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

Opponent Identification in Multi-Agent Reinforcement Learning

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

Supervisor: Prof. Marcello Restelli
Co-supervisor: Dr. Giorgia Ramponi

Master Graduation Thesis by:
Davide Spinelli
Student ID n. 900229

Academic Year 2019-2020

To all the people I met in this incredible journey.

Acknowledgements

I would like to thank Prof. Marcello Restelli, for the opportunity to work on this project and for the important insights you have given me in these months, and Dr. Giorgia Ramponi, for the patience shown to me especially in the initial stages during the understanding of the subject and for the constant assistance during the development of this work. Furthermore, I would like to thank both of them for their availability which, despite the events and the distance, has never failed.

I will always be grateful to my parents and family for supporting me morally and materially all these years, allowing me to focus full-time on my university career. The certainty of being able to count on their support was fundamental to overcome difficult moments and reach the goals I had set for myself. I want to dedicate a particular thanks to my parents for their teachings, thanks to them I grew up into the person I am today.

A special thanks also goes to my dearest friends, whose company over the years has brought countless moments of joy into my life. Without them, this experience would not have been as enjoyable.

Finally, I would like to thank all the people who accompanied me for some parts of this journey, my classmates during the Bachelor's and Master's, the 'big family' of BEST, my clubmates at UCL and my flatmates in Berlin. Thanks to them and to all the activities carried out together I was able to experience moments and build memories that will stay with me forever.

Davide

Milan, 15 December 2020

Ringraziamenti

Desidero ringraziare il Prof. Marcello Restelli, per la possibilità di lavorare su questo progetto e per le importanti indicazioni fornitemi in questi mesi, e la Dott. Giorgia Ramponi, per la pazienza dimostratami soprattutto nelle fasi iniziali di comprensione della materia e per la costante assistenza durante lo sviluppo di questo lavoro. Inoltre, vorrei ringraziare entrambi per la disponibilità che, nonostante gli avvenimenti e la distanza, non è mai venuta meno.

Sarò per sempre grato ai miei genitori e alla mia famiglia per avermi supportato moralmente e materialmente in tutti questi anni, permettendomi di concentrarmi sulla carriera universitaria a tempo pieno. La certezza di poter contare sul loro sostegno è stata fondamentale per superare momenti complicati e raggiungere i traguardi che mi ero prefissato. Un particolare ringraziamento lo voglio dedicare ai miei genitori per i loro insegnamenti, grazie ad essi sono cresciuto nella persona che sono oggi.

Un grazie speciale va anche ai miei più cari amici, la cui compagnia in questi anni ha portato innumerevoli momenti di gioia nella mia vita. Senza di loro quest'esperienza non sarebbe stata altrettanto piacevole.

Infine, vorrei ringraziare tutte le persone che mi hanno accompagnato per alcuni tratti di questo viaggio, i miei compagni della triennale e della magistrale, la 'grande famiglia' di BEST, i miei compagni della UCL e i miei coinquilini a Berlino. Grazie a loro e a tutte le attività svolte insieme ho avuto modo di vivere momenti e costruire ricordi che resteranno con me per sempre.

Davide

Milano, 15 dicembre 2020

Contents

Mathematical notation

$\mathcal{S}$: the state space;
$\mathcal{S}_0$: the initial state distribution;
$\mathcal{A}$: the action space;
$\mathcal{R}$: the reward function;
$\mathcal{P}$: the transition model;
$\mathcal{T}$: the set of trajectories;
$\mathcal{J}$: the Jacobian;
$\mathcal{J}_o$: the anti-diagonal blocks of $\mathcal{J}$;
$\mathcal{M}$: the set of algorithms;
$\mathcal{L}$: the value of the log-likelihood;
$\nabla\mathcal{H}$: the gradient of the Hamiltonian function;

T: the horizon;
J: the expected return;
I: the identity matrix;
S: the Potential part of the game;
A: the Hamiltonian part of the game;
S_u^{-1}: the Positive Truncated Inverse of S;

$\mathbb{R}$: the set of real numbers;
$\mathbb{N}$: the set of natural numbers;
$\mathbb{E}$: the expectation;

s: the state;
a: the action;
r: the reward;
p: the transition probability;
t: the time step;
k: the learning step;

v: the state-value function;
q: the action-value function;
d: the number of dimensions of the parameter's vector;
n: the number of agents;

Π: the policy space;
Λ: the likelihood ratio test;
π: the policy;
γ: the discount factor of the reward;
τ: the trajectory;
$\boldsymbol{\theta}$: the policy parameters;
α: the learning rate;
ξ: the simultaneous gradient;
ν: the Behavioral Cloning learning rate;
ϕ: the feature vector of the reward;
ψ: the cumulative discounted feature vector of the reward;
$\boldsymbol{\omega}$: the weight vector of the reward;
ζ: the threshold for the likelihood ratio test;
λ: the likelihood ratio;
ρ: the Importance Sampling ratio;
β: the order of the Renyi divergence;

$\mathscr{L}()$: the likelihood function;
$\ell()$: the log-likelihood function;
$PD()$: the probability distribution function;
$Upd^m()$: the update function using algorithm m;
$MIS()$: the Multiple Importance Sampling function;

$\cdot$: the placeholder operator;
$\overline{\cdot}$: the estimated value;
$\widehat{\cdot}$: the optimized value;
$\frown$: the concatenation operator;
$\| \cdot \|$: the 2-norm;
$\cdot_{x:y}$: the history of values from x to y;
$D_\beta(\cdot \| \cdot)$: the Renyi divergence;
$D_{KL}(\cdot \| \cdot)$: the Kullback-Leibler divergence;
$\cdot^{superscript}$: the agent and/or the candidate algorithm;
$\cdot_{subscript}$: the time step and/or the learning step.

List of Figures

List of Algorithms

Abstract

Imitation Learning is the problem of recovering information about other agents' strategies and goals. The research in this area mainly focuses on imitating the demonstrations of expert agents. However, in Multi-Agent environments, the agents usually learn simultaneously, thus, imitating a non-optimal policy does not provide the information desired and, in most cases, does not lead to high payoffs. Recent works overcame this assumption and developed imitation techniques on learning agents in Single-Agent environments.

In this work, evolving from these results, we develop a technique to estimate the reward function of the opponent agent in a Multi-Agent environment where the agents are still learning. We present the developed approach and the results obtained by applying the method in different scenarios.

Moreover, during the learning phase, agents update their strategies following specific algorithms. In this document, within a Multi-Agent environment, we present a technique to identify the algorithm used by the other agent from a finite set of possible algorithms. Above this technique, we further develop an exploration strategy to facilitate the identification of the algorithm, maximizing the Kullback-Leibler divergence of the estimated future strategies of the agent following the different algorithms. We present the results of these identification techniques over a set of gradient-based algorithms and the result of a possible application in which the agent plays a Best Response strategy to the identified opponent.

Sommario

L'apprendimento per imitazione rappresenta il problema di recuperare le informazioni riguardo le strategie e gli obiettivi degli altri agenti. La ricerca in quest'area si è concentrata principalmente nell'imitazione delle dimostrazioni di agenti esperti. Ciononostante, negli ambienti Multi-Agente, normalmente gli agenti apprendono in simultanea, quindi, imitare strategie sub-ottimali non fornisce le informazioni volute e, nella maggior parte dei casi, non porta a maggiori ricompense. Recenti lavori hanno superato questa supposizione e hanno sviluppato tecniche di imitazione di agenti in fase di apprendimento in ambienti a Singolo-Agente.
In questo lavoro, costruendo sopra questi risultati, abbiamo sviluppato una tecnica per stimare la funzione di rinforzo dell'agente avversario in un ambiente Multi-Agente, dove gli agenti stanno ancora apprendendo. Presentiamo l'approccio sviluppato e i risultati ottenuti applicando il metodo in vari scenari.
Inoltre, durante la fase di apprendimento, gli agenti aggiornano le loro strategie seguendo algoritmi specifici. In questo testo, considerando un ambiente Multi-Agente, presentiamo una tecnica per identificare l'algoritmo usato dall'altro agente all'interno di un set finito di possibili algoritmi. Partendo da questa tecnica, abbiamo sviluppato una strategia di esplorazione che faciliti l'identificazione dell'algoritmo attraverso la massimizzazione della divergenza di Kullback-Leibler tra le strategie future stimate che l'agente avrà seguendo i vari algoritmi. Presentiamo i risultati di queste tecniche di identificazione applicate ad un set di algoritmi a gradiente e il risultato di una possibile applicazione in cui l'agente gioca seguendo la miglior strategia di risposta nei confronti dell'avversario identificato.

Chapter 1

Introduction

Reinforcement Learning is a machine learning method in which an agent learns to increase its payoff by experiencing the environment, following a trial-and-error approach. In many cases, there is more than one agent that interacts with the environment at the same time. In some cases, the agent does not start with prior information on the other agents and, due to this lack of knowledge, the agent's learning process is more difficult, so the ability to model the other agents is of great importance. Imitation Learning is the field that aims at retrieving information about other agents' strategies and goals. In Imitation Learning we can distinguish two techniques: Behavioural Cloning (BC), which has the task of recovering the strategy, and Inverse Reinforcement Learning (IRL), which has the task of recovering the reward function characterizing the goals. The agent needs to obtain this information to devise its strategy to achieve its objective. In the meantime, it must keep into consideration the behavior of the other agents in the environment and plan its interactions with them accordingly.
Another useful information about the other agents is the updating method they are using while learning the environment. The updating method is the algorithm used by each agent to choose how to interact with the environment and how to devise new strategies based on the observations it receives. An agent recovering this information could precisely compute the strategies the other agents will use in the following step and use this knowledge to exploit their behaviors for its own goal. In a competitive environment, this means that the agent could figuratively be always "one step ahead" of its competitors while, in a cooperative environment, the agent could devise strategies to influence the other agents to facilitate cooperation among them.

1.1 Contribution

The contribution of this document is two-fold. First, we apply the IRL technique used in [48] to the Multi-Agent environment, specifically a 2-agent environment, and devise two new techniques for other, more complex, algorithms. This is relevant to the literature since most of the Imitation Learning methods rely on the assumption that the other agent is already an expert while, in Multi-Agent environments, this is often not the case. Then, we evaluate both techniques in a simple bimatrix game and a more complex continuous gridworld. Second, we propose a new method to identify an algorithm from a given selection used by a learning agent, analyzing its different interactions with the environment during the process. Then we expand our method to include an exploration technique which can reduce the time needed for the identification of the algorithm and that, paired with a Best Response strategy, could lead to a higher payoff. The relevance of this approach relies on the novelty of the information recovered since, to the best of our knowledge, no other method in the literature identifies the update rule used by the other agent.

1.2 Document outline

This thesis is organized as follows.

In Chapter 2 we present the background knowledge needed to follow this work. We start presenting Reinforcement Learning and Markov Decision Processes, then proceed with Multi-Agent environments, Stochastic Games and a possible classification of the algorithms used in this environment. Then, we present Imitation Learning, Behavioural Cloning and Inverse Reinforcement Learning, and more general concepts like Likelihood functions, Importance Sampling and Renyi and Kulback-Leibler divergences. We conclude with an analysis of previous works in the literature related to the identification of other agents' algorithms.

In Chapter 3, we start presenting an extension of the algorithm LOGEL [48] to perform IRL in a 2-agent environment while the opponent is learning with a gradient-based algorithm. Then we consider two variations of this scenario with algorithms like IGA-PP [72] and LOLA [19] and devise a method to recover the reward function of an opponent that uses these algorithms.

Then, in Chapter 4, we evaluate the performances of our methods and

we present the advantage of using an algorithm-specific IRL method on two environments like Matching Pennies and a continuous gridworld.

In Chapter 5, we present a method to passively identify the algorithm used by another learning agent from a finite set of possible algorithms. Then, we introduce an exploration technique to choose a strategy that induces the opponent to reveal its learning algorithm, thus significantly reducing the time needed for the agent to identify its opponent's algorithm.

Next, in Chapter 6, we present the performance obtained by the passive algorithm identification method and the advantage of using the active algorithm identification method as an exploratory strategy towards this objective both in Matching Pennies and in a continuous gridworld.
We conclude with a performance evaluation of an agent using the algorithm identification method to identify the opponent's algorithm and then exploit this knowledge to compute a Best Response and obtain a higher payoff.

In Chapter 7, we draw conclusions by summarizing the contributions of this document and the results obtained. Moreover, we present possible directions for future developments of this work.

To conclude, in Appendix A, we present a variation of Littman's soccer that shows a cyclical behavior similar to Matching Pennies, useful for further experiments, and, in Appendix B, we include more results of the algorithm-specific IRL method using LOLA in Matching Pennies, similar to the ones presented in Section 4.2.

Chapter 2

Preliminaries and related works

In this chapter, we present an introduction to the main concepts that will be used in the following chapters. We start presenting Reinforcement Learning (RL, [60]) in Section 2.1 and Markov Decision Processes (MDP, [47]) in Section 2.1.1, then proceed with Multi-Agent (MA) environments in Section 2.2, Stochastic Games (SG, [57]) in Section 2.2.1 and their algorithms in the following sections, and conclude with Imitation Learning in Section 2.4, addressing Behavioral Cloning (BC, [3]) in Section 2.4.1 and Inverse Reinforcement Learning (IRL, [39]) in Section 2.4.2, and more general concepts like likelihood in Section 2.5, Importance Sampling (IS, [27]) in Section 2.6 and Renyi and Kulback-Leibler divergences respectively in Sections 2.7 and 2.7.1.
These preliminaries are based on [60],[18] and [13].

