi-agent environment. They introduce a unique measure of MDP similarity based on n-step returns and employ it to develop a technique for transferring value functions.

[1] When we refer to 1-to-1 state mapping, we mean that, given a source state s_S, the mapped state in the target domain $s_T = \mathrm{M}_s(s_S)$ exhibits, as state variables, a combination of state variables values of s_S. The same principle applies to 1-to-1 action mapping M_a.

4.2.2.3 Instance Transfer

In the context of cross-domain scenarios, there has been limited exploration of instance transfer. One of the initial methods for instance transfer was introduced by [112].

Their approach is a model-based instance transfer. The transfer occurs online during the learning of the target task and assumes that the following inter-task mappings are provided in advance:

$$\begin{aligned} \mathrm{M}_s &: \mathcal{S}_S \to \mathcal{S}_T \\ \mathrm{M}_s^{-1} &: \mathcal{S}_T \to \mathcal{S}_S \\ \mathrm{M}_a &: \mathcal{A}_S \to \mathcal{A}_T \\ \mathrm{M}_a^{-1} &: \mathcal{A}_T \to \mathcal{A}_S \end{aligned}$$

At each time-step, the reinforcement learning algorithm used (we refer to Fitted R-Max in [112]) collect a triplet (s_T, a_T, s'_T) and estimates the transition model. If the model estimate is not accurate enough, their transfer algorithm search for one or more triplets s_S, a_S, s'_S from a source dataset $\mathcal{D}_S$ that are "near" enough to the mapped target triplet $(\mathrm{M}_s^{-1}(s_T), \mathrm{M}_a^{-1}(a_T), \mathrm{M}_s^{-1}(s'_T))$. Once these triplets are identified, they are remapped via M_s and M_a and employed (if not already selected in previous learning iterations) in the estimation of the transition model of the target task.

A notable attempt to automatically derive an inter-task mapping in the cross-domain setting has been proposed in the study by [5]. In their research, the primary objective is to learn an inter-task mapping M to transfer source triplets $z_S = (s_S, a_S, s'_S)$ to the target domain, amplifying the sample pool for an offline RL algorithm. Utilizing two datasets, one from the source ($\mathcal{D}_S$) and another from the target ($\mathcal{D}_T$), they utilize a sparse coding technique to project data from both datasets into a shared, higher-dimensional space. After the projection of samples from $\mathcal{D}_S$ and $\mathcal{D}_T$, the algorithm entails a pairing process. For each target triplet $z_T = (s_T, a_T, s'_T)$, the algorithm identifies the closest source triplet z_S in the projected space using Euclidean distance metrics, effectively creating a dataset $P = \{(z_S, z_T) | z_T \in \mathcal{D}_T, z_S = \arg\min_{z \in \mathcal{D}_S} ||p_S(z) - p_T(z_T)||_2\}$, where p_S and p_T respectively represent the projection of source and target triplets into the shared space. The authors subsequently employ supervised learning techniques (sparse Gaussian processes) on dataset P to determine M. Ultimately, the source dataset is mapped into the target domain ($\hat{\mathcal{D}}_T = \mathrm{M}(\mathcal{D}_S)$) and utilized in conjunction with $\mathcal{D}_T$ to derive an optimal policy for the target task using offline RL techniques, particularly FQI [25] and LSPI [60]).

A somehow different approach has been proposed in [38]. Authors, instead of transferring samples, employ available samples from source and target tasks to guide the exploration in the target task using reward shaping [62]. Similarly to [5], the approach aims to learn a mapping between the states of each task toward a shared informative space. The method assumes that RL agents in both tasks possess knowledge in solving a common task (e.g., transferring cooking skills among robots already skilled in object manipulation). This common task is necessary for extrapolating a pairing function $\mathrm{M}_s^{-1} : \mathcal{S}_T \to \mathcal{S}_S$ between source and target states. Given trajectory datasets of the solved common task in source and target domains, respectively $\mathcal{D}_S$ and $\mathcal{D}_T$, the authors employ a time-based approach or dynamic time warping [82] to determine M_s^{-1}, and creates

a dataset of pairs $P = \{(s_{S_p}, s_{T_p}) | s_{S_p} = \mathrm{M}_s^{-1}(s_{T_p}) \in \mathcal{D}_S, s_{T_p} \in \mathcal{D}_T\}$. This dataset P is the employed to address the following optimization problem:

$$\min_{\theta_f, \theta_g} \sum_{(s_{S_p}, s_{T_p}) \in P} \left|\left| f(s_{S_p}; \theta_f) - g(s_{T_p}; \theta_g) \right|\right|_2, \tag{4.26}$$

resulting in two functions, f and g, implemented via neural networks, projecting source and target states into a shared informative normalized space. To prevent degeneracies in defining functions f and g (e.g. $f(s_{S_p}; \theta_f) = g(s_{T_p}; \theta_g) = 0$) the authors treat these functions as encoders in an autoencoder [57] configuration, incorporating additional decoding losses into the problem (4.26). Utilizing the pairing function M_s^{-1} and these projection functions f and g, authors reformulate the target task's reward in a reward shaping manner as:

$$r(s) = r_T(s) + \alpha \left|\left| f(M_s^{-1}(s); \theta_f) - g(s; \theta_g) \right|\right|_2. \tag{4.27}$$

Here r_T is the original target task reward. In essence, this additional reward acts as a form of reward shaping, offering extra guidance for learning in the target task. In environments with sparse rewards, task performance heavily relies on directed exploration, and this additional incentive to match trajectory distributions in the shared space significantly improve the target task's resolution speed.

4.2.3 Evaluation Metrics

While supervised learning typically assesses the performance of a classifier or regressor in terms of prediction error, reinforcement learning offers a plethora of measures to evaluate the quality of the solution provided by the learning algorithm. Consequently, transfer algorithms can be appraised based on various performance metrics. In [109], multiple metrics were introduced to quantify the enhancement brought by transfer methods over single-task approaches:

- *Jumpstart*: The initial performance of an agent in a target task might receive a boost through transfer from a source task. This metric evaluates the agent's initial performance in a target task and addresses the question: "Can transfer facilitate an increase in the initial performance compared to the performance of an initial (random) policy?" Despite the appeal of such an initial boost, this metric fails to capture the learning behavior within the target task, concentrating solely on the performance before such a learning process begins.

- *Ultimate Performance*: This metric compares the final performance of learners in the target task, with and without employing transfer. Nevertheless, determining the point of actual convergence for the learner can be challenging, especially in tasks with infinite state spaces, or convergence may take an impractically long time. In numerous scenarios, the crucial factor is the quantity of samples necessary for learning rather than the performance of a learner with an infinite sample size. Additionally, various learning algorithms can converge to identical ultimate performance levels but may demand substantially different numbers of samples to achieve this level of performance.

- *Overall Reward*: The cumulative reward obtained by an agent (i.e., the area under the learning curve) could be increased by utilizing transfer, compared to learning without it. Enhancing the initial performance and accelerating the learning proces contributes to agents acquiring greater online rewards. In reinforcement learning, convergence is not assured with function approximation. Even if learners do converge, they might reach diverse, sub-optimal performance levels. Given sufficient samples or adequate training time, a learning method that swiftly attains high performance might amass less total reward compared to a method that learns slowly but eventually reaches a marginally higher performance plateau. This metric is mostly suited for tasks with a fixed duration.

- *Transfer Efficiency*: The ratio between the total reward accumulated by the transfer learner and that accumulated by the non-transfer learner. Considering two learning curves in the target task representing the achieved reward, one incorporating transfer and the other without it, presuming that the transfer learner accumulates a higher reward, the area under the transfer learning curve will exceed the area beneath the non-transfer learning curve.

$$r = \frac{\text{area under curve with transfer - area under curve without transfer}}{\text{area under curve without transfer}} \tag{4.28}$$

  Equation (4.28) provides a measure that quantifies the enhancement resulting from transfer learning. It is most suitable when aiming for identical final performance or when the task has a fixed duration. Alternatively, the ratio will be directly influenced by the duration the agents spend in the target task if there is not a predetermined endpoint for the task.

- *Time to Threshold*: The learning time required by the agent to attain a predefined performance level might be reduced through knowledge transfer. This metric face the challenge of necessitating the definition of a (possibly arbitrary) performance level that agents must attain. While some suggestions have been made to guide the appropriate selection of such thresholds [110], it is evident that the relative advantage of transfer learning methods will inevitably hinge on the specific threshold chosen, a decision that will inherently vary according to the domain and the learning method employed.

4.2.4 Challenges in Cross-Domain Transfer Learning

Transfer learning in reinforcement learning, particularly in the context of cross-domain transfer, presents both opportunities and challenges. While it offers the promise of leveraging knowledge gained in one domain to expedite learning in another, it also poses significant criticalities. One primary challenge lies in the assumption of similarity between the source and target domains. In cross-domain transfer learning, ensuring similarity between these domains becomes a critical factor, as dissimilarity can impede the effective transfer of knowledge. Domains may differ in dynamics, state spaces, or even the nature of tasks, leading to the issue of domain shift. Addressing such shifts and discrepancies becomes crucial for successful knowledge transfer, as the effectiveness of transfer learning greatly relies on the alignment or adaptation of the domains.

The research of this adaptation, practically implemented as an inter-task mapping, has emerged as a significant challenge in transfer learning. Initially, the research in this domain assumed that an "oracle" would provide the inter-task mapping in advance [110, 112]. However, practical applications necessitated the development of algorithms capable of autonomously deriving this mapping function [5, 6, 113, 129, 131]. Despite these advances, a key hurdle remains: these algorithms typically require the presence of a source policy to effectively train the mapping [129, 131]. The dependency on the existence of a source policy poses a significant limitation, as it may not always be feasible to obtain or assume the availability of such a policy in all scenarios. Consequently, while considerable progress has been made in the endeavor to automate the discovery of the inter-task mapping, the requirement for a source policy introduces a critical constraint in the practical application of such algorithms in cross-domain transfer learning.