2.1 Reinforcement Learning

In Reinforcement Learning (RL), the agent learns by trial and error, performing actions to improve the numerical reward signals it receives from the environment. Moreover, the environment can have delayed rewards, in which the actions of the agent do not immediately affect the rewards it receives, so the agent learns how to improve its cumulative reward in the long term.

In computer science, a reinforcement learning system has a policy, a reward function and a model. A policy π is a mapping from each state s to

the probabilities of choosing each possible action a available in that state. Solving a reinforcement learning problem means finding the optimal policy π^* that maximize the cumulative reward. All policies that are better or equal than all the others are called optimal and there always exists at least one.
Given the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta(\mathbb{R})$, where $\mathcal{R}(s, a, s')$ is the reward obtained by choosing action a in state s and ending in state s', a policy space Π and the optimal policy $\pi^* \in \Pi$, π^* satisfies:

$$\mathcal{R}_{\pi^*}(s, a, s') \geq \mathcal{R}_{\pi}(s, a, s'), \quad \forall \pi \in \Pi$$

The knowledge or not of the model yields to the first of many ways in which RL algorithms can be classified.
Reinforcement Learning algorithms with a model are called model-based, otherwise, they are called model-free.

Another classification is based on the difference between the target policy, the policy learned, and the behavior policy, the policy used to interact with the environment.
This dichotomy yields two algorithms:

- on-policy, in which the two policies coincide;
- off-policy, in which the target policy is learned from data obtained using the behavior policy.

Another difference is due to the relation between experience and learning and depends on the concept of time. Time is represented as a sequence of natural numbers $\mathbb{T} = 0, 1, \ldots, t, \ldots, T$ where t is a step and T is the horizon which can be finite, $T \in \mathbb{N}^+$, or infinite, $T \in \infty$. In applications where the horizon is finite, we have finite sequences of states and actions that we call episodes. This concept leads to another classification in which we have:

- Continuous learning, in which the policy is updated online at each step;
- Episodic learning, in which the policy is updated in between episodes of experience;
- Batch learning, in which the experience is collected with a behavior policy and used to learn the target policy.

In this document, we will mainly consider on-policy, episodic, finite-horizon, model-free learning. The episodes have a constant length of T steps, which means that after T actions following policy π the simulation is stopped.

One of the biggest challenges in RL is the trade-off between *exploration*

and *exploitation*. Since the agent has to try different actions to understand their rewards and improve its performance it has to choose each time between exploring to find better actions and greedily exploiting its actual knowledge to receive what it believes is the highest possible reward.

2.1.1 Markov Decision Processes

A Markov Decision Process (MDP, [47]) is a formalization of a problem in which an agent must interact with an environment through actions in order to maximize a reward. MDPs are a useful abstraction to problems in which a goal-oriented agent improves its performances by taking different actions in different states and accumulating rewards, learning to focus on the cumulative reward instead of the immediate reward only. We consider discrete MDPs in which at every time step the environment is in a given state and an agent has to choose an action that will lead to a new state, under a transition probability, and receiving a reward. MDPs are an example of sequential decision making problems since the actions taken do not only influence the immediate rewards but also future states and rewards.

Definition 2.1. An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{S}_0 \rangle$ where:

- $\mathcal{S}$ is the state space;
- $\mathcal{A}$ is the action space;
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition model, where $\mathcal{P}(s'|s,a)$ is the probability that starting from state s and taking action a the resulting state is s';
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, where $\mathcal{R}(s,a,s')$ is the reward obtained by choosing action a in state s and ending in state s';
- $\gamma \in (0,1]$ is the discount factor of the reward;
- $\mathcal{S}_0$ is the initial state distribution.

An MDP is called *terminal* if there is a terminal state in which no other state can be reached and which usually yields reward zero.

A *trajectory* τ is a sequence of state-action-reward $\tau = (s_t, a_t, r_t)_{t=0}^{T(\tau)}$ where $T(\tau)$ represents the horizon of the trajectory τ. We call the set of all trajectories $\mathcal{T} = (\mathcal{S}, \mathcal{A}, \mathcal{R})^T$.

We denote with $\pi(a|s)$ the probability of choosing action a when in state s. We say that an agent follows a policy π if it chooses its actions based on

these probabilities. A policy is called *deterministic* if for each state s there exist some action a such that $\pi(a|s) = 1$.
MDPs satisfy two important properties:

- *Markov property*: each state depends only on the previous state and the action taken. This means that each state fully represents the environment and previous states and actions do not influence the future state;
- *Stationarity*: the dynamics of the environment does not change over time.

The goodness of a policy π is expressed as the utility function J_π and is called the *expected return* for the policy π. J_π is the expectation of the sum of the rewards collected along every trajectory τ discounted by a factor γ:

$$J_\pi = \mathop{\mathbb{E}}_{\tau}\left[\sum_{t=0}^{T(\tau)} \gamma^t \mathcal{R}(s_{\tau,t}, a_{\tau,t}, s'_{\tau,t})\right]$$

Having a measurement of performance, the goal of the agent is to maximize it, thus this problem can be seen as the optimization problem:

$$\pi^* = \arg\max_{\pi} J_\pi$$

where π^* is the optimal policy. In case the policy is parameterized, i.e. policy π depends on parameters $\boldsymbol{\theta}$, the expected return will also depend on $\boldsymbol{\theta}$ and this dependency is expressed with $J(\boldsymbol{\theta})$. Thus, the optimization problem can be rewritten as:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.1}$$

2.1.2 Algorithms

In this setting, algorithms can be classified into two main categories: Value-Based algorithms, which use the value function (action-value function) to find the optimal policy, and Policy Search algorithms, which directly search the optimal policy in the policy space.

2.1.2.1 Value-based algorithms

In value-based algorithms, the agent computes a value function (action-value function) that keeps track of which states (which actions in which states) are better in the long-term to find the optimal policy.

This value function represents the goodness of each state (state-action pair) with respect to the utility function.
The value function (action-value function) can be expressed with arrays or tables, with a value for each state (state-action pair). This method is called *tabular*. In many real-world examples, however, there are far too many states or actions to fit in a table, so we have to approximate them with a parameterized function.
In the case of MDPs, the utility function is the Expected Return J_π and the state-value function is typically the expected return of the agent if it follows the policy π starting from state s:

$$v^\pi(s) = \mathop{\mathbb{E}}_{\tau}\left[\sum_{t=t'}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t} r_t) \middle| s_{\tau,t'} = s\right] \quad \forall s \in \mathcal{S}$$

while the action-value function $q^\pi(s,a)$ is the expected return of the agent if it follows the policy π starting from state s and taking action a:

$$q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau}\left[\sum_{t=t'}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t}) \middle| s_{\tau,t'} = s, a_{\tau,t'} = a\right] \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

In other words, the state-value function represents the total reward that an agent can expect by being in state s, while the action-value function represents the total reward that an agent can expect by being in state s and choosing action a.
The *Bellman equation* represents the relationship between the value of a state (state-action pair) and its subsequent states (state-action pairs). The state-value function v_π and the action-value function q_π are the unique solution to their respective Bellman equations, defined as:

$$v^\pi(s) = r(s) + \gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v^\pi(s') \tag{2.2}$$

$$q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(s',a') q^\pi(s',a') \tag{2.3}$$

The relation between the optimal policy and the optimal value function is that all the optimal policies share the same optimal state-value function defined as:

$$v^*(s) \doteq \max_\pi v^\pi(s)$$

Algorithm 1: Q-Learning

Input: learning rate α
Initialize $q(s, a)$ arbitrarily for all states s and actions a
for *each episode* **do**
  Initialize starting state s
  for *each step in episode, until s is terminal* **do**
    Choose action a from possible actions in state s using
     action-value function $q(s, \cdot)$
    Take action a, observe s' and $R(s, a, s')$
    $q(s, a) = q(s, a) + \alpha[R(s, a, s') + \gamma \max_{a'} q(s', a') - q(s, a)]$
    $s = s'$
  end
end

or the same action-value function defined as:

$$q^*(s, a) \doteq \max_{\pi} q^{\pi}(s, a)$$

If we know the dynamics of the MDP, have sufficient computational resources and it satisfies the Markov property, we can find the optimal policy just selecting in each state the action with the maximum value. This is called the *greedy* policy.
There are many algorithms that approximate Q^*: model-based using Dynamic Programming [7][47], on-line model-free methods that estimate the value function [68][40], and methods that learn a model and then use model-based algorithms [59][36].
In *Q-Learning* [68], the Bellman equation (2.3) is used to update the action-value function during a sequence of episodes to approximate Q^*. The updates are performed over the experience of the environment and at each step, an action is chosen using a strategy derived from Q and the reward and subsequent state are observed (see Algorithm 1).
Given the current learning step t, after performing action a from state s and observing the ending state s' and the reward $R(s, a, s')$, the update can be computed as:

$$q_{t+1}(s, a) = q_t(s, a) + \alpha_t[R(s, a, s') + \gamma \max_{a'} q_t(s', a') - q_t(s, a)] \qquad (2.4)$$

where α_t is the learning rate, which indicates how much the current values of Q are updated towards the perceived sample and that typically decrease

with time to learn faster at the beginning of the learning and perform small changes, or fine-tuning, afterward.
Since Equation (2.4) does not depend on the knowledge of the reward function $\mathcal{R}$ or the transition function $\mathcal{P}$, Q-Learning is a model-free method.
Q converges to Q^* under the following conditions [68][25][65]:

- Distinct values of the action-value function are saved and updated at each learning step,
- The agent keep exploring all possible state-action pairs with non-zero probability, $\pi(s,a) > 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$,
- The learning rates sums to infinity while the sum of their squares is finite (Robins-Monro condition [52]).

The second condition can be achieved with an ϵ-greedy exploration, in which at each step a random action is chosen with probability $\epsilon \in (0,1)$, or with a Boltzmann strategy which in state s selects action a with probability:

$$\pi(s,a) = \frac{e^{\frac{q(s,a)}{\tau_\pi}}}{\sum_{a'} e^{\frac{q(s,a')}{\tau_\pi}}} \tag{2.5}$$

where $\tau_\pi > 0$ is the temperature that controls the exploration's randomness. Actions with a higher action-value have a greater chance to be chosen. When $\tau_\pi \to \infty$ actions tend to be chosen completely random, on the other hand, when $\tau_\pi \to 0$ actions tend to be chosen with a greedy strategy.

2.1.2.2 Policy Search algorithms

Unlike value function (action-value function) methods, direct policy search methods look for the optimal policy in the policy space. In model-free policy search algorithms, this is done through the use of parameterized policies determined by a set of parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ in which $\pi_{\boldsymbol{\theta}}(a|s)$ is the probability of taking action a in state s with vector parameter $\boldsymbol{\theta}$. These policies are continuously evaluated and updated.
The algorithms are divided into three main areas: policy exploration, policy evaluation and policy update.
Policy exploration strategies can search in action space or parameter space and they can also be episode-based or step-based.
If the exploration is in the parameter space, a noise ϵ is added as a perturbation to the parameter vector $\boldsymbol{\theta}$. Instead, if the exploration is in the action space then the noise is directly added to the policy π.
If the exploration is episode-based, the perturbation is added at the begin-

Algorithm 2: Model-free policy gradient method

Output: $\boldsymbol{\theta}_K \sim \boldsymbol{\theta}^*$
Initialize $\boldsymbol{\theta}_1$ arbitrarily
for *k = 1,2,...,K learning steps* **do**
  Generate trajectories $\mathcal{T}_k$
  Estimate gradient $\overline{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)}$
  Update policy parameters $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \overline{\nabla J(\boldsymbol{\theta}_k)}$
  (Eventually) Update learning rate α_k
end

ning of the episode and kept constant for the whole duration of the episode. Instead, if the exploration is step-based, then a different independent noise is added at each step.

Policy evaluation strategies can be step-based or episode-based.
Episode-based evaluation strategies use the expected return of the whole episode as a performance function of the policy. Instead, step-based evaluation strategies estimate the quality of the individual actions as the expected return that the agent can receive from that state choosing that action.

In this document, for the policy update, we will use a *Policy Gradient* [71][41][42] step-based likelihood-ratio method [61].

In model-free policy gradient methods, a performance based on the policy parameters is needed. Their goal is to find $\boldsymbol{\theta}^*$ such that: $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ where $J(\boldsymbol{\theta})$ is the performance measure.
To obtain $\boldsymbol{\theta}^*$, these methods update the policy parameters following gradient ascent in J:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \overline{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)} \tag{2.6}$$

where α is the learning rate and $\overline{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)}$ is the estimate of the gradient of the expected return J_k at step k with respect to its parameters $\boldsymbol{\theta}_k$ (see Algorithm 2).
In MDPs, being $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ the gradient of the expected return, it can be rewritten as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau} \left[\nabla_{\boldsymbol{\theta}} \log \mathcal{P}_{\boldsymbol{\theta}}(\tau) \mathcal{R}(\tau) \right] \tag{2.7}$$

where $\nabla_{\boldsymbol{\theta}} \log \mathcal{P}_{\boldsymbol{\theta}}(\tau)$ can be computed as:

$$\nabla_{\boldsymbol{\theta}} \log \mathcal{P}_{\boldsymbol{\theta}}(\tau) = \sum_{t}^{T(\tau)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s_{\tau,t}, a_{\tau,t})$$

Algorithm 3: GPOMDP update rule

Input: Policy parameters $\boldsymbol{\theta}$
Data: Trajectories $\mathcal{T}_{i=1,\dots,N}$
Output: $\nabla^{GP}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
for *each time step* t **do**
  for *each parameter* j **do**
    //estimate the baseline

$$b^{GP}_{j,t} = \frac{\sum_{i}^{N}\left(\sum_{t'=0}^{t}\nabla_{\boldsymbol{\theta}_j}\log\pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)^2\gamma^t R(s_{\tau_i,t},a_{\tau_i,t})}{\sum_{i}^{N}\left(\sum_{t'=0}^{t}\nabla_{\boldsymbol{\theta}_j}\log\pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)^2}$$

  end
end
for *each parameter* j **do**
  //estimate the gradient

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

end

so that we can rewrite:

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

To estimate the policy gradient we used a likelihood ratio method in which b is a baseline used to reduce variance, as shown in [71], hence we can rewrite Equation (2.7) as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau}\left[\nabla_{\boldsymbol{\theta}}\log\mathcal{P}_{\boldsymbol{\theta}}(\tau)(\mathcal{R}(\tau) - \boldsymbol{b})\right]$$

yielding to a popular algorithm like Gradient Partially Observable Markov Decision Processes (GPOMDP, [6]).
In GPOMDP (see Algorithm 3), the expected return is computed considering the reward at each step. On the contrary, the algorithm REINFORCE [71] (see Algorithm 4), on which GPOMDP is based, considers only the total reward of the trajectory.
In GPOMDP, the expectation is replaced by the empirical over a set of N

Algorithm 4: REINFORCE update rule

Input: Policy parameters $\boldsymbol{\theta}$

Data: Trajectories $\mathcal{T}_{i=1,\dots,N}$

Output: $\nabla^{RF}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

for *each trajectory* i **do**

  //compute cumulative reward

  $R_i^{TOT} = \sum_{t}^{T(\tau_i)} R(a_{\tau_i,t}|s_{\tau_i,t})$

end

for *each time step* t **do**

  for *each parameter* j **do**

    //estimate the baseline

    $$b_j^{RF} = \frac{\sum_{i}^{N}\left(\sum_{t'=0}^{t}\nabla_{\boldsymbol{\theta}_j}\log\pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)^2\gamma^t R_i^{TOT}}{\sum_{i}^{N}\left(\sum_{t'=0}^{t}\nabla_{\boldsymbol{\theta}_j}\log\pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)^2}$$

  end

end

for *each parameter* j **do**

  //estimate the gradient

  $$\nabla^{RF}_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}^{N}\sum_{t}^{T(\tau_i)}\left(\sum_{t'=0}^{t}\nabla_{\boldsymbol{\theta}_j}\log\pi_{\boldsymbol{\theta}}(a_{\tau_i,t'}|s_{\tau_i,t'})\right)\left(\gamma^t R_i^{TOT} - b_j^{RF}\right)$$

end

independent samples and the update rule is formulated as:

$$\nabla_{\boldsymbol{\theta}}^{GP} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i}^{N} \left(\sum_{t}^{T(\tau_i)} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right) \left(\gamma^t R(s_{\tau_i,t}, a_{\tau_i,t}) - \boldsymbol{b}_t^{GP} \right) \right)$$

while in REINFORCE, considering $R^{TOT} = \sum_{t}^{T(\tau)} R(s_t, a_t)$ the cumulative reward at the end of the trajectory, the update rule is formulated as:

$$\nabla_{\boldsymbol{\theta}}^{RF} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i}^{N} \left(\sum_{t}^{T(\tau_i)} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right) \left(\gamma^t R_i^{TOT} - \boldsymbol{b}^{RF} \right) \right)$$

The baseline is used to reduce the variance of the gradient estimate and can be scalar or vectorial. The optimal baseline for parameter j in $\boldsymbol{\theta}$ is the one that minimizes the variance of $\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta})$, i.e. it satisfies the condition:

$$\begin{aligned} \frac{\partial}{\partial b} \mathrm{Var}[\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta})] &= \frac{\partial}{\partial b} \left(\mathop{\mathbb{E}}_{\mathcal{P}_{\boldsymbol{\theta}}(\tau)} \left[(\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}))^2 \right] - \mathop{\mathbb{E}}_{\mathcal{P}_{\boldsymbol{\theta}}(\tau)} \left[\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) \right]^2 \right) \\ &= \frac{\partial}{\partial b} \mathop{\mathbb{E}}_{\mathcal{P}_{\boldsymbol{\theta}}(\tau)} \left[(\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}))^2 \right] = 0 \end{aligned}$$

since the baseline does not affect the expected gradient in the second term. In GPOMDP, to obtain a vectorial (component-wise) baseline for each parameter j and each timestep t, we formulate it as [6]:

$$b_{j,t}^{GP} = \frac{\sum_{i}^{N} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right)^2 \gamma^t R(s_{\tau_i,t}, a_{\tau_i,t})}{\sum_{i}^{N} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right)^2}$$

while in REINFORCE, being the baseline equal for each timestep t of the same trajectory, is formulated as [71]:

$$b_{j}^{RF} = \frac{\sum_{i}^{N} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right)^2 \gamma^t R_i^{TOT}}{\sum_{i}^{N} \left(\sum_{t'=0}^{t} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,t'} | s_{\tau_i,t'}) \right)^2}$$

2.2 Multi-Agent Environments

When the environment involves more than one agent it is called a Multi-Agent environment (MA) and methods fall under the category of Multi-Agent Reinforcement Learning (MARL). The generalization of an MDP to the MA setting is the Stochastic Game (SG, [57]).

2.2.1 Stochastic Games

Definition 2.2. A Stochastic Game is a tuple $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{P}, \boldsymbol{\mathcal{R}}, \gamma, \mathcal{S}_0 \rangle$ where:

- n is the number of agents;
- $\mathcal{S}$ is the state space;
- $\boldsymbol{\mathcal{A}}$ is the joint action space made of the actions spaces $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of all the agents;
- $\mathcal{P} : \mathcal{S} \times \boldsymbol{\mathcal{A}} \to \Delta(\mathcal{S})$ is the transition model, where $\mathcal{P}(s'|s, \boldsymbol{a})$ is the probability that starting from state s and taking joint actions $\boldsymbol{a}$ the resulting state is s';
- $\boldsymbol{\mathcal{R}} : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to \mathbb{R}$ is the joint reward function made of the reward functions $\mathcal{R}_1, \ldots, \mathcal{R}_n$ of all the agents, where $\mathcal{R}_i(s, \boldsymbol{a}, s')$ is the reward obtained by agent i with chosen joint actions $\boldsymbol{a}$ in state s and ending in state s';
- $\gamma \in (0, 1]$ is the discount factor of the reward;
- $\mathcal{S}_0$ is the initial state distribution.

If all the agents have the same reward function then they all maximize the same utility function and the SG is called *fully cooperative*. If there are two agents, $n = 2$, and $\mathcal{R}_1 = -\mathcal{R}_2$, the agents have opposite goals, then the SG is called *zero-sum game*.
A *trajectory* τ is a sequence of state-joint actions-reward $\tau = (s_t, \boldsymbol{a}_t, r_t)_{t=0}^{T(\tau)}$ where $T(\tau)$ represents the horizon of the trajectory τ. We call the set of all trajectories $\mathcal{T} = (\mathcal{S}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})^T$.

2.2.2 Nash Equilibrium

In this document, we focus only on mixed games, in which the rewards of the agents are not always cooperative or always competitive.

The goal of RL algorithms is to find the best equilibrium for the agent and if multiple equilibria exist, the agents need to be able to choose the same one to achieve the highest reward.
One important equilibrium concept is the Nash Equilibrium (NE, [38]).
To present it, we first need to define the concept of a Best Response strategy, which is the optimal strategy given the other agents' strategies. Given a set of n players N, where π_i is the strategy for player $i \in N$ and π_{-i} is the joint strategy of all the other agents, and where for each agent i the expected return is defined as $J_{\pi_i,\pi_{-i}} = \mathbb{E}_{\pi_i,\pi_{-i}}\left[\sum_{t=0}^{T(\tau)} \gamma^t \mathcal{R}(s_{\tau,t}, \boldsymbol{a}_{\tau,t}, s'_{\tau,t})\right]$, the best response strategy $\pi_i^*(\pi_{-i})$ satisfies:

$$\pi_i^*(\pi_{-i}) = \arg\max_{\pi_i} J_{\pi_i,\pi_{-i}}$$

A Nash Equilibrium is the joint strategy in which each strategy $\pi_i^*(\pi_{-i})$ is the best response to the others, i.e. $[\pi_1^*, \ldots, \pi_i^*, \ldots, \pi_n^*]$, so a NE is a *status quo* reached when no player can increase its expected reward by unilaterally changing its strategy. Every static (state-less) game has at least one (possibly infinite) Nash Equilibria. In [37], Nash proved that an equilibrium point always exists in finite games, i.e. games in which the number of players is finite, there is a finite set of pure strategies to choose from and the players use mixed strategies (every player has a probability distribution to choose from the pure strategies).

2.2.3 MARL algorithms

MARL algorithms can be divided into different criteria:

- The degree of awareness of the other agents, which is related to the learning goal: the agents attempt to achieve stability (algorithms *unaware*, or *independent*, of the other agents), they attempt to adapt to the other agents (algorithms *tracking* the other agents) or they attempt to do both (algorithms *aware* of the other agents).
- The homogeneity of the algorithms used by the agents: all the agents must use the same algorithm (homogeneous, e.g. *Nash-Q* [22]) or can use different algorithms (heterogeneous, e.g. *AWESOME* [16], *WoLF-PHC* [9][10]) to reach the goal.
- Knowledge of the task: model-based (e.g. *AWESOME*) or model-free (e.g. *Nash-Q*, *WoLF-PHC*).
- Requirements regarding other agents' observations: the algorithms

might need to know other agents' actions (e.g. *AWESOME*), their actions and rewards (e.g. *Nash-Q*) or neither of them (e.g. *WoLF-PHC*).

Now we present a more detailed categorization of the algorithms based on the degree of awareness.

2.2.3.1 Single-agent RL algorithms

Single-agent RL algorithms can still be used in the multi-agent environment. They consider the other agents as part of the environment which is now non-stationary because of them, losing most of their guarantees of convergence. Despite this, Q-Learning-like algorithms have been widely used in MARL applications [56][34][17][32][33].
In [66] the authors analyzed the dynamic behavior of Boltzmann policies in iterated games, which is the policy used in the second part of this document. They proved that with particular settings, Q-Learning agents can reach a coordinated equilibrium (the same or corresponding Nash Equilibrium strategies) in certain games. In other cases, which we will see later, Gradient-based and Q-Learning-like algorithms show a cyclic behavior.

2.2.3.2 Agent-independent algorithms

Agent-independent algorithms find a strategy independently from the other agents, computing their action-value function and then using algorithms like Nash Q-Learning [23][22] to obtain the expected return of a strategy in the NE and then determine each agent strategy for that equilibrium.
This requires two assumptions: the rewards and the actions of all the agents are known and observable, and all the agents use the same or similar algorithms. Moreover, it requires that every state encountered by all the agents at any point of the learning has a NE in which all the expected returns are maximal, or every state has a NE in which the agent does not increase its expected return by changing its strategy while all the other agents do, i.e. a saddle point.
These requirements are satisfied in a small class of problems and in case the last requirement is missing some methods of equilibrium selection are needed.

2.2.3.3 Agent-tracking algorithms

Agent-tracking algorithms estimate the model, or directly the policy, of the other agents and determine a Best Response to those strategies. This can happen both in static tasks, like in *Fictitious Play* [12], *MetaStrategy* [46] and *Hyper-Q* [63], based on Q-Learning, and in dynamic tasks, like the *Non-Stationary Converging Policies* [69].

2.2.3.4 Agent-aware algorithms

In *agent-aware algorithms*, both convergence and adaptation to the other agents are taken into account.
In static tasks, a model of the environment, typically the reward function, is usually assumed. The *AWESOME* algorithm (Adapt When Everyone is Stationary, Otherwise Move to Equilibrium, [16]) make use of fictitious play when the other agents are stationary but switch to a centrally precomputed NE if it deduces that the other agents are non-stationary. In the Policy Search algorithm category some methods use gradient update rules to guarantee convergence in particular classes of SG: Infinitesimal Gradient Ascent (IGA, [58]), Win-or-Learn-Fast-IGA (WoLF-IGA, [9]), Generalized IGA (GIGA, [73]), and GIGA-WoLF [8].
In IGA-like algorithms, the update function can be generalized with:

$$\theta^i_{k+1} = \theta^i_k + \alpha^i_k \frac{\partial \mathbb{E}\left[r^i|\boldsymbol{\theta}_k\right]}{\partial \theta^i} \tag{2.8}$$

where θ^i are the policy parameters of agent i, $\boldsymbol{\theta}$ are the joint policy parameters of all the agents, real or estimated, k is the learning step and α is the constant gradient step. The expected return converges to a NE with infinitesimal step size, i.e. $\alpha \to 0$.
In dynamic tasks, Win-or-Learn-Fast Policy Hill-Climbing (WoLF-PHC, [9][10]) updates an action-value function similar to Q-Learning with a WoLF-inspired rule, in which α is higher if losing, so to quickly change policy, and lower if winning, so to converge towards the optimal.

2.3 Algorithms with Future Policy Prediction

In this document, other than the standard gradient ascent algorithm GPOMDP, we consider another class of algorithms, the *agent-aware algorithms with future policy prediction*. These algorithms take into account in their update

function knowledge about the other agents' policy updates, but they differ on the policy predicted and in the guarantees they provide.