Additionally, the prevailing trend in cross-domain transfer learning often hinges on the assumption that the source tasks have already been successfully addressed, and a transferrable solution is at hand. These methods typically presume that the source tasks are in a solved state, and this solution is poised for transfer, be it in the form of policies or action value-functions [6, 110, 129, 131]. Other approaches, instead, operate under the assumption that the target learning agent already possesses some prior knowledge regarding task resolution [38]. While these can be powerful assumptions in settings where different tasks are indeed solved, it may not be realistic for scenarios where tasks are complex and challenging to solve definitively. Hence, this aspect of source task completion stands as a fundamental consideration in the design and evaluation of cross-domain transfer learning methods, as it significantly impacts the feasibility and success of knowledge transfer.

Finally, the majority of documented algorithms either rely on the process of online learning to attain an optimal solution [6, 38, 110, 113, 129], or they may not actively pursue the search for such an optimal solution [131]. While this method may seem like a natural approach to perform transfer in the RL context, it introduces specific implications and considerations within the domain of cross-domain transfer learning. For instance, the reliance on online learning assumes the availability of continuous interaction in the target domain, which may not always be feasible or practical in real-world applications. Additionally, the lack of active search for an optimal solution may lead to suboptimal outcomes in the target domain, impacting the efficiency of transfer methods. Hence, while online learning presents a common avenue for knowledge transfer, its limitations and dependencies on specific conditions necessitate critical evaluation within the context of cross-domain transfer learning.

[5] faced all the previous challenges, proposing a instance transfer approach aimed to improve the performances in the offline RL's setting. But the main drawback of this approach is that, after projecting source and target datasets in an high dimensional informative space, finding pairs of closest projected triplets in this new space is very computationally expensive, by scaling with an order of $O(n_S \cdot n_T)$, where n_S and n_T are the sizes of source and target dataset respectively. In modern and complex applications requiring vast volumes of data, this approach becomes challenging to implement.

Table 4.1: Summary of the reported research works along with their main limitations.

Work	Technique	Limitations
[38]	Instance Transfer	Online learning, previous knowledge about common task
[112]	Instance Transfer	Oracle-given inter-task mapping, online learning
[61, 64]	Instance Transfer	Intra-domain
[5]	Instance Transfer	Heavy computational cost
[110]	Representation Transfer	Oracle-given inter-task mapping, pre-learned source Q-function, online learning
[29, 32, 76]	Representation Transfer	Intra-domain
[113]	Representation Transfer	Heavy computational cost, pre-learned source Q-function, online learning
[78, 79, 103, 107]	Option Transfer	Intra-domain, online learning
[13]	Option Transfer	Intra-domain, pre-learned source policies
[31, 87]	Policy Transfer	Intra-domain, pre-learned source policy
[131]	Policy Transfer	Pre-learned source policy
[129]	Policy Transfer	Pre-learned source policy, policies sharing the same structure
[6]	Policy Transfer	Pre-learned source policy, online learning

4.2.5 Advantages of Instance Transfer

Although relatively unexplored in the cross-domain context, instance transfer exhibits features capable of addressing the aforementioned challenges.

A prominent advantage of instance transfer algorithms is their detachment from the requirement for optimal solutions in the source tasks. Unlike many transfer algorithms that assume either resolved source tasks or access to nearly optimal solutions, instance transfer algorithms operate independently of whether actions performed are optimal. The resultant transitions would still yield valuable insights about the source environment. When these environments are someway related (for instance, through an inter-task mapping) to the target task, transferring these instances markedly enriches the dataset, thereby enhancing the knowledge available for exploitation by an RL algorithm in the target task.

Furthermore, another fundamental property of instance transfer pertains to the very nature of the samples. While knowledge obtained through the transfer of policies or value functions is typically tied to the specific RL algorithm intended for use, individual samples provide a more "raw" yet adaptable form of information. Regardless of the chosen approach, the foundation of any RL algorithm lies in the transition (s, a, r, s'). This transition represents the core element pivotal for understanding the agent's interaction with its environment. Unlike pre-defined policy or value function transfer, these individual samples encapsulate real interactions, embodying the essence of the RL process, where each instance contributes to the understanding of the agent's

learning process. These samples are not constrained to a specific algorithm; rather, they serve as fundamental units of learning that can be applied across a diverse range of RL models. This adaptability provides a broader spectrum of applicability, allowing these instances to be leveraged across different tasks, scenarios, and learning algorithms.

Finally, differently from policies and representations, a wealth of experience from related environments is often readily accessible. In the domain of autonomous driving, consider a scenario where a fleet of self-driving cars operates in various cities. Each vehicle might have slightly different configurations and be exposed to distinct environments, such as varied terrains, local traffic patterns, and climatic conditions. Over time, these cars accumulate driving experience, including historical data of their operations—details like sensor readings, driving behaviors, and responses to various situations. In an instance transfer application, the collective historical records from these different cars, despite their nuanced operational differences, can be harnessed as valuable experiential data. This shared knowledge could be used to improve and fine-tune the autonomous driving systems in a newer car model or to adapt the vehicle's navigation behavior in a city with different traffic norms. This identical scenario is pervasive in a spectrum of real-world control problems, where changes transpire in the control systems, environment, or objectives over time. These situations span domains such as robotics, autonomous vehicles, power facilities, water reservoir management, and more.

4.3 Generative Adversarial Autoencoding Networks for RL Cross-Domain Instance Transferring

In light of the challenges associated with state-of-the-art cross-domain transfer learning approaches and recognizing the advantages of instance transfer in comparison to other transfer methods, this section introduces a model-based instance transfer algorithm. This approach employs the feature extraction abilities of autoencoders [57] in combination with the generative capabilities of GANs [36] to establish an inter-task mapping connecting the source and target domains. Our algorithm also ensures alignment with the dynamics observed in the dataset of the target task through the estimation (and use) of its transition model. Once this mapping is obtained, the source transitions' dataset is transformed into a synthetic dataset that aligns with the dynamics observed in the target task's dataset. The underlying motivation is to generate a larger pool of triplets for the target task, which can be harnessed for estimating a transition model using model-based control algorithms or for application in model-free offline RL techniques. Additionally, this approach enables the mapping of multiple source tasks onto the same target task, thereby harnessing knowledge from various domains to create a more extensive and informative synthetic dataset.

For simplicity, let's denote $\mathcal{Z}_S \triangleq \mathcal{S}_S \times \mathcal{A}_S \times \mathcal{S}_S$, and $\mathcal{Z}_T \triangleq \mathcal{S}_T \times \mathcal{A}_T \times \mathcal{S}_T$. We operate under the reasonable assumption of having a large dataset $\mathcal{D}_S$ of triplets $(s_S, a_S, s'_S) \in \mathcal{Z}_S$ from the source domain and only a limited set of triplets $\mathcal{D}_T$ from the target domain, without any assumption on the collecting policy. Our objective is to obtain an inter-task mapping function $\mathrm{M} : \mathcal{Z}_S \to \mathcal{Z}_T$

that can efficiently convert source triplets $z_S \in \mathcal{Z}_S$ into their equivalent target triplets $\hat{z}_T \in \mathcal{Z}_T$ and viceversa ($\mathrm{M}^{-1} : \mathcal{Z}_T \to \mathcal{Z}_S$).

4.3.1 Transfer of Instances via Generative Adversarial Autoencoding Networks

We begin by introducing an autoencoder architecture [57]. Within this structure, the mapping from the source domain to the target domain, denoted as M, is implemented by the encoder module. Simultaneously, the decoder module handles the inverse mapping, M^{-1}, aimed at capturing the most crucial data features. The training of this autoencoder focuses on minimizing the L1 loss between the input, represented as the source triplets z_S, and the output, which is the reconstruction $\hat{z}_S = \mathrm{M}^{-1}(\mathrm{M}(z_S))$:

$$J_R(z_S, \hat{z}_S) = \|z_S - \hat{z}_S\|_1 \tag{4.29}$$

Subsequently, we introduce two constraints to force the encoder in aligning its mappings with the distribution of the target data. Initially, we ensure that the encoded triplet $\hat{z}_T = \mathrm{M}(z_S)$ maintains consistency with the patterns observed in the target triplets within $\mathcal{D}_T$. To enforce this condition, we undergo a pre-training step for a deterministic approximate version of the target transition model, $\hat{\mathrm{T}}_\mathrm{T} : \mathcal{S}_T \times \mathcal{A}_T \to \mathcal{S}_T$. This model receives a state-action pair as input and predicts the subsequent state, effectively approximating the dynamics of the target task. The training process leverages a supervised regression objective:

$$\min_{\hat{\mathrm{T}}_\mathrm{T}} J_{\text{model}}(\hat{\mathrm{T}}_\mathrm{T}) \triangleq \min_{\hat{\mathrm{T}}_\mathrm{T}} \mathbb{E}_{(s_T, a_T, s'_T) \in \mathcal{D}_T} \left[\left\| \hat{\mathrm{T}}_\mathrm{T}(s_T, a_T) - s'_T \right\|_2^2 \right] \tag{4.30}$$

We use this transition model to force the encoder in generating triplets whose $\hat{s}'_T$ component aligns with the predictions made by $\hat{\mathrm{T}}_\mathrm{T}$ when provided with $\hat{s}_T$ and $\hat{a}_T$. This is achieved by incorporating the following loss function, which is minimized during the training of the autoencoder:

$$J_T(\hat{z}_T; \hat{\mathrm{T}}_\mathrm{T}) \triangleq \left\| \hat{\mathrm{T}}_\mathrm{T}(\hat{s}_T, \hat{a}_T) - \hat{s}'_T \right\|_2^2 . \tag{4.31}$$