2.3.1 IGA-PP and LOLA

The main algorithms we consider are Iterated Gradient Ascent Policy Prediction (IGA-PP, [72]) and Learning with Opponent Learning Awareness (LOLA, [19]). They both take into account, in their update function, knowledge about the other agents' policy updates, but they differ on the policy predicted and in the guarantees they provide.
IGA-PP assumes to know its opponent's current strategy and its change direction and, with this knowledge, IGA-PP can predict the updated strategy of the other agent and play a Best Response. IGA-PP is shown to converge to a Nash Equilibrium if played in self-play and if the learning rate is sufficiently small.
LOLA, on the other hand, try to directly influence the other agent, shaping its learning direction. This is done by taking into account the impact of the update of the LOLA agent into the shaping of the other agent's learning direction. LOLA assumes that the other agent uses an update rule similar to (2.6).
If the policies are parameterized, given n agents, the expected return J^x of agent x depends on the set of the agents' parameters and this dependency is expressed with $J^x(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2)$.
Given $\boldsymbol{\theta}^x_k$ the policy parameters of agent x at time step k, $\nabla_{\boldsymbol{\theta}^y} J^x(\boldsymbol{\theta}^1_k, \boldsymbol{\theta}^2_k)$ the gradient of agent x over the parameters of agent y and $\nabla_{\boldsymbol{\theta}^1,\boldsymbol{\theta}^2} J^x(\boldsymbol{\theta}^1_k, \boldsymbol{\theta}^2_k)$ the Hessian of agent x over the parameters of both the opponent agent and its own, the update rule for IGA-PP is formulated as:

$$\boldsymbol{\theta}^1_{k+1} = \boldsymbol{\theta}^1_k + \alpha^1 \nabla_{\boldsymbol{\theta}^1}(J^1\left(\boldsymbol{\theta}^1_k, \boldsymbol{\theta}^2_k + \alpha^2 \nabla_{\boldsymbol{\theta}^2} J^2(\boldsymbol{\theta}^1_k, \boldsymbol{\theta}^2_k)\right))$$

Writing $\nabla_x J^y = \nabla_{\boldsymbol{\theta}^x} J^y(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2)$ and $\nabla_{xy} J^z = \nabla_{\boldsymbol{\theta}^x,\boldsymbol{\theta}^y} J^z(\boldsymbol{\theta}^1, \boldsymbol{\theta}^2)$ for any x, y, z, we can rewrite it as:

$$\boldsymbol{\theta}^1_{k+1} = \boldsymbol{\theta}^1_k + \alpha^1(\nabla_1 J^1 + \alpha^2(\nabla_{12} J^1)^T \nabla_2 J^2) \tag{2.9}$$

Instead, the update rule for LOLA is formulated as:

$$\boldsymbol{\theta}^1_{k+1} = \boldsymbol{\theta}^1_k + \alpha^1(\nabla_1 J^1 + \alpha^2 \left((\nabla_2 J^1)^T \nabla_{12} J^2\right)^T) \tag{2.10}$$

where, with the last term $\left((\nabla_2 J^1)^T \nabla_{12} J^2\right)$, the agent actively shapes the opponent's learning process.

2.3.2 NOHD, SOS, SGA, CO, CGD

Moreover, we take into consideration other gradient-based algorithms, with prediction of the opponent's policy, to show how our method can identify the correct algorithm among them. They are Newton Optimization on Helmholtz Decomposition (NOHD, [49]), Stable Opponent Shaping (SOS, [30]), Symplectic Gradient Adjustment (SGA, [5]), Consensus Optimization (CO, [35]), and Competitive Gradient Descent (CGD, [55]).

We start by presenting, as defined in those works, some concept used by their update rules:

- the simultaneous gradient, i.e. the concatenation of the gradients of the agents, is:
$$\xi = (\nabla_1 J^1, \nabla_2 J^2)$$
- the Jacobian, i.e. the derivative of the simultaneous gradients, is:
$$\mathcal{J} = \nabla\xi = \begin{pmatrix} \nabla_{11} J^1 & \nabla_{12} J^1 \\ \nabla_{21} J^2 & \nabla_{22} J^2 \end{pmatrix}$$
- $\mathcal{J}_o$ is the matrix of the anti-diagonal blocks of $\mathcal{J}$;
- $\boldsymbol{\alpha} = (\alpha^1, \alpha^2)$ is the concatenation of the values of the learning rates of the agents;
- $\text{diag} : \mathbb{R}^{d \times n} \to \mathbb{R}^d$, where d is the number of parameters representing the policy, is the operator that creates a vector made by the values in the diagonal of a matrix;
- I is the identity matrix;
- $S = \frac{1}{2}(\mathcal{J} + \mathcal{J}^T)$ and $A = \frac{1}{2}(\mathcal{J} - \mathcal{J}^T)$ are respectively the Potential and Hamiltonian part of a game;
- $\nabla\mathcal{H} = (S + A)^T \xi$ is the gradient of the Hamiltonian function;
- S_u^{-1} is the Positive Truncated Inverse (PT-Inverse) of matrix S, in which negative eigenvalues' signs are flipped and small eigenvalues are replaced by u.

In *NOHD*, the authors perform the update differently depending if the method identifies the game as Potential or Hamiltonian. If the game is Po-

Algorithm 5: NOHD update rule selection

```
if cos ν_S ≥ 0 then
    if cos ν_S ≥ cos ν_A then
        use Potential update rule
    else
        use Hamiltonian update rule
else
    if cos ν_S ≤ cos ν_A then
        use Potential update rule
    else
        use Hamiltonian update rule
```

tential the update rule is $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\alpha}(S_u^{-1}\xi)$, if instead is Hamiltonian the update rule is $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\alpha}(A^{-1}\xi)$. To identify if the game is Potential or Hamiltonian, the cosine between $\nabla\mathcal{H}$ and the directions of the two candidates update is computed as:

$$\cos\nu_S = \frac{(S_u^{-1}\xi)^T\nabla\mathcal{H}}{\left\|S_u^{-1}\xi\right\|\|\nabla\mathcal{H}\|}, \quad \cos\nu_A = \frac{(A^{-1}\xi)^T\nabla\mathcal{H}}{\|A^{-1}\xi\|\|\nabla\mathcal{H}\|}$$

The identification algorithm can be expressed with Algorithm 5.
NOHD converges to a local Nash Equilibrium in general games if: $\alpha > 0$ and sufficiently small; the loss function is twice differentiable; ξ is Lipschitz; A is invertible and $\mathcal{J}, S, A, S^{-1}, A^{-1}$ are bounded and Lipschitz continuous in the surroundings of a fixed point.

In *SOS*, first, $\xi_0 = (I - \alpha\mathcal{J}_o)\xi$ and $\mathcal{X} = \text{diag}\big(\mathcal{J}_o^T(\nabla_1 J^1, \nabla_2 J^2)\big)$ are computed. Then, given the hyperparameter $0 < a, b < 1$, the two-part criterion p controlling the update is calculated with Algorithm 6 and the update rule is given by $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\alpha}(\xi_0 - p\boldsymbol{\alpha}\mathcal{X})$.
SOS converges to a local Nash Equilibrium in all differentiable games if: $\alpha > 0$ and sufficiently small; the loss function is thrice differentiable; $\mathcal{J} \succeq 0$ is invertible with symmetrical diagonal blocks.

In *SGA*, the update rule is given by $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\alpha}(\xi + \chi A^T\xi)$, with $\chi = \text{sign}\big(\langle\xi, \nabla\mathcal{H}\rangle\langle A^T\xi, \nabla\mathcal{H}\rangle + \epsilon\big)$, where sign is the sign function and ϵ is a small bias that directs the update towards stable fixed points. SGA converges to a Nash Equilibrium in general games if $\alpha > 0$ and sufficiently small.

Algorithm 6: SOS criterion parameter

if $\langle -\boldsymbol{\alpha}\mathcal{X}, \xi_0 \rangle > 0$ **then**
  $p_1 = 1$
else
  $p_1 = \min\left\{1, \frac{-\alpha\|\xi_0\|^2}{\langle -\boldsymbol{\alpha}\mathcal{X}, \xi_0 \rangle}\right\}$
if $\|\xi\| < b$ **then**
  $p_2 = \|\xi\|^2$
else
  $p_2 = 1$
$p = \min\{p_1, p_2\}$

In *CO*, the update rule is formulated as $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{\alpha}\big(\xi + h(\mathcal{J}^T\xi)\big)$, where h is the regularization parameter. CO converges to a Nash Equilibrium in two-players zero-sum games if: $\alpha > 0$ and sufficiently small; $I - h\mathcal{J}T$ is invertible; $\mathcal{J}$ at the local Nash Equilibrium is negative, semi-definite and invertible.

In *CGD*, the update rule for agent x, where y is the other agent, is given by $\boldsymbol{\theta}^x_{k+1} = \boldsymbol{\theta}^x_k - \alpha^x(I - (\alpha^x)^2\nabla_{xy}J^x\nabla_{yx}J^y)^{-1}(\nabla_x J^x - \alpha^x\nabla_{xy}J^x\nabla_y J^y)$. CGD converges to a Nash Equilibrium in two-players zero-sum games, and in games dominated by competition according to the authors' expectations, if: $\alpha > 0$ and sufficiently small; the loss function is twice continuous differentiable with L-Lipschitz continuous mixed Hessian and convex-concave or $\nabla_{xx}J^x, \nabla_{yy}J^x$ are L-Lipschitz continuous and the Hessian's diagonal blocks satisfy $\alpha\|\nabla_{xx}J^x\|, \alpha\|\nabla_{yy}J^x\| \leq 1$;

For more details about these algorithms we refer to the original documents.

2.4 Imitation Learning

To compute these updates, the learning agent has to know the policies and the reward functions of the other agents, but this is not always the case.
To overcome this obstacle, we present an approach based on Imitation Learning in which the information about an agent is learned by observing its interaction with the environment and recovering from that the policy used, the reward function maximized or both.
Imitation Learning can be divided into two main approaches: Behavioral Cloning (BC, [3]) and Inverse Reinforcement Learning (IRL, [39]).

The agent whose actions are observed is called expert or demonstrator.
BC recovers the policy from its trajectories in a supervised learning fashion, while IRL recovers the reward function it maximizes. BC recovers the policy used but it does not say anything about the reason for that policy (its goal) which is instead recovered by IRL in the reward function.
On the other hand, knowing the reward with IRL is more informative than knowing the policy with BC since the latter can be reconstructed from the reward function, even after changes in the transition function of the environment.

2.4.1 Behavioral Cloning

Behavioral Cloning is used to recover a policy that can be mapped to the demonstrated one. If the policy to recover is deterministic and the action space is finite classification techniques like Gaussian Mixture Models (GMM) [15], Decision Trees [53], Bayesian Networks [24] and k-Nearest Neighbors [54] are employed. Instead, if the action space is continuous regression techniques like Lazy Learning [4] and Neural Networks (NN, [45]) are adopted. If the policy to recover is stochastic, a set of policy parameters can be estimated to satisfy the demonstration and it can be seen as a probability estimation problem.

In this document, we perform BC following a maximum-likelihood estimation approach as presented in [43]. Given probability distributions P_{θ_1} and P_{θ_2}, respectively dependent on parameters θ_1 and θ_2 in parameter space Θ, if $P_{\theta_1} = P_{\theta_2} \Rightarrow \theta_1 = \theta_2 \quad \forall \theta_1, \theta_2 \in \Theta$, then the model satisfy the identifiability property. Assuming differentiable policies belonging to a parameterized policy space, mild regularity conditions and the identifiability property [14], the optimization problem is formulated as:

$$\widehat{\boldsymbol{\theta}} = \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{\tau}^{N} \sum_{t}^{T(\tau)} log \pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t}) \tag{2.11}$$

To solve this maximization, we arbitrarily initialize $\widehat{\boldsymbol{\theta}}$, then we iteratively compute the gradients between the trajectories demonstrated and the policy parameters $\widehat{\boldsymbol{\theta}}$ and use this gradient to update the policy parameters. We iteratively repeat this process till convergence or till the iteration limit is reached (see Algorithm 7).