Additionally, we train a discriminator $\hat{\mathrm{D}}_\mathrm{T} : \mathcal{Z}_T \to (0, 1]$ with the purpose of distinguishing target domain triplets as either valid or fake. The training process of $\hat{\mathrm{D}}_\mathrm{T}$ must be adversarial [36], as its capacity to differentiate between the two categories should progressively enhance alongside the encoder's ability in mapping. This is achieved by solving:

$$\begin{aligned} \max_{\hat{\mathrm{D}}_\mathrm{T}} J_{\text{discr}}(\hat{\mathrm{D}}_\mathrm{T}) \triangleq \max_{\hat{\mathrm{D}}_\mathrm{T}} & \mathbb{E}_{z_T \sim \mathcal{D}_T} \left[\log\left(\hat{\mathrm{D}}_\mathrm{T}(z_T) \right) \right] + \\ & + \mathbb{E}_{z_S \sim \mathcal{D}_S} \left[\log\left(1 - \hat{\mathrm{D}}_\mathrm{T}(\mathrm{M}(z_S)) \right) \right]. \end{aligned} \tag{4.32}$$

The objective specified in equation (4.32) aims to enhance the discriminator ability to distinguish the mapped triplets $\mathrm{M}(z_S)$ from the target task triplets z_T. The first component of the loss penalizes the misclassification of actual target triplets as fake, whereas the second term penalizes the recognition of mapped triplets as genuine target ones. Similarly to what has been done with

the approximate transition model $\hat{\mathrm{T}}_{\mathrm{T}}$, we employ the discriminator $\hat{\mathrm{D}}_{\mathrm{T}}$ to formulate an additional loss function to be "minimized" during the training of the autoencoder:

$$J_D(\hat{z}_T; \hat{\mathrm{D}}_{\mathrm{T}}) \triangleq -\log \hat{\mathrm{D}}_{\mathrm{T}}(\hat{z}_T), \tag{4.33}$$

with the goal of making the autoencoder able to trick the discriminator in the triplets mapping process. Equation (4.33) returns a high value when $\hat{\mathrm{D}}_{\mathrm{T}}$ recognizes mapped triplets as fake, and a low value when the encoder succeeds to deceive the discriminator.

At this point, we define our full objective for the training of the autoencoder model involving M and M^{-1} based on previous loss functions (4.29), (4.31), and (4.33) as:

$$\begin{aligned} \min_{\mathrm{M},\mathrm{M}^{-1}} J_{\mathrm{AE}}(\mathrm{M}, \mathrm{M}^{-1}) = \mathbb{E}_{z_S \sim \mathcal{D}_S} \Big[& \lambda_R J_R\Big(z_S, \mathrm{M}^{-1}(\mathrm{M}(z_S))\Big) \\ & + \lambda_T J_T\Big(\mathrm{M}(z_S); \hat{\mathrm{T}}_{\mathrm{T}}\Big) \\ & + \lambda_D J_D\Big(\mathrm{M}(z_S); \hat{\mathrm{D}}_{\mathrm{T}}\Big)\Big], \end{aligned} \tag{4.34}$$

where each independent loss is weighted by a factor λ_R, λ_T and λ_D for balancing the effect of each loss function.

We provide an overview of our Generative Adversarial Autoencoding Network (GAAN) and its associated loss functions in figure 4.1. To enhance the discriminative capabilities of $\hat{\mathrm{D}}_{\mathrm{T}}$, we chose to follow the approach outlined in algorithm 3 rather than directly optimizing equation (4.34). This approach allows for the simultaneous improvement of $\hat{\mathrm{D}}_{\mathrm{T}}$'s ability to differentiate between real and generated samples while minimizing the adversarial loss.

The algorithm begins by taking the source dataset $\mathcal{D}_S$ and target dataset $\mathcal{D}_T$, as well as untrained models for $\hat{\mathrm{T}}_{\mathrm{T}}$, $\hat{\mathrm{D}}_{\mathrm{T}}$, M and M^{-1} as inputs. The initial step uses the target dataset $\mathcal{D}_T$ to train $\hat{\mathrm{T}}_{\mathrm{T}}$ and $\hat{\mathrm{D}}_{\mathrm{T}}$ by solving optimization problems (4.30) and (4.32), respectively. This step equips us with a transition model to support the subsequent autoencoder training process and prepares the discriminator to recognize valid data from the original target dataset $\mathcal{D}_T$. Afterwards, the iterative training process starts by training the encoder and decoder models (M and M^{-1}) solving optimization problem (4.34) and using pre-trained models $\hat{\mathrm{T}}_{\mathrm{T}}$ and $\hat{\mathrm{D}}_{\mathrm{T}}$. Encoder M is then employed to transfer instances from the source dataset to the target task. The transferred dataset $\hat{\mathcal{D}}_T$ is then used for training the target discriminator model $\hat{\mathrm{D}}_{\mathrm{T}}$, in conjunction with the original target dataset $\mathcal{D}_T$. By iteratively enhancing $\hat{\mathrm{D}}_{\mathrm{T}}$'s ability to distinguish genuine and generated target samples, we indirectly improve the encoder's proficiency in mapping source samples into different yet valid regions of the target domain.

The iterative training process continues until convergence, culminating in the encoder and decoder components of the GAAN.

4.3.2 Experiments

Now, we numerically evaluate our approach for learning the inter-task mapping and successively transforming triplets instances from a source to a target task. For all our validation experiments,

Algorithm 3 Instance Transferring through Generative Adversarial Autoencoding Network

Require:
Source Dataset: $\mathcal{D}_S = \{z_S^i \in \mathcal{Z}_S\}^{i=1...n_S}$
Target Dataset: $\mathcal{D}_T = \{z_T^i \in \mathcal{Z}_T\}^{i=1...n_T}$
Target transition model: $\hat{\mathrm{T}}_\mathrm{T}$
Target discriminator model: $\hat{\mathrm{D}}_\mathrm{T}$
Autoencoder model: AE
Ensure: Trained inter-task mapping M and M^{-1}
Train $\hat{\mathrm{T}}_\mathrm{T}$ solving optimization problem (4.30)
Train $\hat{\mathrm{D}}_\mathrm{T}$ solving optimization problem (4.32) using only $\mathcal{D}_T$
while not converged **do**
    Train AE solving optimization problem (4.34)
    $\mathrm{M} \leftarrow$ AE 's encoder
    $\hat{\mathcal{D}}_T \leftarrow \mathrm{M}(\mathcal{D}_S)$
    Train $\hat{\mathrm{D}}_\mathrm{T}$ solving problem (4.32) using $\mathcal{D}_T$ and $\hat{\mathcal{D}}_T$
end while
return encoder M

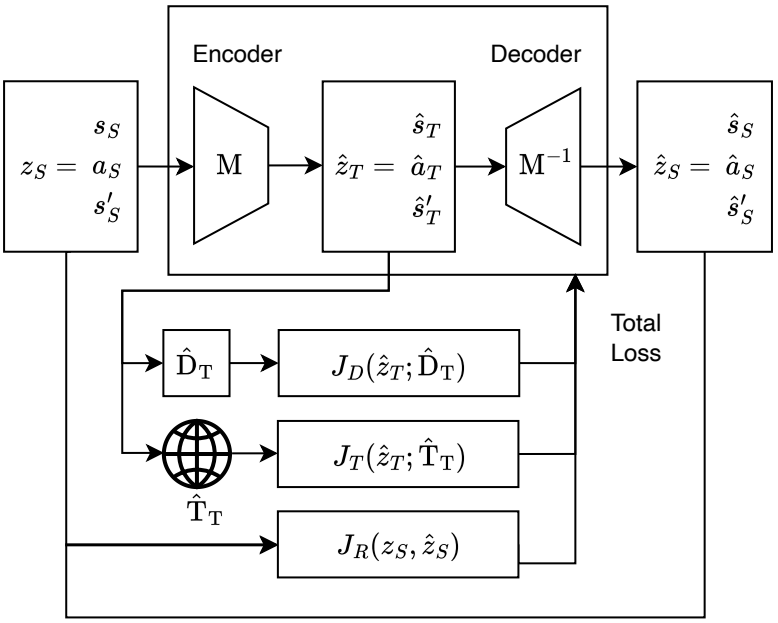


FIG. 4.1: Sketch of the proposed model to learn the mapping among two different tasks.

we utilized the environments provided by the OpenAI Gym framework [17]. We chose as testing tasks three well-known benchmark environments in RL, namely the Mountain Car (MC) with continuous action, the Cart Pole (CP) and the Inverted Pendulum (IP) (see appendix A for a better insight on adopted environments), but our method may be extended to a diverse range of scenarios without any particular assumption.

Experimental Setup

Starting with a dataset of source triplets $\mathcal{D}_S$, we conducted two sets of experiments. The initial experiment, outlined in section 4.3.2.1, involved assessing the effectiveness of the acquired inter-task mapping by directly analyzing the effect of the transformed samples $\hat{\mathcal{D}}_T = \mathrm{M}(\mathcal{D}_S)$

on the prediction error of a transition model for the target environment. The subsequent experiment, detailed in section 4.3.2.2, involved further evaluating the mapping's performance by examining the influence of the synthetically generated data $\hat{\mathcal{D}}_T$ on the estimation of a target transition model, which is utilized as a generative model for the Deep Deterministic Policy Gradient (DDPG) algorithm [72]. In this way we can evaluate our inter-task mapping technique and its practical application in the context of Reinforcement Learning.

In the upcoming sections, we will use the notation T_L^S to denote a transition model trained on L triplets from the target dataset $\mathcal{D}_T$ and S synthetic triplets from $\hat{\mathcal{D}}_T$; the corresponding index is omitted when either L or S equals 0.

All employed datasets consists of triplets from either source or target tasks, collected via random exploration on their respective environments. To ensure proper weighting of the independent components within the datasets $\mathcal{D}_S$ and $\mathcal{D}_T$, we normalized the triplets before commencing the mapping learning process. Finally, the neural network architectures employed in algorithm 3 for the models M, M^{-1}, $\hat{\mathrm{D}}_\mathrm{T}$, and $\hat{\mathrm{T}}_\mathrm{T}$ are depicted in figure 4.2, while learning hyper-parameters are shown in Table 4.2.