Algorithm 7: BC for estimation of $\boldsymbol{\theta}$

Input: BC's learning factor η, (optional) number of iterations N_i
Data: N trajectories $\mathcal{T}$ with states and actions (no rewards)
Output: Policy parameters $\widehat{\boldsymbol{\theta}}$
Initialize random $\widehat{\boldsymbol{\theta}}$
while *iteration* $\leq N_i$ **and** not converged **do**
  Set $update = \mathbf{0}$
  for *each trajectory* τ **do**
    for *each trajectory step* t **do**
      $update \mathrel{+}= \log \pi_{\widehat{\boldsymbol{\theta}}}(a_{\tau,t}|s_{\tau,t})$
    end
  end
  $\widehat{\boldsymbol{\theta}} \mathrel{+}= \eta \cdot update$
end

2.4.2 Inverse Reinforcement Learning

To present Inverse Reinforcement Learning, we take advantage of the concept of Feature Expectation [1]. The idea is that the rewards can be expressed as feature vectors $\boldsymbol{\phi}$, linearly combined with the policy parameters. Given a trajectory τ generated by following a policy π, we can compute the feature expectations as the cumulative discounted feature vector $\boldsymbol{\psi}_\pi$ with respect to the distribution of the trajectory.

$$\boldsymbol{\psi}_\pi = \mathop{\mathbb{E}}_{\substack{s_t\sim\tau \\ a_t\sim\tau}} \left[\sum_{t=0}^{T(\tau)} \gamma^t \boldsymbol{\phi}(s_{\tau,t}, a_{\tau,t}) \right]$$

Hence we can rewrite the expected return in terms of the feature expectations:

$$J_\pi = \mathop{\mathbb{E}}_{\substack{s_t\sim\tau \\ a_t\sim\tau}} \left[\sum_{t=0}^{T(\tau)} \gamma^t R(s_{\tau,t}, a_{\tau,t}, s'_{\tau,t}) \right] = \mathop{\mathbb{E}}_{\substack{s_t\sim\tau \\ a_t\sim\tau}} \left[\sum_{t=0}^{T(\tau)} \gamma^t \phi_\pi(s_{\tau,t}, a_{\tau,t})^T \boldsymbol{\omega} \right] = \boldsymbol{\psi}_\pi^T \boldsymbol{\omega}$$

Where $\boldsymbol{\omega}$ is the weight vector of the rewards.
Considering π^E the policy of the expert (the policy imitated) and its reward function $\mathcal{R}^E$, $\pi^{\boldsymbol{\omega}}$ the policy induced by the reward function $\boldsymbol{\phi}^T\boldsymbol{\omega}$ where $\boldsymbol{\phi}$ is the feature vector of reward vector $\boldsymbol{\omega}$, our goal is to find a vector $\boldsymbol{\omega}$ that produces a policy as close as possible to the policy demonstrated. This can be achieved by minimizing the difference (e.g. Euclidean distance) between

$\mathcal{R}^E$ and $\boldsymbol{\phi}^T\boldsymbol{\omega}$.

$$\widehat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} \left\|\mathcal{R}^E - \boldsymbol{\phi}^T\boldsymbol{\omega}\right\|_2^2$$

where $\mathcal{R}^E$ can be estimated as $\widehat{\mathcal{R}}^E = \frac{1}{N}\sum_{\tau=1}^{N}\sum_{t=0}^{T(\tau)}\boldsymbol{\phi}(s_{\tau,t}, a_{\tau,t})$ [1].

2.4.3 Policy Gradient-based IRL

Policy Gradient methods that perform IRL use the minimization of the policy gradient to retrieve the reward function. We focus on Gradient Inverse Reinforcement Learning (GIRL, [44]), since it is a batch model-free approach [50], but policy gradients have been notably used also in [62] and [21].
Given the expert policy $\pi_{\boldsymbol{\theta}}^E$ over the vector parameter $\boldsymbol{\theta}$, we can formulate the policy gradient as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\omega}) = \mathop{\mathbb{E}}_{\substack{s_t\sim\tau \\ a_t\sim\tau}}\left[\sum_{t}^{T(\tau)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}^E(a_{\tau,t}|s_{\tau,t})\gamma^t R_{\boldsymbol{\omega}}(s_{\tau,t}, a_{\tau,t})\right]$$

The expert policy $\pi_{\boldsymbol{\omega}}^E$ is a stationary point of $J(\boldsymbol{\theta}^E, \boldsymbol{\omega}^E)$ because of the reward vector parameter $\boldsymbol{\omega}^E$. To recover the estimated reward vector $\widehat{\boldsymbol{\omega}}$, the authors proposed a minimization of the norm with respect to the gradient. This optimization is possible because the objective function is convex as long as the reward function is convex with respect to $\boldsymbol{\omega}$. Moreover, if the reward parameterization is linear, the problem is ill-posed and has a trivial solution in zero. This is solved restricting the expert's vector $\boldsymbol{\omega}^E$ into the unit $(q-1)$-simplex $\Delta^{q-1} = \{\boldsymbol{\omega} \in \mathbb{R}^q : \|\boldsymbol{\omega}\|_1 = 1 \wedge \boldsymbol{\omega} \succeq 0\}$ where $\succeq$ is the component-wise inequality and $q > 1$ is the dimension of the reward vector. Under these conditions, the optimization problem can be formulated as:

$$\widehat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} \frac{1}{y}\left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^E, \boldsymbol{\omega})\right\|_x^y$$

where $x, y \geq 1$.
This means that the minimum norm gradient $\widehat{\boldsymbol{\omega}}$ is the reward that minimizes the improvement in the expected return depending on the policy parameters.

2.4.4 IRL from a Learning agent

Till now we have always performed IRL on expert agents but, in multi-agent environments, it is important to learn the reward even if the other agent is still learning to be able to decide if cooperating or competing with them.

In [26] the authors showed how it is possible to learn the reward function not only from an optimal demonstrator but also from a sub-optimal one which is learning to perform the task itself. The learning agent is called the *learner* while the agent who is observing the learner and recovering the reward function is the *observer*. They assume that the learner's policy is strictly improving over time, i.e. the expected return following the policy at time $k+1$ is greater than at time k. To do so they assume the learning to be under the framework of entropy-regularized RL.
However, this is not always the case. Both when the learning process is performed by humans and when is performed by RL algorithms, the learning process does not satisfy the property of monotonic improvements [60].
To overcome this assumption, in [48], the authors propose an algorithm called Learning Observing a Gradient not-Expert Learner (LOGEL). The authors assume that the learner's algorithm is gradient-based, following the direction of its expected discounted return, since many successful RL algorithms are in this category [41]. The optimization problem determines the reward function that minimizes the distance between the given policy parameters and the parameters obtained following policy gradient ascent from the beginning of the demonstration using that reward function.
Being $\Delta_k = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$, the minimization problem at learning step $k+1$ is defined as:

$$\widehat{\boldsymbol{\omega}} = \min_{\boldsymbol{\omega}} \sum_{k'=1}^{k} \left\| \Delta_{k'} - \alpha \nabla_{\boldsymbol{\theta}_{k'}} \boldsymbol{\psi}_{k'}^T \boldsymbol{\omega} \right\|_2^2 \tag{2.12}$$

Moreover, in realistic scenarios, the observer has access only to the trajectories demonstrated and not to information regarding the learner, like the learning rate or the exact policy parameters. In this setting, the observer has to estimate the policy parameters $\Theta = \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{k+1}$, the learning rates $\boldsymbol{\alpha} = \alpha_1, \ldots, \alpha_k$ and the reward weight $\boldsymbol{\omega}$.
Under the assumption of gradient-based updates and given $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_k^T \boldsymbol{\omega}$, this problem is a maximization of the log-likelihood of $p(\boldsymbol{\theta}_1, \boldsymbol{\alpha}, \boldsymbol{\omega} | \mathcal{T})$:

$$\max_{\boldsymbol{\theta}_1, \boldsymbol{\alpha}, \boldsymbol{\omega}} \sum_{(s,a) \in \tau_1} log \pi_{\boldsymbol{\theta}_1}(a|s) + \sum_{k'=2}^{k+1} \sum_{(s,a) \in \tau_{k'}} log \pi_{\boldsymbol{\theta}_{k'}}(a|s)$$

To solve this problem, gradients up to the k-th order need to be computed, thus this is not practical. To solve it, the problem is broken into two parts, first the estimation of the policy parameters Θ, then the estimation of the

learning rates $\boldsymbol{\alpha}$ and reward weights $\boldsymbol{\omega}$.

To recover the policy parameters, they used BC as a maximum-likelihood estimation problem as shown in Equation (2.11) and in Algorithm 7.
Now, given the estimated policy parameters $(\overline{\boldsymbol{\theta}}_1, \ldots, \overline{\boldsymbol{\theta}}_{k+1})$, if the learning rate of the learner is constant, Equation (2.12) can be directly used, otherwise we need to estimate $\boldsymbol{\alpha}$ first. In this case, given $\overline{\Delta}_k = \overline{\boldsymbol{\theta}}_{k+1} - \overline{\boldsymbol{\theta}}_k$ and $\alpha_k \geq \epsilon$ where ϵ is a small constant, the optimization problem becomes:

$$\min_{\boldsymbol{\omega},\boldsymbol{\alpha}} \sum_{k=1}^{m} \left\| \overline{\Delta}_k - \alpha_k \overline{\nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_k} \boldsymbol{\omega} \right\|_2^2$$

To optimize this function the optimization of $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are alternated using alternate block-coordinate descent [64].

2.5 Likelihood

In statistics, the likelihood is a function representing how well a statistical model fits a sample of data, given the values of the unknown parameters. In other words, the likelihood function associates the probability of collecting the sample provided for each set of parameters. For this reason, the likelihood is not a probability but rather a value proportional to a probability.

2.5.1 Discrete probability distribution

Given a sample of observations $x \in X$, a probability mass function $p(x)$ and its unknown parameters θ, we call $\mathscr{L}(\theta|x) = p(x, \theta)$ the *likelihood function* that states how likely each set of parameters θ is, given the outcome sample x. If X is a set of independent and identically distributed (i.i.d.) random variables, the likelihood function is equal to the product of the probabilities of the sample:

$$\mathscr{L}(\theta|x) = \prod_{i=1}^{n} p(x_i, \theta)$$

2.5.2 Continuous probability distribution

Given a random variable X, an absolutely continuous probability distribution with density function $f(x)$ and its unknown parameters θ, we call $\mathscr{L}(\theta|x) = f(x, \theta)$ the *likelihood function* that states how likely parameters θ are, given the outcome x of X. If the n outcomes $x = x_0, \ldots, x_i, \ldots, x_n$

of the random variable X are i.i.d., the likelihood function is equal to the product of the marginal densities of the outcomes:

$$\mathscr{L}(\theta|x) = \prod_{i=1}^{n} f(x_i, \theta)$$

2.5.3 Log-likelihood

Instead of the likelihood, the log-likelihood is often taken. The reason lies in the unstable nature of products which tend to converge quickly to zero or infinity. Since the likelihood is numerically solved on computers with limited precision, this fast convergence leads to results not distinguishable from zero or infinity. To solve this problem, the log-likelihood is usually computed. This is due to the logarithm property that the logarithm of a product of values is equivalent to the sum of the logarithms of those values. Having a sum instead of a product greatly alleviates the convergence problem. Moreover, the concavity of the function is a key property in the maximization and the most common probability distribution family, the exponential family, is only logarithmically concave.
The log-likelihood is often formulated as:

$$\mathcal{L} = \ell(\theta|x) = \log(\mathscr{L}(\theta|x)) \tag{2.13}$$

2.5.4 Likelihood and Log-Likelihood Ratio

The *law of likelihood*, based on the *likelihood principle*, states that the support of the evidence of one set of parameters θ_1 against another set θ_2 can be determined as the ratio of their two likelihoods and it is called *likelihood ratio*:

$$l = \frac{\mathscr{L}(\theta_1|x)}{\mathscr{L}(\theta_2|x)}$$

If the likelihood ratio l is greater than 1, then the evidence supports θ_1 against θ_2; if it is lower than 1, then vice-versa; if it is equal to 1, then the evidence is indifferent to the two sets of parameters.
The parameters that maximize the likelihood function are the values that the evidence supports the most. This fact is the basis of the method of Maximum Likelihood Estimation (MLE).

As we saw for the likelihood function, also the likelihood ratio is more often expressed as the *log-likelihood ratio*, since maximizing the likelihood is equivalent to maximizing the log-likelihood due to the strictly increasing

property of the logarithms.
Given a i.i.d. sample and the properties of the logarithms, the log-likelihood ratio can be expressed as the difference of the log-likelihoods:

$$\lambda = \log\left(\frac{\mathscr{L}(\theta_1|x)}{\mathscr{L}(\theta_2|x)}\right) = \log(\mathscr{L}(\theta_1|x)) - \log(\mathscr{L}(\theta_2|x)) = \ell(\theta_1|x) - \ell(\theta_2|x)$$

2.5.5 Likelihood and Log-Likelihood Ratio Test

The likelihood-ratio test states the relation between the goodness of fit of two statistical models, based on their likelihood-ratio. In the ratio, the numerator is the statistical model in which the maximization over the parameter space is bounded by some constraints while the denominator is the statistical model whose maximization over the parameter space is unbounded.
Formally, given a parameter space Θ, a subset $\Theta_0 \subset \Theta$ due to some constraints, the null hypothesis $H_0 : \theta \in \Theta_0$ and being sup the supremum function, the likelihood-ratio test statistic is formulated as:

$$\Lambda = -2\log\left[\frac{\sup_{\theta\in\Theta_0}\mathscr{L}(\theta)}{\sup_{\theta\in\Theta}\mathscr{L}(\theta)}\right]$$

The test is carried out looking at the result of the ratio. If the value Λ is lower than a predefined value ζ then it is said that the null hypothesis H_0, the constrained model, is supported by the sample and the null hypothesis can not be rejected. Otherwise, if it is greater than ζ the null hypothesis is rejected. If $\Lambda = \zeta$ the null hypothesis is rejected with probability q. Usually, ζ and q are chosen to have a significance level of α.

As seen above, when performing operations with likelihoods, usually they are expressed in terms of log-likelihoods, making use of the properties of logarithms. Then, the log-likelihood ratio test is formulated as:

$$\Lambda = -2\big[\ell(\theta_0) - \ell(\widehat{\theta})\big]$$

where $\theta_0 \in \Theta_0$, $\widehat{\theta} \in \Theta$, $\ell(\widehat{\theta})$ is defined as $\log\left[\sup_{\theta\in\Theta}\mathscr{L}(\theta)\right]$ and $\ell(\theta_0)$ is the maximal value in the case in which the null hypothesis is true.

2.5.6 Simple-vs-simple hypothesis test

In the simple-vs-simple hypothesis test, the null and alternative hypothesis are both specified and the distribution of the data is completely known,

there are no parameters to estimate. The two hypotheses are formulated in terms of the fixed parameter θ as:

$$H_0 : \theta = \theta_0$$
$$H_1 : \theta = \theta_1$$

Then, given the i.i.d. sample x, the likelihood-ratio test is expressed as:

$$\Lambda = \frac{\mathscr{L}(\theta_0|x)}{\mathscr{L}(\theta_1|x)}$$

If $\Lambda > \zeta$ the null hypothesis H_0 can not be rejected. Otherwise, if $\Lambda < \zeta$ the null hypothesis H_0 is rejected. If $\Lambda = \zeta$ the null hypothesis is rejected with probability q.

As before, this test can be performed also in logarithmic terms and formulated as:

$$\begin{aligned} \Lambda &= -2\log\left(\frac{\mathscr{L}(\theta_0|x)}{\mathscr{L}(\theta_1|x)}\right) \\ &= -2(\log(\mathscr{L}(\theta_0|x)) - \log(\mathscr{L}(\theta_1|x))) \\ &= -2(\ell(\theta_0|x) - \ell(\theta_1|x)) \end{aligned} \tag{2.14}$$

where multiplying by -2 ensures that, if the null hypothesis H_0 is true, Λ converges asymptotically to being χ^2-distributed thanks to Wilks' theorem [70].
In this case, the rejection condition is the opposite: If $\Lambda < \zeta$ the null hypothesis H_0 can not be rejected. Otherwise, if $\Lambda > \zeta$ the null hypothesis H_0 is rejected. As above, if $\Lambda = \zeta$ the null hypothesis is rejected with probability q.

2.6 Importance Sampling

Importance Sampling (IS, [27]) is the technique used to estimate the expected values of a distribution given only the sample from a different one. The main idea is to sample from a different distribution to lower the variance of the estimation needed or when sampling from the wanted distribution is difficult or impossible.
For the purpose of this document, we present importance sampling applied to a simulation when sampling is impossible.
In a SG, given $(\,\cdot\,)_{0:t}$ the history of values $(\,\cdot\,)$ from time step 0 to time

step t, given a trajectory $\mathcal{T}_{0:t}$, starting from state s_0 and continuing with joint actions $\boldsymbol{a}_{0:t-1}$ under the joint policy $\boldsymbol{\pi}$, and the transition probability function $\mathcal{P}$, the probability of this trajectory is:

$$\begin{aligned} &Pr(\boldsymbol{a}_0, s_1, \boldsymbol{a}_1, \ldots, s_t | s_0, \boldsymbol{a}_{0:t-1}, \boldsymbol{\pi}) = \\ &= \pi(\boldsymbol{a}_0|s_0)\mathcal{P}(s_1|s_0, \boldsymbol{a}_0)\pi(\boldsymbol{a}_1|s_1)\cdots\mathcal{P}(s_t|s_{t-1}, \boldsymbol{a}_{t-1}) = \\ &= \prod_{t'=0}^{t-1} \pi(\boldsymbol{a}_{t'}|s_{t'})\mathcal{P}(s_{t'+1}|s_{t'}, \boldsymbol{a}_{t'}) \end{aligned}$$

Thus, given two joint policies, $\boldsymbol{\pi}_0$ following θ_0 and $\boldsymbol{\pi}_1$ following θ_1, the relative probability of trajectory $\mathcal{T}_{0:t}$ under joint policy $\boldsymbol{\pi}_0$ with respect to joint policy $\boldsymbol{\pi}_1$ (the IS ratio) is:

$$\rho_{0:t-1} = \frac{\prod_{t'=0}^{t-1} \pi_0(\boldsymbol{a}_{t'}|s_{t'})\mathcal{P}(s_{t'+1}|s_{t'}, \boldsymbol{a}_{t'})}{\prod_{t'=0}^{t-1} \pi_1(\boldsymbol{a}_{t'}|s_{t'})\mathcal{P}(s_{t'+1}|s_{t'}, \boldsymbol{a}_{t'})} = \prod_{t'=0}^{t-1} \frac{\pi_0(\boldsymbol{a}_{t'}|s_{t'})}{\pi_1(\boldsymbol{a}_{t'}|s_{t'})}$$

The ratio $\rho_{0:t-1}$ represent the adjustment needed to correctly estimate the expected value of a distribution given samples of another one.
Given n trajectories $\mathcal{T}_{1,0:t}, \ldots, \mathcal{T}_{\tau,0:t}, \ldots, \mathcal{T}_{n,0:t}$, the IS estimator is the average of the single IS ratios $\rho^{\tau}_{0:t-1}$:

$$\rho^{IS}_{0:t-1} = \frac{1}{n}\sum_{\tau=1}^{n} \rho^{\tau}_{0:t-1} = \frac{1}{n}\sum_{\tau=1}^{n}\prod_{t'=0}^{t-1} \frac{\pi_0(\boldsymbol{a}_{\tau,t'}|s_{\tau,t'})}{\pi_1(\boldsymbol{a}_{\tau,t'}|s_{\tau,t'})}$$

2.6.1 Multiple Importance Sampling

In the case in which we have a sample based on more than one distribution, we can rely on *Multiple Importance Sampling* (MIS, [67]). Given the distribution without samples $f_0(\mathcal{T})$ and n distributions $f_1(\mathcal{T}), \ldots, f_j(\mathcal{T}), \ldots, f_n(\mathcal{T})$ each with m_j trajectories $\mathcal{T}_{j,\tau,0:t}$, the MIS ratio is the weighted combination of the IS estimators of each distribution, weighted for the combining estimator $w_j(\mathcal{T}_{j,\tau})$, dependent on the sample $\mathcal{T}_{j,\tau}$.

$$\begin{aligned} \rho^{MIS}_{0:t-1} &= \sum_{j=1}^{n} \frac{1}{m_j}\sum_{\tau=1}^{m_j} w_j(\mathcal{T}_{j,\tau})\frac{f_0(\mathcal{T})}{f_j(\mathcal{T})} \\ &= \sum_{j=1}^{n} \frac{1}{m_j}\sum_{\tau=1}^{m_j} w_j(\mathcal{T}_{j,\tau})\prod_{t'=0}^{t-1} \frac{\pi_0(\boldsymbol{a}_{j,\tau,t'}|s_{j,\tau,t'})}{\pi_j(\boldsymbol{a}_{j,\tau,t'}|s_{j,\tau,t'})} \end{aligned}$$

where $\forall\ \mathcal{T} : \sum_{j=1}^{n} w_j(\mathcal{T}) = 1$.
$w_j(\mathcal{T})$ can be an arithmetic average, hence simply $w_j(\mathcal{T}) = \frac{1}{n}$, or a more thoughtful balance heuristic:

$$w_j(\mathcal{T}) = \frac{m_j f_j(\mathcal{T})}{\sum_{k=1}^{n} m_k f_k(\mathcal{T})}$$

Using the balance estimator, we can rewrite the MIS ratio as:

$$\rho_{0:t-1}^{MIS} = \sum_{j=1}^{n} \sum_{\tau=1}^{m_j} \frac{\prod_{t'=0}^{t-1} \pi_0(\boldsymbol{a}_{j,\tau,t'} | s_{j,\tau,t'})}{\sum_{k=1}^{n} m_k \prod_{t'=0}^{t-1} \pi_k(\boldsymbol{a}_{k,\tau,t'} | s_{k,\tau,t'})} \tag{2.15}$$

2.7 Renyi divergence

The Renyi divergence [51] is a function that measures how a probability distribution differs from a second one. As with all divergences, the Renyi divergence is a weaker notion of distance, since it need not be symmetric (i.e. the divergence from P to Q does not need to be equal to the divergence from Q to P) and need not satisfy the triangle inequality.
Given Z the space of all probability distributions and P, Q two probability distributions, a divergence is formulated as a function $D(\cdot \,\|\, \cdot) : Z \times Z \to \mathbb{R}$ where $D(P\|Q) = 0 \iff P = Q$ and $D(P\|Q) \geq 0 \quad \forall P, Q \in Z$.
In the case of the Renyi divergence of order β, given X the discrete probability space and r its probability function, the divergence of P from Q is defined as:

$$D_\beta(P\|Q) = \frac{1}{\beta - 1} \log \left(\sum_{i=1}^{n} \frac{p_i^\beta}{q_i^{\beta-1}} \right)$$

with $0 < \beta < \infty$ and $\beta \neq 1$.
The Renyi divergence is also the generalization of the Kullback-Leibler divergence.

2.7.1 Kullback-Leibler divergence

Taking the limit $\beta \to 1$, the Renyi divergence gives the Kullback-Leibler divergence [29] for discrete probability distributions defined as:

$$D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{2.16}$$

For distributions of a continuous random variable, the Kullback-Leibler divergence is formulated as:

$$D_{KL}(P\|Q) = \int_{-\infty}^{+\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{2.17}$$

Despite not being symmetrical, a version satisfying that property can be formulated as:

$$D_{KL}^{SYM}(P, Q) = \frac{D_{KL}(P\|Q) + D_{KL}(Q\|P)}{2}$$

2.8 Opponent identification: related works

As defined in [2], *Opponent Modeling* is the process of constructing the model of another agent, defined as a function that takes the interactions observed as input and returns a prediction of some property of the modeled agent. Most of the literature investigates how to model the policies, the parameters or the reward functions of other agents. In type-based reasoning methods [2], the modeling agent tries to identify the *type* of the modeled agent that best matches one of several known types. To perform this identification, it updates its belief about the other agent's type according to the prediction accuracy of the actions observed for the different types. These methods differ from our approach since they consider as types already expert agents or agents with already known strategies and their adaptation properties to other agents. Instead, in our method, we consider learning agents which do not have any knowledge of the environment or strategies at the beginning of the learning, so they do not have a predefined *type* as intended in these works. In recursive reasoning [2][20], the agent's belief is nested, i.e. when the agent models the knowledge of the other agent, it includes what itself knows about the other agent, adding a level of complexity to the modeling of the reasoning of the other agent. This process can be repeated infinitely, leading to recursive modeling. Recursive reasoning methods approximate

the belief up to a fixed level of recursion. These methods differ from our approach since they assume rational agents, i.e. agents that always choose the optimal action according to their knowledge, which is an assumption we do not make. Moreover, recursive reasoning stops to a fixed level of recursion k, while our method could be seen as stopping always to a (k+1)-level since, after identifying the correct algorithm, we can predict the precise update of the other agent's knowledge, hence consider its entire reasoning up to its limit at level k.

Chapter 3

Multi-Agent Inverse Reinforcement Learning

In this chapter, we present an extension of LOGEL ([48], Section 2.4.4) to perform IRL in a 2-agents environment in which the opponent is learning, thus deriving the settings from LfL [26]. We take the same assumption as in LOGEL that the learner is optimizing its utility function using gradient-based algorithms but we take the learning rate α as known and constant.
We start by applying LOGEL considering a simple gradient-descent opponent that learns only from its interactions with the environment without being aware of its Multi-Agent nature (Section 3.1). Then we consider two variations of this scenario introducing two famous gradient-based algorithms presented in Section 2.3.1 like IGA-PP [72] and LOLA [19] (Section 3.2).

3.1 Single-Agent Reinforcement Learning approach

The algorithm structure is the same as the one in LOGEL. Given a set of trajectories $\mathcal{T}$, the reward weights $\boldsymbol{\omega}$ are obtained in two steps. First, the algorithm recovers the parameters $\boldsymbol{\theta}$ generating the policies shown in the given set of trajectories, then it uses those parameters to recover the reward weights, as shown in Algorithm 8.
For the standard policy-gradient opponent, like GPOMDP, the formula is directly taken from LOGEL.
Given $\Delta_k = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$, the minimization problem at learning step k is defined as:

$$\widehat{\boldsymbol{\omega}} = \min_{\boldsymbol{\omega}} \sum_{k'=1}^{k} \|\Delta_{k'} - \alpha \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'} \boldsymbol{\omega}\|_2^2 \tag{3.1}$$

Algorithm 8: LOGEL 2-steps IRL

Data: Trajectories $\mathcal{T}_{i=1,\dots,N}$
Output: $\boldsymbol{\omega}$
Use Behavioral Cloning (see Algorithm 7) to estimate $\boldsymbol{\theta}$
Compute $\boldsymbol{\omega}$ with Equation (3.3)

Moreover, LOGEL's authors prove that, if the matrix $\sum_{k'=1}^{k} \alpha \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'}$ is full-rank, then the optimization problem can be solved in closed form by:

$$\widehat{\boldsymbol{\omega}} = \left(\sum_{k'=1}^{k} \alpha^2 \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'}^T \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'} \right)^{-1} \left(\sum_{k'=1}^{k} \alpha \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'}^T \Delta_{k'} \right) \tag{3.2}$$

We can call $A_{k'} = \alpha \nabla_{\boldsymbol{\theta}} \boldsymbol{\psi}_{k'}$ and rewrite (3.2) in the more general form:

$$\widehat{\boldsymbol{\omega}} = \left(\sum_{k'=1}^{k} A_{k'}^T A_{k'} \right)^{-1} \left(\sum_{k'=1}^{k} A_{k'} \Delta_{k'} \right) \tag{3.3}$$