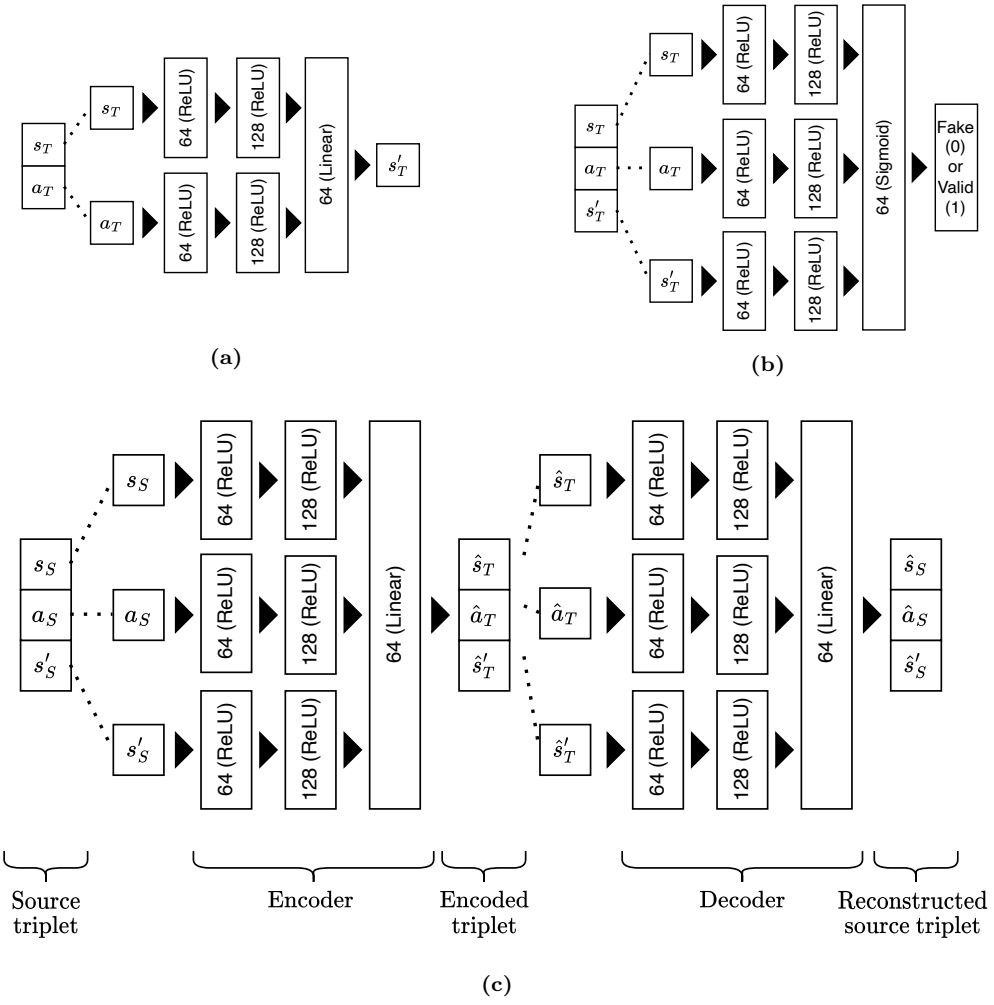


FIG. 4.2: Neural Network architectures for (a) transition model $\hat{\mathrm{T}}_\mathrm{T}$, (b) discriminator $\hat{\mathrm{D}}_\mathrm{T}$ and (c) autoencoder composed by encoder M and decoder M^{-1}.

Table 4.2: Hyper-parameters used during the learning process of all the models.

	Learning Rate	Batch Size	Epochs
Discriminator (pre-training)	0.00256	237	23
Discriminator (adversarial training)	0.00256	176	18
Transition model	0.001	100	100
Autoencoder	0.00256	237	68
Adversarial Iterations	λ_D	λ_T	λ_R
20	5.31979	1.14826	12.62707

4.3.2.1 Model evaluation through prediction errors

A core motivation behind the generation of synthetic triplets via the inter-task mapping is to feed such dataset $\hat{\mathcal{D}}_T$ into the learning process of a target transition model so as to improve its generalization capability.

We started by training two transition models, denoted as T_{200} and T_{5000}, utilizing two MC target datasets $\mathcal{D}_T$ containing 200 and 5000 triplets, respectively. The first model was employed within algorithm 3 to aid in the acquisition of the inter-task mapping, while the second model was specifically designed to serve as a benchmark for our analysis.

Following this, from two datasets $\mathcal{D}_S$ each containing 5000 triplets originating from the source tasks IP and CP, we learnt, via algorithm 3, two inter-task mappings M: one for IP $\rightarrow$ MC, and the other for CP $\rightarrow$ MC. These mappings were then applied to convert the source datasets into synthetic ones, $\hat{\mathcal{D}}_T$, for the MC task. These transformed triplets were used, for each source task, to train two types of target transition models: one solely with the transformed data (namely, T_pend^{5000} for the IP $\rightarrow$ MC mapping and T_cp^{5000} for the CP $\rightarrow$ MC mapping), and the other by combining target and transformed triplets together (namely, $T\_pend_{200}^{5000}$ and $T\_cp_{200}^{5000}$).

Finally, we trained an additional model, $T\_multi_{200}^{10000}$, combining synthetic samples transformed from the two source environments, IP and CP (each with 5000 triplets), and actual samples from the MC task (200 triplets).

To evaluate the models' predictive accuracy, a test dataset $\mathcal{D}_T^{\text{test}}$ was collected from the MC environment, containing 50000 triplets that were not employed at any stage of the algorithm or during the training of the target transition models. For a model T, the prediction error was calculated as $||T(s_T, a_T) - s'_T||_2^2$, where the normalized triplets (s_T, a_T, s'_T) are taken from the dataset $\mathcal{D}_T^{\text{test}}$. The error distribution for each trained model is depicted in Figure 4.3.

The findings reveal that models trained solely with synthetic data, namely T_{pend}^{5000} and T_{cp}^{5000}, exhibit a distribution of errors similar to that of T_{200}, the transition model trained exclusively with available target data. More importantly, the distribution of prediction errors for models utilizing both target and synthetic datasets, namely $T_{pend}^{5000}200$ and $T_{cp}^{5000}200$, is notably lower than that of T_{200}, which represents the model without any transfer technique. This result is

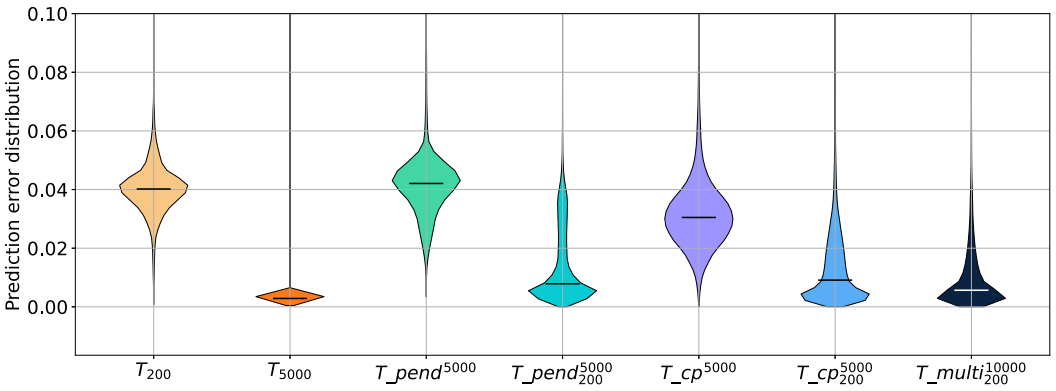
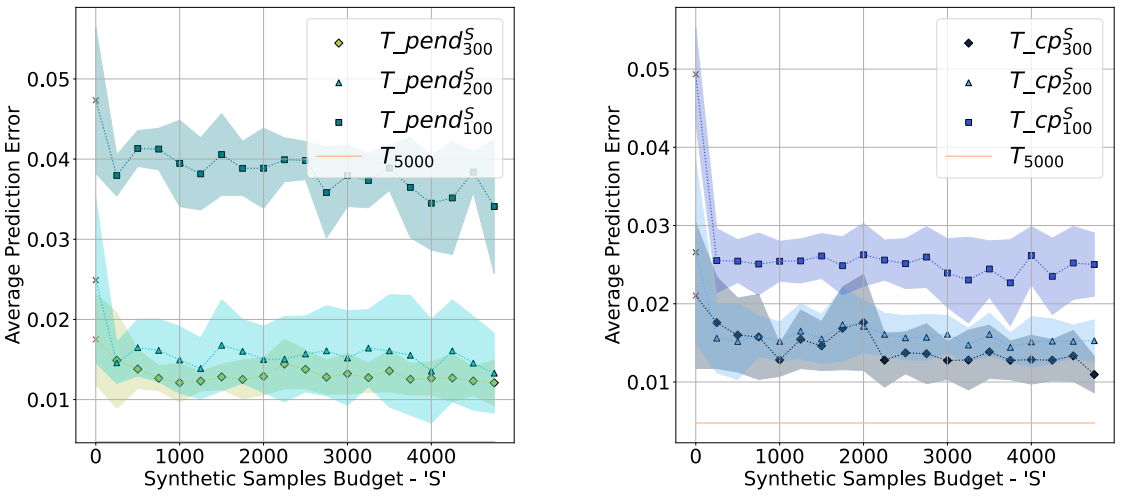


FIG. 4.3: Prediction Error distribution for relevant data configurations.

even more evident for the model that combines both source datasets transformed for the target task, $T\_multi_{200}^{10000}$.

We conducted further analysis to evaluate the effectiveness of the proposed inter-task mapping procedure, varying the sizes of the synthetic datasets $\hat{\mathcal{D}}_T$ from 0 (representing no transfer at all) to 5000. This process was repeated for three target datasets $\mathcal{D}_T$ of sizes 100, 200, and 300, respectively. Across all three cases, we noticed a similar impact of the transfer concerning the number of synthetic samples incorporated during model training.

Figures 4.4a and 4.4b show the prediction errors resulting from the inclusion of the synthetic datasets from the IP and CP source environments, respectively using the test dataset $\mathcal{D}_T^{\text{test}}$, together with its 95% confidence interval over 20 repetitions of the experiment.



(a) **(b)**

FIG. 4.4: (a) Effect on the prediction error of using a varying budget of synthetic samples mapped from IP, fixing the amount of target MC samples. (b) Effect on the prediction error of using a varying budget of synthetic samples mapped from CP, fixing the amount of target MC samples.

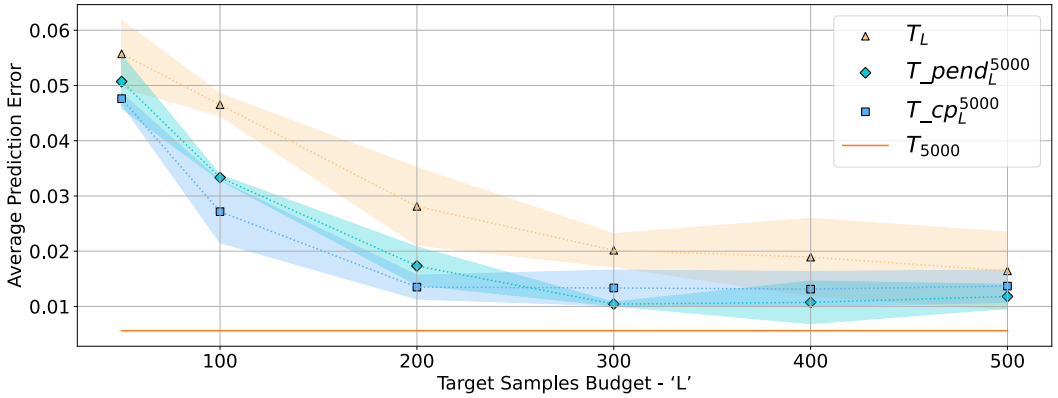


FIG. 4.5: Effect on the prediction error of using different amount of target MC samples, fixing the amount of synthetic data.

In both scenarios, it is evident that the advantage offered by the transformed triplets is primarily noticeable for the initial synthetic samples, and this advantage tends to remain relatively constant even with larger quantities of additional samples.

Finally, as an additional proof of concepts, we maintained the quantity of synthetic triplets constant and altered the number of actual target triplets at our disposal. This method allowed us to evaluate the effectiveness of our approach in scenarios with varying total amounts of available data. As anticipated, the average prediction error diminishes as the number of utilized target triplets increases. This outcome is shown in Figure 4.5.

We have consistently observed that a target transition model trained with a sufficiently large collection of target triplets achieves the lowest prediction error, see T_{5000} in figure 4.3. However, for tasks with low data availability, our method has proven to be an effective alternative, as shown in figure 4.3 for all the models trained with both synthetic and actual target data. It allows the model trained with both synthetic and target data to decrease its prediction error significantly when compared to models that use only the same amount of target triplets.

4.3.2.2 Model evaluation through Reinforcement Learning

Transfer Learning seeks to improve the efficacy of RL control while minimizing the necessity of gathering extra data from the target task. Within this investigation, we utilized some of the models obtained in section 4.3.2.1 as generative models for the target environment. These generative models produce data utilized by an RL algorithm to acquire a control strategy. The target task's reward is supposed to be known in advance. The efficiency of the acquired policies is then assessed in the actual target environment by quantifying the time taken to reach the goal (i.e., the number of steps) for the MC problem. As an additional benchmark for comparing our outcomes, a control policy is also acquired using the same RL algorithm, this time employing the actual MC environment. In other words, referring to section 4.2.3, we employed as metric the *ultimate performance*, comparing the number of steps required for the resolution of the MC problem (limiting an episode to a maximum duration) with and without our transfer technique.

For our RL experiments, we adopted the DDPG algorithm [72] to learn the control policy using the parameters described in Table 4.3, along with the default ones from Ray's library RLLib 2.2.0 [70]. We conducted 20 training iterations of 600 transitions each.

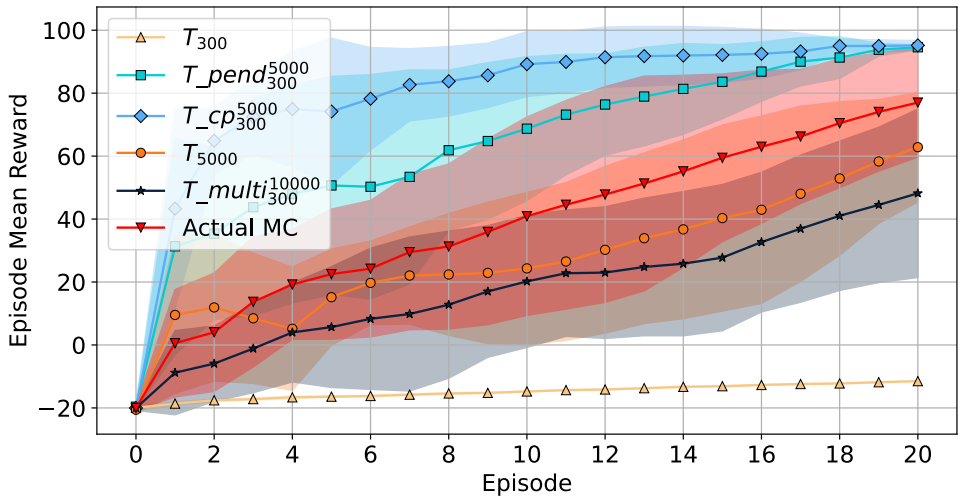


FIG. 4.6: Episode Mean Reward of DDPG Training for each trained generative model and for the actual Mountain Car. (5 runs, 95% c.i.)

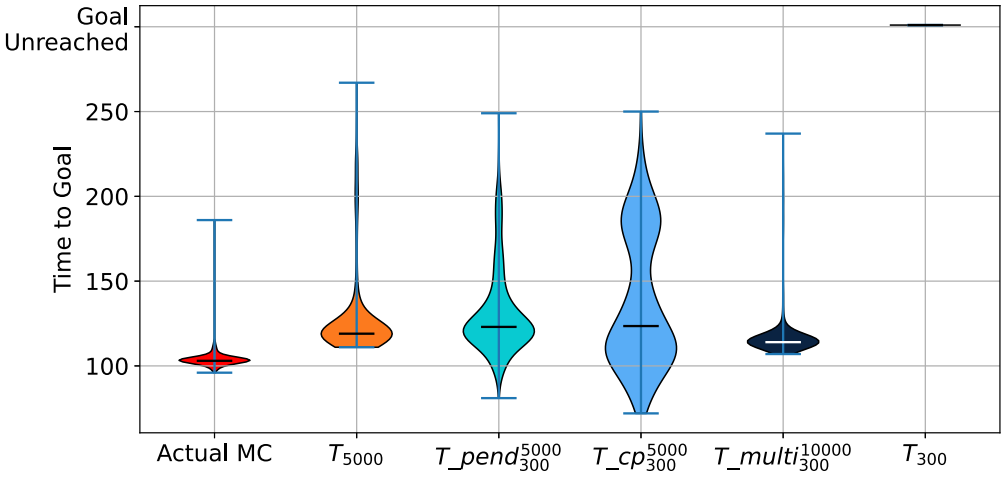


FIG. 4.7: Comparison between learnt control policies over 500 episodes with maximum length of 450 steps.

TABLE 4.3: Parameters used during the Reinforcement Learning process of DDPG.

Discount Factor	Learning Rate	Actor's Units per Hidden Layer	Critic's Units per Hidden Layer	Batch size	Q-learning steps
0.99	0.001	[32,64]	[64, 64]	64	3

Figure 4.6 reports the progression of DDPG's learning and the episode mean reward for each learning iteration. Given that the precision of transition models impacts the agent's reward during learning, direct conclusions from the comparison of each curve in Figure 4.6 might not be straightforward. Nevertheless, a consistent upsurge in the average reward across most policies utilizing the generative models is evident. However, it is notable that the policy trained with model T_{300}, which was trained exclusively with a restricted number of target triplets and without utilizing transfer learning, stands out as the only one that does not exhibit effective learning.

Once we've learned a control policy from each of the generative model, we test them in the actual MC environment.

In Figure 4.7, we show some of the trained policies of each generative model, employing the number of steps the agent requires to complete 500 episodes, each with a maximum length of 450, as a performance metric. We can observe from both figure 4.6 and figure 4.7 that the model T_{300}, trained solely using 300 target triplets, lacks the accuracy necessary to effectively learn a control policy. However, our approach demonstrates the ability to leverage this limited target data, acquiring a mapping capable of converting additional samples from the source task to the target task, thus enabling effective learning. In this evaluation, both $T\_pend_{300}^{5000}$ and $T\_cp_{300}^{5000}$ exhibit similar performance to T_{5000}. These policies consistently reach the goal, with a median time to goal similar to the policy learned from T_{5000}, although displaying slightly higher variance. Notably, combining transformed data from multiple sources with the target data enhances performance. The policy learned from $T\_multi_{300}^{10000}$ displays reduced median and variance for the time to goal compared to policies learned using other models. Regarding the actual MC, even though it is trained in the same test environment, policies learned through other simulators sometimes achieve the goal in fewer steps. This variation might be attributed to the fixed number of training iterations for fair comparison. However, the key evaluation here lies in the average time to goal, where the actual MC exhibits the lowest median compared to every other generative model.

4.4 Partial Conclusions

In this chapter, an introduction to transfer learning within the RL's context has been presented. This investigation began by establishing a distinction between intra-domain and cross-domain transfer learning, setting the stage for a comprehensive understanding of the two scenarios.

Furthermore, a concise overview of the main TL algorithms for both intra-domain and cross-domain settings was presented, describing their own characteristics and critical points. Additionally, the advantages of instance transferring approaches were presented, emphasizing their significance in this domain.