3.2 Agent-aware with Future Policy Prediction approach

In Multi-Agent Reinforcement Learning, update algorithms may be non-rational and the learning agent may not always change its policy parameters following the direction of the gradient. For this reason, we consider other possible algorithms like IGA-PP and LOLA and we start by recollecting their update rules.
Given agent 1 and opponent agent 2 the update for IGA-PP is formulated as:

$$\boldsymbol{\theta}_{k+1}^1 = \boldsymbol{\theta}_k^1 + \alpha^1 (\nabla_{\boldsymbol{\theta}^1} J^1(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2) + \alpha^2 \nabla_{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2} J^1(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2)^T \nabla_{\boldsymbol{\theta}^2} J^2(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2))$$

while the update for LOLA is formulated as:

$$\boldsymbol{\theta}_{k+1}^1 = \boldsymbol{\theta}_k^1 + \alpha^1 (\nabla_{\boldsymbol{\theta}^1} J^1(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2) + \alpha^2 \left(\nabla_{\boldsymbol{\theta}^2} J^1(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2)^T \nabla_{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2} J^2(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2) \right)^T)$$

Therefore, given $\Delta_k^2 = \boldsymbol{\theta}_{k+1}^2 - \boldsymbol{\theta}_k^2$ and $J^1(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^2) = (\boldsymbol{\psi}_k^1)^T \boldsymbol{\omega}^1$ the optimization problem for IGA-PP over the opponent agent can be written as:

$$\widehat{\boldsymbol{\omega}} = \min_{\boldsymbol{\omega}} \sum_{k'=1}^{k} \left\| \Delta_{k'}^2 - \alpha^2 (\nabla_{\boldsymbol{\theta}^2} \boldsymbol{\psi}_{k'}^2 + \alpha^1 \left((\nabla_{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2} \boldsymbol{\psi}_{k'}^2)^T ((\nabla_{\boldsymbol{\theta}^1} \boldsymbol{\psi}_{k'}^1)^T \boldsymbol{\omega}^1) \right)) \boldsymbol{\omega} \right\|_2^2$$

and, with an analogous proof to LOGEL's, given the matrix:

$$A_{k'} = \alpha^2(\nabla_{\boldsymbol{\theta}^2}\boldsymbol{\psi}^2_{k'} + \alpha^1\left((\nabla_{\boldsymbol{\theta}^1,\boldsymbol{\theta}^2}\boldsymbol{\psi}^2_{k'})^T((\nabla_{\boldsymbol{\theta}^1}\boldsymbol{\psi}^1_{k'})^T\boldsymbol{\omega}^1)\right))$$

under the similar assumption that $\sum_{k'=1}^{k} A_{k'}$ is full-rank, we can express it in closed form as:

$$\widehat{\boldsymbol{\omega}} = \left(\sum_{k'=1}^{k} A^T_{k'}A_{k'}\right)^{-1}\left(\sum_{k'=1}^{k} A_{k'}\Delta^2_{k'}\right) \tag{3.4}$$

The same is valid for LOLA where the optimization problem is:

$$\widehat{\boldsymbol{\omega}} = \min_{\boldsymbol{\omega}} \sum_{k'=1}^{k} \left\|\Delta^2_{k'} - \alpha^2(\nabla_{\boldsymbol{\theta}^2}\boldsymbol{\psi}^2_{k'} + \alpha^1\left((\nabla_{\boldsymbol{\theta}^1}\boldsymbol{\psi}^2_{k'})^T((\nabla_{\boldsymbol{\theta}^1,\boldsymbol{\theta}^2}\boldsymbol{\psi}^1_{k'})^T\boldsymbol{\omega}^1)\right)^T)\boldsymbol{\omega}\right\|_2^2$$

and, given the matrix

$$A_k = \alpha^2(\nabla_{\boldsymbol{\theta}^2}\boldsymbol{\psi}^2_{k'} + \alpha^1\left((\nabla_{\boldsymbol{\theta}^1}\boldsymbol{\psi}^2_{k'})^T((\nabla_{\boldsymbol{\theta}^1,\boldsymbol{\theta}^2}\boldsymbol{\psi}^1_{k'})^T\boldsymbol{\omega}^1)\right)^T)$$

under the similar assumption that $\sum_{k'=1}^{k} A_{k'}$ is full-rank, we can express it in closed form as:

$$\widehat{\boldsymbol{\omega}} = \left(\sum_{k'=1}^{k} A^T_{k'}A_{k'}\right)^{-1}\left(\sum_{k'=1}^{k} A_{k'}\Delta^2_{k'}\right) \tag{3.5}$$

In the next chapter, we will show how the IRL methods for the Single-Agent and the Agent-Independent algorithms obtain good results in challenging Multi-Agent environments.

Chapter 4

Experimental Evaluation of Multi-Agent Inverse Reinforcement Learning

In this chapter, we evaluate the results of the IRL algorithms on a set of environments. The first one is a bimatrix game called Matching Pennies, which is described in Section 4.1.1, while the other two are continuous grid-worlds presented in Section 4.1.2.
We present the results with a comparison of two agents, playing both with algorithm IGA-PP and estimating gradients, $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ of the other player, first under the hypothesis that the opponent is using a standard gradient descent algorithm, then under the hypothesis it is using IGA-PP. We perform the experiments both in Matching Pennies (see Section 4.2) and, providing the correct $\boldsymbol{\theta}$, in the continuous environment Gridworld 1 (see Section 4.3). In the graphs, the shaded region represents the 95% confidence interval.

4.1 Environments

We now present the different typologies of environment used in this document.

4.1.1 Bimatrix

Bimatrix games are a class of problems in which two players perform simultaneously an action chosen from a finite set. It is called bimatrix because it is usually represented as a couple of matrices, the first one describing the

	Heads	Tails
Heads	+1 / -1	-1 / +1
Tails	-1 / +1	+1 / -1

Figure 4.1: Matching Pennies payoff table.

rewards of the first agent and the second one describing the rewards of the second agent.
We consider a particular bimatrix game called Matching Pennies (MP). In MP, the two players, called Even and Odd, have both a coin and have to decide to play their coin as Heads or Tails, then they reveal their choices simultaneously. If the two coins match (both Heads or Tails), then Even obtain both coins, hence getting a payoff of +1 while Odd gets -1. If the coins do not match (one Heads and one Tails), Odd receive the two coins, hence getting a payoff of +1 while Even gets -1 (see Table 4.1). Since the sum of the payoffs is always zero, Matching Pennies is called a zero-sum game. Matching Pennies is a good benchmark since there is no pure strategy Nash Equilibrium and there is a unique Nash Equilibrium in the mixed strategy in which both players choose Heads and Tails with equal probability. In Matching Pennies, as well as in all zero-sum games, gradient descent algorithms have a cyclical behavior and never converge to a Nash Equilibrium [5], making them interesting for our goals with respect to other bimatrix games.
All our simulations are performed 20 times and the results are averaged on them. We consider the class of Boltzmann policies (as shown with Equation 2.5) with temperature $\tau_\pi = 1$, where a single parameter θ is mapped to the probability of taking action 1 or 2.
We take learning rate $\alpha = 1$ and the starting θ^i of agent i is sampled from a uniform distribution in range $[-2, +2]$. For the estimation of θ and $\boldsymbol{\omega}$, the agents assume the other agent considers the same feature vectors, i.e. one feature as the policy parameter θ and four features, the four possible outcomes of the match, as the feature vector of the reward $\boldsymbol{\omega}$.

4.1.2 Continuous Gridworld

Gridworlds are environments represented with 2D-matrices $N \times M$, where N and M are the dimensions of the matrix. In gridworlds, the agent can generally move north, east, south, west or stay still and take other actions that are dependent on the specific environment. The agent, moving through

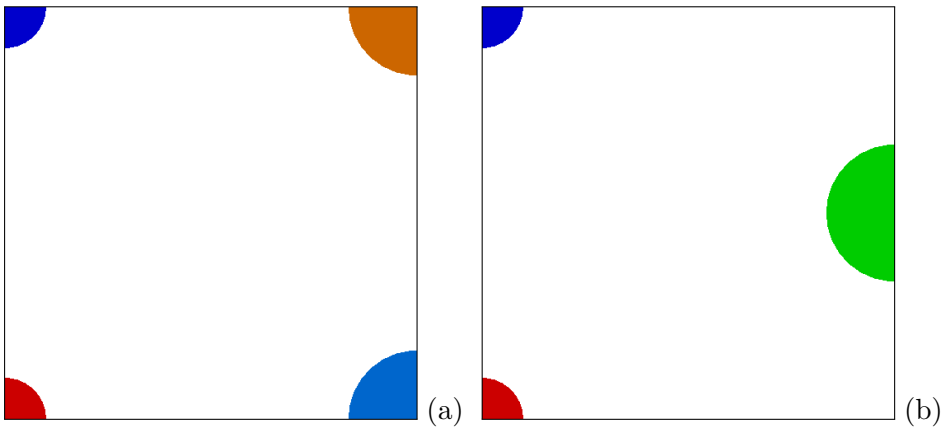


Figure 4.2: Gridworld 1 (a) and Gridworld 2 (b).

the environment, has to reach a goal to achieve a reward.
In continuous gridworlds, the environment is not a discrete matrix anymore but a continuous space. In our environments, similar to the ones in [49], the grid has size 3×3 and the agent chooses an action that corresponds to the direction to take, expressed as the angle from the horizontal direction of reference, and moves of 1 length in that direction. The starting state of the first agent is the lower-left corner, while the one of the second agent is the upper-left. Actions have 0.9 probability of success and 0.1 probability of being taken randomly. Moreover, the action is taken from a Gaussian distribution where the mean is the angle chosen and the variance is 0.5. The actions of the two agents are chosen and performed simultaneously.
If the distance between the two agents is lower than 0.5, the two agents collide, their movement is canceled, thus they remain in their previous positions, and receive a penalty of 1. If an agent takes an action that would move it outside of the grid, which is not allowed, the agent is placed on the nearest border of the grid from its illegal ending position. When an agent reaches its goal, it receives a reward of 10.
To represent the state of the environment at each time, we use Radial Basis Functions (RBF, [11]) as general feature function approximators. As RBFs we use Gaussian functions and, for a given state s, each state-feature $\phi_i(s)$ is represented by [28]:

$$\phi_i(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-||c_i - s||^2/2\sigma^2} \tag{4.1}$$

where c_i is the center of the i-th Gaussian and σ^2 is the variance. The Gaussians are usually distributed tiling the state-space, evenly distributing the centers along each dimension. This way, b^d centers are needed, where d is the number of dimensions and b is the number of centers aligned on each axis. With this distribution of the centers, and given $N \times N$ the size of the grid, we chose $\sigma^2 = \frac{N}{2(b-1)}^2$ as in [49]. For the estimation of $\boldsymbol{\omega}$, the agents assume the other agent considers the same feature vector, i.e. three features, one for reaching the goal, one for colliding with each other and one for every other possible outcome of a step.

The two continuous gridworlds differ from each other because of the objectives of the agents.
In Gridworld 1 (see Figure 4.2a), the agents have each of them their own goal, which are circles with radius 0.5 placed on the opposite corners from their starting positions. The small blue quadrant is the starting point of agent 1, the big blue quadrant is its goal and the small and big red quadrants are agent 2's starting point and goal respectively. An agent reaches its goal if its ending position after a movement, the center of the agent's circle, is inside the circular area of the goal. When that happens, after each agent receives its reward for the last movement, the following actions are always taken as zero without actually performing them on the environment and both agents receive zero reward for the rest of the episode until reaching the horizon T.
In Gridworld 2 (see Figure 4.2b), the agents have a common goal, represented as a green circle of radius 0.5 with the center situated on the opposite side of their starting points, equally distant from both of them, but a new condition is introduced. At the start of each episode, a token is randomly assigned to one of the players. A player can obtain the reward associated with reaching the goal only if it reaches the goal and it owns the token. In this environment, the collision does not generate a penalty but instead changes the ownership of the token and it is the only way for the token to change possession.

4.2 IRL in Bimatrix

We start by analyzing two scenarios in which the agents are playing Matching Pennies, both are using algorithm IGA-PP and both are estimating the gradients, the parameters of the opponent with Behavioral Cloning and the parameters of the opponent's reward function.
The only difference between the scenarios is that in the first scenario they

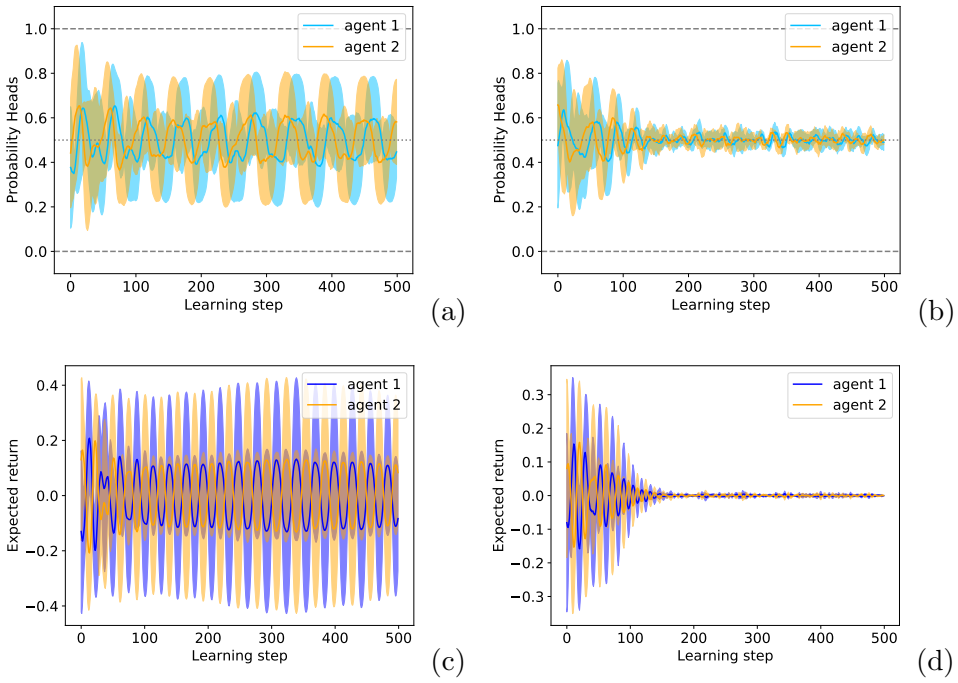


Figure 4.3: Matching Pennies results. Probability of choosing Heads with IRL on GPOMDP (a) and on IGA-PP (b). Expected return with IRL on GPOMDP (c) and on IGA-PP (d).

believe the other agent is using standard gradient descent and therefore they use Equation (3.3) to estimate the opponent's reward function. In the second scenario instead, they know the algorithm used by the other agent, IGA-PP, and therefore use Equation (3.4) to estimate the opponent's reward function.
For the experiments, we run the simulation 20 times and each run is composed of 500 learning steps with 250 plays for each step. The learning rates of the agents are both equal to 1.