The main objective of this chapter was to present a novel cross-domain instance transfer approach leveraging deep learning techniques, namely GANs and autoencoders, to learn an inter-task mapping only employing a limited amount of target environment data. This methodology aimed to transform source samples into target ones, effectively replicating the dynamics of the target system. Experimental results demonstrate how, by exploiting the proposed method to compute

an inter-task mapping, it is possible to achieve higher prediction accuracy in the estimation of a transition model of the target environment. In particular, experiments showed how this estimated transition model can be employed as a generative model for a RL algorithm to learn a policy when the original target data is insufficient. Moreover, the methodology exhibited adaptability by extending its capabilities to seamlessly combine knowledge from diverse source environments, facilitating multi-source transfers.

Chapter 5

Conclusions and Future Research Avenues

In this thesis, we have tackled the challenge of sample complexity. Leveraging the concepts underlying inverse reinforcement learning and transfer learning, we've introduced two distinct approaches aimed at indirectly reducing the sample complexity, i.e. the number of interactions an agent must undertake with its environment to achieve a satisfactory level of performance. As discussed in the first chapters, the high sample complexity associated with certain RL algorithms poses practical challenges, especially in real-world applications where data collection can be resource-intensive or time-consuming. Addressing the sample complexity problem requires sophisticated learning algorithms that can extract meaningful insights from limited data.

5.1 Research Outcome

In the first part of the thesis, we've introduced an IRL method that consider the finite sample availability in the subsequent forward RL phase during the reward selection process. The core concept involved the selection of both the reward parameters and the discount factor by framing a min-max problem, aiming to minimize the difference between the expert's policy and the learned policy in the subsequent forward learning task. This allowed the algorithm to strike a balance between a potentially sub-optimal reward and the estimation error stemming from using a limited number of samples in the forward learning phase. The conducted numerical simulations highlighted the positive impact of selecting a reward function that considers the available samples in the following forward RL phase, ultimately enhancing performance in cases of sample scarcity and thus reducing the number of required samples to achieve similar performance levels.

In the second part of the thesis, we introduced an adversarial approach for transferring knowledge from various domains by generating synthetic samples (instance transfer). The approach learned an inter-task mapping from a small amount of target environment data to convert source triplets into target ones. This aimed to accurately replicate the dynamics of the target system and estimate a precise transition model. Additionally, our methodology allowed combining knowledge

from various source environments for a smooth transition to multi-source transfers. By utilizing the proposed inter-task mapping method, our experimental results showed that through the transformed samples, higher prediction accuracy in the estimation of target environment's transition model could be achieved. This estimated transition model could be exploited as a generative model to learn a policy through a RL algorithm, specially when the original target data proved insufficient for learning a policy.

5.2 Future Research

Concerning our IRL research, there remains potential for improvement, particularly concerning the expansion of the presented approach to more intricate and demanding environments. There's an opportunity to enhance the methodology to accommodate more complex scenarios and address challenging conditions, thus further advancing the solution's effectiveness and adaptability in more realistic and practical environments.

On the other hand, for our TL research, we identified one important limitation: its reliance on the quality of the transition model employed by our algorithm, which is contingent on the approximation architecture and fitting algorithm used. This particular aspect warrants further investigation in future works. Additionally, in this study, we did not explore the potential application of the inverse inter-task mapping, a side product of our methodology that could be further utilized to enhance the quality of our instance transfer method.

Bibliography

[1] Mountain car continous. https://gymnasium.farama.org/environments/classic_control/mountain_car_continuous/. Accessed: 2023-01-28.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[3] Stephen Adams, Tyler Cody, and Peter A Beling. A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 55(6):4307–4346, 2022.

[4] James S Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). 1975.

[5] Haitham B Ammar, Karl Tuyls, Matthew E Taylor, Kurt Driessens, and Gerhard Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems*, volume 1, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems ..., 2012.

[6] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[7] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6): 26–38, 2017.

[9] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J. Kappen. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Machine Learning*, 91(3):325–349, 2013. doi: 10.1007/s10994-013-5368-1.

[10] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 897–904, 2011.

[11] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[12] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[13] Daniel S Bernstein. Reusing old policies to accelerate learning on new mdps. Technical report, Citeseer, 1999.

[14] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

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

[16] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 182–189. JMLR Workshop and Conference Proceedings, 2011.

[17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[18] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[19] Haoyang Cao, Samuel Cohen, and Lukasz Szpruch. Identifiability in inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12362–12373, 2021.

[20] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine*, 109:101964, 2020.

[21] Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64: 645–703, 2019.

[22] Angelo Damiani, Giorgio Manganini, Alberto Maria Metelli, and Marcello Restelli. Balancing sample efficiency and suboptimality in inverse reinforcement learning. In *International Conference on Machine Learning*, pages 4618–4629. PMLR, 2022.

[23] John M Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

[24] Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear-quadratic control: an introduction*. Simon & Schuster, Inc., 1994.

[25] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 04 2005.

[26] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[27] Xing Fang, Qichao Zhang, Yinfeng Gao, and Dongbin Zhao. Offline reinforcement learning for autonomous driving with real world driving data. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3417–3422. IEEE, 2022.

[28] Amir Massoud Farahmand, Rémi Munos, and Csaba Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems 23 (NIPS)*, pages 568–576, 2010.

[29] Kimberly Ferguson and Sridhar Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series*, page 151, 2006.

[30] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.

[31] Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. *arXiv preprint arXiv:1207.4114*, 2012.

[32] Eliseo Ferrante, Alessandro Lazaric, Marcello Restelli, et al. Transfer of task representation in reinforcement learning using policy-based proto-value functions. In *AAMAS (3)*, pages 1329–1332, 2008.

[33] Angelos Filos. Reinforcement learning for portfolio management. *arXiv preprint arXiv:1909.09571*, 2019.

[34] Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.

[35] Zoubin Ghahramani. Learning dynamic bayesian networks. *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pages 168–197, 1997.

[36] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[37] Ivo Grondman, Lucian Busoniu, Gabriel AD 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.

[38] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[39] Verena Heidrich-Meisner and Christian Igel. Evolution strategies for direct policy search. In *International Conference on Parallel Problem Solving from Nature*, pages 428–437. Springer, 2008.

[40] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), April 2017. ISSN 0360-0300. doi: 10.1145/3054912. URL https://doi.org/10.1145/3054912.

[41] Hitoshi Iima and Yasuaki Kuroe. Swarm reinforcement learning algorithm based on particle swarm optimization whose personal bests have lifespans. In *Neural Information Processing: 16th International Conference, ICONIP 2009, Bangkok, Thailand, December 1-5, 2009, Proceedings, Part II 16*, pages 169–178. Springer, 2009.

[42] Vasile I Istratescu. *Fixed point theory: an introduction*. Springer, 1981.

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

[44] Zhengyao Jiang and Jinjun Liang. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent systems conference (IntelliSys)*, pages 905–913. IEEE, 2017.

[45] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

[46] Olivier Jin and Hamza El-Saawy. Portfolio management using reinforcement learning. *Stanford University*, 2016.

[47] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[48] Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[49] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[50] Zsolt Kalmár and Csaba Szepesvári. An evaluation criterion for macro learning and some results. *TR99-01, Mindmaker Ltd*, 1999.

[51] Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49:193–208, 2002.

[52] Taylor W Killian, Haoran Zhang, Jayakumar Subramanian, Mehdi Fatemi, and Marzyeh Ghassemi. An empirical study of representation learning for reinforcement learning in healthcare. *arXiv preprint arXiv:2011.11235*, 2020.

[53] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013. doi: 10.1177/0278364913495721. URL https://doi.org/10.1177/0278364913495721.

[54] Abi Komanduru and Jean Honorio. On the correctness and sample complexity of inverse reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[55] Abi Komanduru and Jean Honorio. A lower bound for the sample complexity of inverse reinforcement learning. In *International Conference on Machine Learning*, pages 5676–5685. PMLR, 2021.

[56] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013. ISSN 2218-6581. doi: 10.3390/robotics2030122. URL https://www.mdpi.com/2218-6581/2/3/122.

[57] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[58] Steven G Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012.

[59] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[60] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[61] Romain Laroche and Merwan Barlier. Transfer reinforcement learning with shared dynamics. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[62] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[63] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. *Reinforcement Learning: State-of-the-Art*, pages 143–173, 2012.

[64] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551, 2008.

[65] Alessandro Lazaric, Mohammad Ghavamzadeh, and Remi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012.

[66] Filippo Lazzati, Alberto Maria Metelli, and Marcello Restelli. On the sample complexity of inverse reinforcement learning. In *Sixteenth European Workshop on Reinforcement Learning*, 2023.

[67] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in neural information processing systems*, 24, 2011.

[68] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[69] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[70] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

[71] Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*, 2018.

[72] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[73] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989. doi: 10.1007/BF01589116. URL https://doi.org/10.1007/BF01589116.

[74] Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the national conference on artificial intelligence*, volume 21, page 415. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[75] Yong Liu, Yujing Hu, Yang Gao, Yingfeng Chen, and Changjie Fan. Value function transfer for deep multi-agent reinforcement learning based on n-step returns. In *IJCAI*, pages 457–463. Macao, 2019.

[76] 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(10), 2007.

[77] Timothy A Mann and Yoonsuck Choe. Directed exploration in reinforcement learning with transferred knowledge. In *European Workshop on Reinforcement Learning*, pages 59–76. PMLR, 2013.

[78] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

[79] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Machine Learning: ECML 2002: 13th European Conference on Machine Learning Helsinki, Finland, August 19–23, 2002 Proceedings 13*, pages 295–306. Springer, 2002.

[80] 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.

[81] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

[82] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

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

[84] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[85] Theodore J Perkins, Doina Precup, et al. Using options for knowledge transfer in reinforcement learning. Technical report, Citeseer, 1999.

[86] Tung Phan-Minh, Forbes Howington, Ting-Sheng Chu, Sang Uk Lee, Momchil S Tomov, Nanxiang Li, Caglayan Dicle, Samuel Findler, Francisco Suarez-Ruiz, Robert Beaudoin, et al. Driving in real life with inverse reinforcement learning. *arXiv preprint arXiv:2206.03004*, 2022.

[87] Caitlin Phillips. Knowledge transfer in markov decision processes. Technical report, Technical report, McGill University, School of Computer Science, 2006. URL ..., 2006.

[88] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[89] Matteo Pirotta and Marcello Restelli. Inverse reinforcement learning through policy gradient minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[90] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Policy gradient in lipschitz markov decision processes. *Mach. Learn.*, 100(2-3):255–283, 2015. doi: 10.1007/s10994-015-5484-1.

[91] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[92] Emmanuel Rachelson and Michail G. Lagoudakis. On the locality of action domination in sequential decision making. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2010.

[93] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[94] Meisam Razaviyayn, Tianjian Huang, Songtao Lu, Maher Nouiehed, Maziar Sanjabi, and Mingyi Hong. Non-convex min-max optimization: Applications, challenges, and recent theoretical advances. *arXiv:2006.08141*, Aug 2020. arXiv: 2006.08141.

[95] Elsa Riachi, Muhammad Mamdani, Michael Fralick, and Frank Rudzicz. Challenges for reinforcement learning in healthcare. *arXiv preprint arXiv:2103.05612*, 2021.

[96] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[97] Bruno Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. *Advances in Neural Information Processing Systems*, 26, 2013.

[98] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[99] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[100] Alexander A Sherstov and Peter Stone. Improving action selection in mdp's via knowledge transfer. In *AAAI*, volume 5, pages 1024–1029, 2005.

[101] Tianyu Shi, Dong Chen, Kaian Chen, and Zhaojian Li. Offline reinforcement learning for autonomous driving with safety and exploration enhancement. *arXiv preprint arXiv:2110.07067*, 2021.

[102] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

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

[104] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pages 907–917. PMLR, 2022.

[105] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[106] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS)*, pages 1057–1063, 1999.

[107] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112 (1-2):181–211, 1999.

[108] Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886, 2007.

[109] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[110] Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(9), 2007.

[111] Matthew E Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, 2007.

[112] Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II 19*, pages 488–505. Springer, 2008.

[113] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *AAMAS (1)*, pages 283–290, 2008.

[114] Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. Importance weighted transfer of samples in reinforcement learning. In *International Conference on Machine Learning*, pages 4936–4945. PMLR, 2018.

[115] Andrea Tirinzoni, Mattia Salvini, and Marcello Restelli. Transfer of samples in policy search via multiple importance sampling. In *International Conference on Machine Learning*, pages 6264–6274. PMLR, 2019.

[116] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[117] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[118] Nurbaiti Wahid and Mohd Fua'ad Rahmat. Pitch control system using lqr and fuzzy logic controller. In *2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA)*, pages 389–394, 2010. doi: 10.1109/ISIEA.2010.5679436.

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

[120] Chang Wang and Sridhar Mahadevan. Manifold alignment without correspondence. In *IJCAI*, volume 2, page 3, 2009.

[121] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2447–2456, 2018.

[122] Yuanhao Wang and Jian Li. Improved algorithms for convex-concave minimax optimization. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.

[123] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[124] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[125] Han Wen, Hui Li, Zhuang Wang, Xianle Hou, and Kangxin He. Application of ddpg-based collision avoidance algorithm in air traffic control. In *2019 12th International Symposium*

on Computational Intelligence and Design (ISCID), volume 1, pages 130–133, 2019. doi: 10.1109/ISCID.2019.00036.

[126] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. doi: 10.1007/BF00992696.

[127] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning, 2016.

[128] Guang Yang, Feng Zhang, Cheng Gong, and Shiwen Zhang. Application of a deep deterministic policy gradient algorithm for energy-aimed timetable rescheduling problem. *Energies*, 12(18), 2019. ISSN 1996-1073. doi: 10.3390/en12183461. URL https://www.mdpi.com/1996-1073/12/18/3461.

[129] Heng You, Tianpei Yang, Yan Zheng, Jianye Hao, and Matthew E Taylor. Cross-domain adaptive transfer reinforcement learning based on state-action correspondence. In *Uncertainty in Artificial Intelligence*, pages 2299–2309. PMLR, 2022.

[130] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

[131] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. *arXiv preprint arXiv:2012.09811*, 2020.

[132] Yintao Zhang, Youmin Zhang, and Ziquan Yu. Path following control for uav using deep reinforcement learning approach. *Guidance, Navigation and Control*, 1(01):2150005, 2021.

[133] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020. doi: 10.1109/SSCI47803.2020.9308468.

[134] Shao Zhifei and Er Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311, 2012.

[135] Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[136] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[137] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendix A

Test Environments

In the field of reinforcement learning, the availability of reliable and realistic test environments is of utmost importance. These test environments serve as critical tools for evaluating and validating the performance of RL algorithms before their deployment in real-world scenarios. They allow researchers and practitioners to iteratively refine and optimize their RL models, leading to more robust and effective solutions.

The necessity of software test environments in RL arises from the inherent complexity and unpredictability of real-world environments. Unlike supervised learning, where training data can be obtained from labeled examples, RL requires agents to interact with the environment, learn from trial and error, and make decisions based on observed feedback. However, conducting RL experiments directly on physical systems or complex domains can be costly, time-consuming, and potentially risky.

To address these challenges, the development of software test environments specifically designed for RL has gained considerable attention. These environments provide a simulated platform where RL agents can learn, explore, and refine their decision-making processes in a controlled and reproducible manner. By simulating various scenarios, researchers can test and validate their RL algorithms efficiently, enabling faster experimentation and iteration cycles.

This section of this work aims to present the main experimental environments adopted for our work, delving into the importance of using well-established and reliable test environments to ensure the validity and generalizability of the results.

We chose one notable software test environment widely adopted in the RL community: Farama Foundation's Gymnasium [1]. Gymnasium provides a collection of standardized RL environments, encompassing a diverse range of problems and challenges. Moreover, Gymnasium offers a unified interface, making it easier to experiment with new custom environments and different RL algorithms while ensuring reproducibility.

In the following of this section we will discuss the test environment we adopted from a modelling point of view.

A.1 Linear Quadratic Gaussian Control Problem

A Linear Quadratic Gaussian (LQG) [24] controller is a type of optimal control algorithm that combines the principles of linear quadratic (LQ) control and optimal linear estimation (Kalman filtering) to design a controller that can optimize a given system's performance. The LQG controller is an important tool in the field of control engineering, as it is able to provide an optimal solution to many nonlinear control problems in a variety of applications, including aircraft control, robotics, and manufacturing. It is particularly useful in applications where the system's dynamics are known only to a certain degree of accuracy, as the LQG controller can adapt to these uncertainties. Additionally, the LQG controller can handle cases where the system's parameters, such as its gains, vary over time.

The LQG problem can be defined both in discrete and continuous time and applies to both linear time-invariant (LTI) systems as well as linear time-varying systems. For a matter of relevance to the experiments we're going to introduce the Discrete LTI version.

Let's consider a LTI dynamical system disturbed by noises:

$$\mathbf{x}_{i+1} = A\mathbf{x}_i + B\mathbf{u}_i + \mathbf{v}_i \tag{A.1}$$

$$\mathbf{y}_i = C\mathbf{x}_i + \mathbf{w}_i. \tag{A.2}$$

Where $\mathbf{x}_i, \mathbf{u}_i$ and $\mathbf{y}_i$ represents respectively the vector of state, of control inputs [1] and measured outputs at time i. A, B and C defines the dynamic of the system and v_i and w_i are discrete-time Gaussian noise with covariance matrices respectively V_i and W_i perturbing both state and output.

Given the LTI system above and an horizon N, the goal of the LQG problem is to find a sequence of inputs minimizing the following objective function:

$$J = \mathbb{E}\Big[\mathbf{x}_N^\top Q_N \mathbf{x}_N + \sum_{i=0}^{N-1} \mathbf{x}_i^\top Q_i \mathbf{x}_i + \mathbf{u}_i^\top R_i \mathbf{u}_i\Big]. \tag{A.3}$$

Where matrices Q_i are symmetric positive semi-definite and R_i are symmetric positive-definite.

Without digging into the details, it has been proven [24] that, if the matrices Q and R are constants (i.e. $Q_i := Q$ and $R_i := R$) the optimal input u_i^* at time step i is given by:

$$u_i^* = -K_i x_{i-1} \tag{A.4}$$

$$K_i = (B^\top P_i B + R)^{-1}(B^\top P_i A) \tag{A.5}$$

$$P_{i-1} = Q + A^\top P_i A - A^\top P_i B(B^\top P_i B + R)^{-1} B^\top P_i A. \tag{A.6}$$

[1] Since the LQG problem has been defined in the control theory research area, we kept the symbols for state and inputs as they were originally defined. In the RL's context we swapped those variables with states $s = x$ and actions $a = u$.

Equation A.6, known as discrete-time dynamic Riccati equation, for $N \to \infty$, is characterized at the steady state as:

$$P = Q + A^\top PA - A^\top PB(B^\top PB + R)^{-1}B^\top PA. \tag{A.7}$$

Although Equation A.7 may have multiple solutions, the designer is mainly interested to the unique, if exists, stabilizing solution thus deriving the optimal input.

If we look to the infinite horizon LQG problem as an RL environment, we can observe how Equation A.3 can be seen as the negative of a state-action value function to be maximized, where the reward function is an element of the sum $x_i^\top Q_i x_i + u_i^\top R_i u_i$. Thus we can reformulate the LQG problem from an RL point of view, as:

$$\max_{\pi^*} \ \mathbb{E}\Big[\sum_{i=0}^{\infty} r(s_i, a_i)\Big] \tag{A.8}$$

$$s.t.$$

$$a_i = \pi^*(s_i) \tag{A.9}$$

$$r(s_i, a_i) = -s_i^\top Q_i s_i - a_i^\top R_i a_i \tag{A.10}$$

$$s_{i+1} = As_i + Ba_i + \mathbf{v}_i \tag{A.11}$$

A.2 Mountain Car Problem

First introduced by Andrew Moore [81], the mountain car problem is a classic RL problem, in which an agent attempts to climb up a steep hill without the use of energy, Figure A.1. The car, indeed, is underpowered to reach the top of the hill only exploiting its own energy, thus it must learn to control its velocity and acceleration in order to reach the goal.

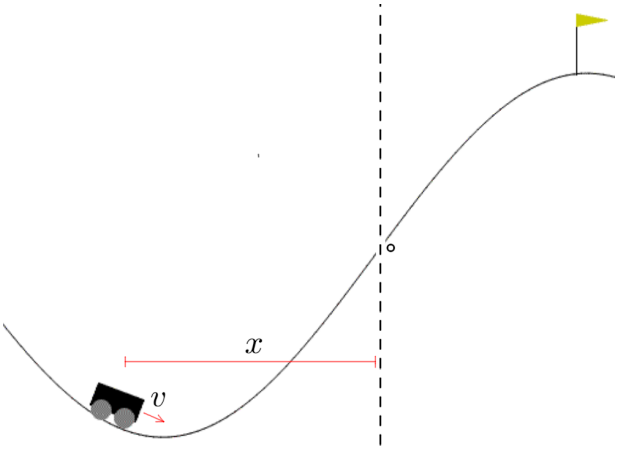


FIG. A.1: Gymnasium's rendering of Mountain Car environment.

The environment adopted in the experiments is described by the following equations taken from the Gymnasium Python library [1], which define the position and dynamics of the car as:

$$v_{t+1} = v_t + ap - \cos(3x_t)g \tag{A.12}$$

$$x_{t+1} = x_t + v_{t+1}. \tag{A.13}$$

Where x is the position, v is the velocity and a is the acceleration of the car (i.e. the action taken by the agent). While $p = 0.001$ and $g = 0.0025$ are environment's parameters reflecting a forces opposing to the car.

In this context, the state and action space are defined as:

$$\mathcal{S} = \{(x, v) \in [-1.2, 0.6] \times [-0.07, 0.07] \subseteq \mathbb{R}^2\} \tag{A.14}$$

$$\mathcal{A} = \{a \in [-1, 1] \subseteq \mathbb{R}\}. \tag{A.15}$$

Where the goal is set to each state having a the position equal to 0.45.

The agent must learn to select an appropriate acceleration at each time step to successfully climb the hill. The rewards associated with each action is a negative value imposed as a punishment for taking large-scale actions. However, when the agent reaches the goal, a bonus of +100 is added to the negative reward of that timestep:

$$r(s, a) = 100 \cdot \mathbf{1}_{[s=\text{goal}]}(s) - 0.1a^2, \tag{A.16}$$

where:

$$\mathbf{1}_{[s=\text{goal}]}(s) = \begin{cases} 1 & \text{if } s[0] \neq 0.45 \\ 0 & \text{otherwise} \end{cases} \tag{A.17}$$

The mountain car problem is a great example of how reinforcement learning can be used to solve complex problems. It is a challenging problem, as the agent must learn how to balance exploration and exploitation to maximize its reward. This environment has been used as a benchmark for reinforcement learning algorithms. It provides a challenging environment for agents to learn and adapt to, while providing valuable insights into how reinforcement learning can be applied to real-world problems. Many different algorithms have been tested on the mountain car problem, including Q-learning, SARSA, and policy-gradient methods.

A.3 Cart Pole Control Problem

Among all the physics-inspired challenges, the Cart Pole problem stands as an intriguing test of balance and control. The problem involves a cart with a pole attached to it, Figure A.2, able to move along a 2-dimensional frictionless track. The objective is to maintain the pole in an upright position for as long as possible while the cart moves back and forth entirely subject to the force of gravity. Despite its apparent simplicity, this problem serves as an engaging platform for exploring the delicate interplay between gravity, inertia, and stability.

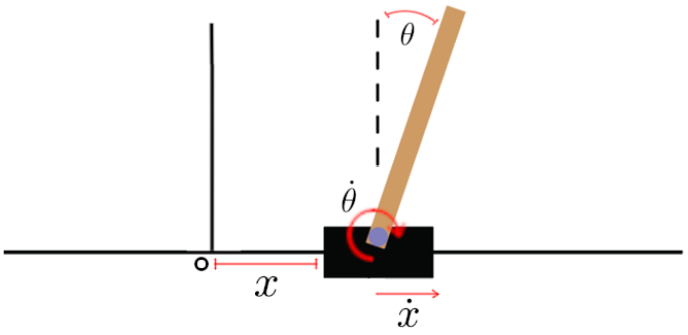


FIG. A.2: Gymnasium's rendering of Cart Pole environment.

To comprehend the intricacies of the Cart Pole problem, we must explore its key elements. Gravity plays a pivotal role, constantly exerting a downward force on the pole. Inertia, on the other hand, influences the pole's tendency to resist changes in its state of motion. The cart's movement introduces additional complexity, as the pole's balance must be actively controlled to counteract the effects of the cart's acceleration and deceleration.

With respect to the Mountain Car problem, Cart-Pole has a complex dynamic:

$$
\begin{aligned}
\ddot{\theta}_{t+1} &= \frac{g \sin\theta_t - \frac{F_t + m_p l \dot{\theta_t}^2 \sin\theta_t}{m_p + m_c} \cos\theta_t}{l(\frac{4}{3} - \frac{m_p cos^2 \theta_t}{m_p + m_c})} \\
\ddot{x}_{t+1} &= \frac{F_t + m_p l(\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t)}{m_p + m_c} \\
\dot{x}_{t+1} &= \dot{x}_t + \tau \ddot{x}_t \\
x_{t+1} &= x_t + \tau \dot{x}_t \\
\dot{\theta}_{t+1} &= \dot{\theta}_t + \tau \ddot{\theta}_t \\
\theta_{t+1} &= \theta_t + \tau \dot{\theta}_t
\end{aligned}
\tag{A.18}
$$

Since it is just marginal for the scope of this thesis, we invite the reader to refer to [34] for further details about the physical dynamic of the system. What we are interested in are the variables determining the state of the system, the actions we can apply and the reward we get.

The (observable) state of the cart is composed by 4 variables describing the cart's position on the track (x), cart's velocity ($\dot{x}$), pole's angle with respect to the cart (θ) and pole's angular velocity ($\dot{\theta}$). The action, instead, is a single discrete variable indicating the direction (left or right) of a fixed force (F_t in Equations A.18) that is applied to the cart for balancing the pole. We can summarize state space and action space (with the gymnasium's ranges) as:

$$\mathcal{S} = \{(x, \dot{x}, \theta, \dot{\theta}) \in [-4.8, 4.8] \times \mathbb{R} \times [-0.418, 0.418] \times \mathbb{R} \subseteq \mathbb{R}^4\} \tag{A.19}$$

$$\mathcal{A} = \{0, 1\}. \tag{A.20}$$

As mentioned, the agent must learn to balance the pole on top of the cart and keep it upright as long as possible. The system's reward, in order to solve this problem, is given by:

$$r(s,a) = \begin{cases} 1 & \text{if } s[2] \in [-0.209, 0.209] \\ 0 & \text{otherwise} \end{cases} \tag{A.21}$$

When the reward is equal to 0, it describes the situation in which the pole has an angle that can't be corrected by moving the cart only. On the other hand, the reward is equal to 1 when there is still the chance of balancing the pole.

The Cart Pole problem offers an exciting opportunity to explore the fundamental principles of physics. Concepts such as torque, angular momentum, and center of mass come into play when analyzing the dynamics of the system.

A.4 Inverted Pendulum Swing-Up Problem

Similar to the Cart Pole problem discussed above, there is the Inverted Pendulum Swing-Up Problem. The setup consists of a pendulum arm, which is initially positioned in an inverted state, meaning the pendulum is upside down. The goal is to apply controlled forces to the pendulum such that it swings up and eventually stabilizes in a vertical position. The pendulum arm is connected to a pivot point, allowing it to rotate freely, and it is subject to external forces and torques, Figure A.3.

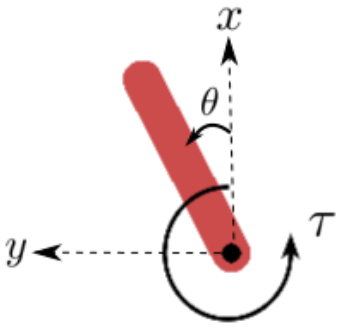


Fig. A.3: Gymnasium's rendering of the inverted pendulum swing-up problem.

Physics' equations describing the dynamic of the pendulum are the following:

$$\dot{\theta}_{t+dt} = \dot{\theta}_t + \left(\frac{3g}{2l}\sin\theta_t + 3\frac{\tau_t}{ml^2}\right)dt \tag{A.22}$$

$$\theta_{t+dt} = \theta_t + \dot{\theta}_{t+dt}dt. \tag{A.23}$$

Gymnasium's library for the pendulum considers the observable part of the state as composed by 3 variables: $\sin(\theta)$, $\cos(\theta), \dot{\theta}$. Here $\dot{\theta}$ is bound in the range $[-8, 8] \in \mathbb{R}$. Actions the RL agent can use are torques directly applied on the free end of the pendulum, τ_t in Equation A.22, and

bound in the range of $[-2, 2] \in \mathbb{R}$. Summarizing:

$$\mathcal{S} = \{(\sin\theta, \cos\theta, \dot{\theta}) \in [-1, 1] \times [-1, 1] \times [-8, 8] \subseteq \mathbb{R}^3 \mid \theta \in [-\pi, \pi]\} \tag{A.24}$$

$$\mathcal{A} = \{\tau \in [-2, 2] \subseteq \mathbb{R}\}. \tag{A.25}$$

Similarly to the Cart Pole problem, here we want to stabilize the pole in a vertical position but using as little energy as possible. For this reasons, the reward in this problem is composed as follow:

$$r(s, a) = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2). \tag{A.26}$$

Where $\theta = \text{atan2}(s[0], s[1])$. As we can observe, this reward function is a penalty composed by 3 parts: the first one penalizes having an angle different from 0 (i.e. standing upright), the second one penalizes having an angular velocity different from 0 (i.e. the pole is moving) and the third one penalizes the application of a strong torque.