The results in Figure 4.3 show that, using the omega estimation formula (3.3), the two agents do not converge and instead diverge due to the lack of the correct estimation of the other agent's $\boldsymbol{\omega}$ (see Figure 4.3a and 4.3c). Instead, using IGA-PP omega estimation formula (3.4), the two agents learn the correct opponent's reward function and converge to the Nash Equilibrium (see Figure 4.3b and 4.3d).

We obtain similar results if instead than two agents playing IGA-PP we use two agents playing LOLA (see Appendix B).

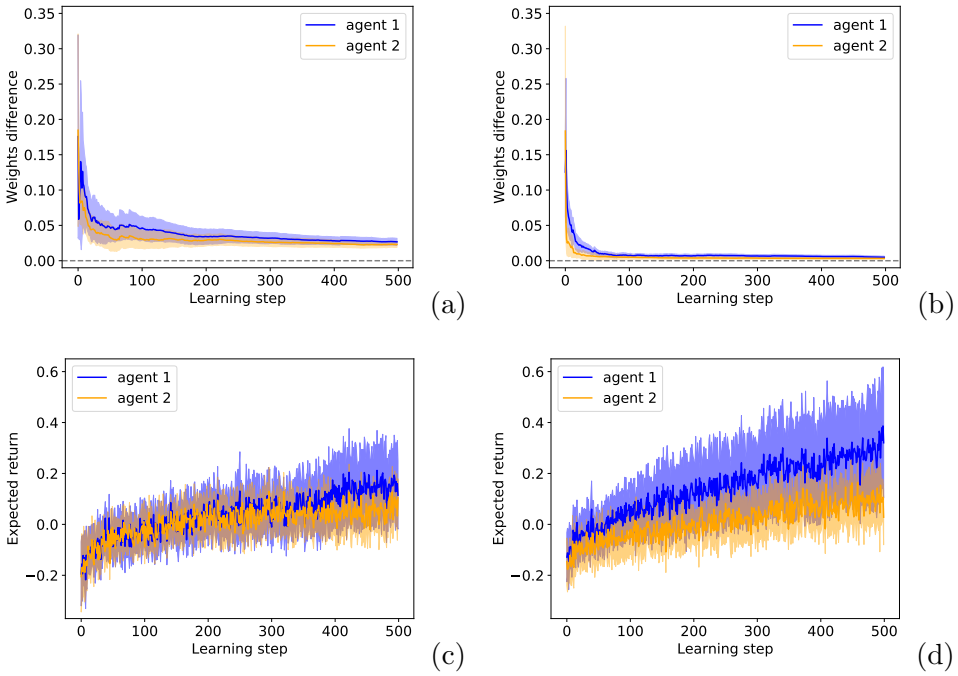


Figure 4.4: Gridworld results. Estimation error with IRL on GPOMDP (a) and on IGA-PP (b). Expected return with IRL on GPOMDP (c) and on IGA-PP (d).

4.3 IRL in Continuous Gridworld

We perform the same experiments in Gridworld 1. In these scenarios, the agents are using algorithm IGA-PP and both know the opponent parameters and estimate only the gradients and the rewards.
As before, the only difference between the experiments is that, in the first, the agents believe the other agent is using standard gradient descent and estimate the opponent's reward function using Equation (3.3) while, in the second, they know the other agent is using IGA-PP and estimate opponent's reward function using Equation (3.4).
For the experiments, we run the simulation 20 times and each run is composed of 500 learning steps. In every learning step, 50 trajectories are generated and each trajectory is 15 steps long. The learning rates of the agents are both equal to 0.1.

The results in Figure 4.4 show that, using the omega estimation formula (3.3), the two agents do not recover a precise estimation of $\boldsymbol{\omega}$ (see Figure 4.4a). Instead, using IGA-PP omega estimation formula (3.4), the agents

both learn, in a quicker way, a more precise estimation of ω (see Figure 4.4b). Nevertheless, the error in the estimation of omega does not significantly influence the behavior of the agents in the environment, since both achieve good results in a similar number of learning steps (see Figure 4.4c and Figure 4.4d).

Chapter 5

Algorithm Identification

In this chapter, we present a method to identify the algorithm used by another learning agent in a Multi-Agent environment from a finite set of possible algorithms (Section 5.1). This can be achieved independently by the algorithm already used by the performing agent, thus without changing any of the current behaviors of the agent. For this reason, we call this the *passive identification* of an agent's algorithm (Section 5.2). Then we present an exploratory method to choose a strategy that lets the agent identify the other agent's algorithm in a reduced number of steps with respect to the passive method. This is done by selecting at each step the strategy that maximizes the distances in the opponent's estimated behaviors depending on the different algorithms. We call this the *active identification* method (Section 5.3).

5.1 Overview

To present the algorithm identification methods, we start from the formalization of the environment of a Stochastic Game as presented in Section 2.2.1. We consider a Stochastic Game with two agents, in which agent 1 is the agent performing the identification over the algorithm used by agent 2, which is included in a set of possible algorithms $\mathcal{M} = \{m_1, \dots, m_M\}$, where M is the number of possible algorithms. Each algorithm is defined by its update rule and its hyperparameters. At some step k, the interaction of the agents with the environment generates a set of demonstrations $\mathcal{T}_k$ that comprises the information on the states visited and the actions taken. We assume that agent 1 knows agent 2's parameters only at the beginning of the learning and that it is not changing its learning algorithm during the

game.
Thanks to these assumptions, agent 1 can always compute the correct parameters of agent 2 for every algorithm in the given set, independently of agent 2's knowledge of agent 1's parameters. What it does not know is which of the algorithm is the correct one and therefore the correct parameters. This problem can be solved by our identification algorithm.

5.2 Passive Algorithm Identification

The *passive algorithm identification* at learning step k is performed taking as input the set of possible algorithms $\mathcal{M}_k$, the known parameters of our agent $\boldsymbol{\theta}_k^1$, the parameters of the opponent agent $\boldsymbol{\theta}_k^{2,m}$ for every algorithm m in $\mathcal{M}_k$, the trajectories generated with those parameters $\mathcal{T}_k$ and the threshold ζ, which is used to keep or eliminate candidate algorithms (see Algorithm 9).
The identification process starts computing, for each algorithm m, the goodness of fit of the model, generated by the the parameters of agent 1 $\boldsymbol{\theta}_k^1$ and agent 2 $\boldsymbol{\theta}_k^{2,m}$, to the sample of data represented by the trajectories $\mathcal{T}_k$. This is performed computing the log-likelihood $\mathcal{L}_k^m$ with Equation (2.13).
Then, it selects the value of the maximum log-likelihood $\widehat{\mathcal{L}_k}$ among the ones calculated, formulated as:

$$\widehat{\mathcal{L}_k} = \max_{m \in \mathcal{M}} \left(\ell(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^{2,m} | \mathcal{T}_k) \right)$$

where $\ell(\,\cdot \,|\, \cdot\,)$ is the log-likelihood as defined in (2.13). Afterward, we initialize an empty set which will be filled with the algorithms which satisfied the criterion of the threshold.
Next, for each algorithm $m \in \mathcal{M}$, the log-likelihood $\mathcal{L}_k^m$ is compared to the maximum log-likelihood $\widehat{\mathcal{L}_k}$ performing a simple-vs-simple hypothesis test in logarithmic form against the threshold ζ, as presented in Section 2.5.6. Given $\Lambda = -2(\mathcal{L}_k^m - \widehat{\mathcal{L}_k})$, if $\Lambda < \zeta$ the algorithm is kept. Otherwise, if $\Lambda \geq \zeta$ the algorithm is discarded. In this test, the alternative hypothesis is the model with the highest log-likelihood and the null hypothesis to be evaluated is the model generated with algorithm m that can be rejected or not. If the outcome of the test does not reject the null hypothesis, the algorithm m can not be excluded and, therefore, is added to the list of the algorithms to be returned, i.e. the algorithms that satisfied the threshold evaluation.

Algorithm 9: Passive Algorithm Identification

Input: Set of possible algorithms at step k $\mathcal{M}_k$, threshold ζ, parameters of agent 1 at k $\boldsymbol{\theta}_k^1$, joint parameters of agent 2 at step k for each algorithm m $\boldsymbol{\theta}_k^{2,m}$
Data: Trajectories at step k $\mathcal{T}_k$
Output: Set of possible algorithms $\mathcal{M}_{k+1} \in \mathcal{M}_k$
for *each algorithm* m *in* $\mathcal{M}_k$ **do**
  $\mathcal{L}_k^m = \ell(\boldsymbol{\theta}_k^1, \boldsymbol{\theta}_k^{2,m} | \mathcal{T}_k)$
end
$\widehat{\mathcal{L}_k} = \max_m(\mathcal{L}_k^m)$
$\mathcal{M}_{k+1} = \emptyset$
for *each algorithm* m *in* $\mathcal{M}_k$ **do**
  $\Lambda = -2(\mathcal{L}_k^m - \widehat{\mathcal{L}_k})$
  if $\Lambda < \zeta$ **then**
    Add algorithm m to $\mathcal{M}_{k+1}$
end
return $\mathcal{M}_{k+1}$

5.3 Active Algorithm Identification

The exploration strategy is determined with the *active algorithm identification* method.

We start recollecting that the identification of the opponent's algorithm is based on the likelihood that each possible $\boldsymbol{\theta}_k^{2,m}$ of the opponent, following algorithm m, generated the observed trajectories at step k. To accelerate the identification of the correct algorithm, we decided to facilitate the discarding of at least one possible algorithm at each step.
In order to do so, we need to differentiate as much as possible the trajectories that one algorithm would generate with respect to another one. Differentiating the trajectories generated by the algorithms means having the greatest distance between their probability distributions. Hence, in order to maximize this distance, we chose to measure it using the Kullback-Leibler (KL) divergence (see Section 2.7.1).

We chose the KL divergence between two probability distributions because, given probability distributions p and q and a set of $X_1, \ldots, X_n$ i.i.d. samples generated from the distribution p, recollecting that the log-likelihood

ratio between p and q normalized by n is:

$$\lambda_n = \frac{1}{n} \sum_{i=1}^{n} \log \frac{p(X_i)}{q(X_i)}$$

we can notice that λ_n is itself a random variable so $\lambda_n \to \mathbb{E}[\lambda_n]$, i.e. as n grows, λ_n tends to its expected value. The expected value of λ_n is:

$$\begin{aligned}
\mathbb{E}[\lambda_n] &= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} \log \frac{p(X_i)}{q(X_i)}\right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[\log \frac{p(X_i)}{q(X_i)}\right] \\
&= \int \log \frac{p(x)}{q(x)} p(x) d(x) \\
&= D_{KL}\left(p(x) \| q(x)\right)
\end{aligned}$$

So the expected value of the likelihood ratio test is equal to the KL divergence between the two probability distribution.

One of the problem encountered is that, having observed the trajectories at step k and not knowing the dynamics of the environment, we can not directly influence the policy of the opponent's agent during the following step $k+1$, since its strategy is computed with the trajectories already observed at step k. After observing the trajectories at step k, we can only influence the opponent's policy at step $k+2$.
To influence it, we choose the strategy $\boldsymbol{\theta}^1_{k+1}$ for step $k+1$ that would induce one of the other agent's possible probability distributions at step $k+2$ to have the greatest KL divergence from the one generated by a different algorithm.

This generates a second complication: while choosing $\boldsymbol{\theta}^1_{k+1}$, we can not simulate the trajectories that this policy will generate, so we are not able to compute the opponent's updates, and its possible policies, since they are dependant on the trajectories observed. To solve this problem we have to rely on off-policy estimation (see Section 2.1) and Importance Sampling (see Section 2.6).
Since we do not have the trajectories at step $k+1$, we need to perform an off-policy estimation to approximate the opponent's policy parameters at step $k+2$ using data obtained in the previous learning steps following different policies. To reduce the variance due to the different distributions, we adopt Multiple Importance Sampling (see Section 2.6.1) to compute the

Algorithm 10: Active Algorithm Identification

Input: Possible algorithms' list at step k $\mathcal{M}_k$, history of parameters of agent 1 $\boldsymbol{\theta}^1_{1:k}$, history of parameters of agent 2 for each algorithm m $\boldsymbol{\theta}^{2,m}_{1:k}$

Data: Trajectories' history $\mathcal{T}_{1:k}$

Output: Parameters of agent 1 at $k+1$ maximizing the KL-Divergence $\widehat{\boldsymbol{\theta}^1}_{k+1}$

for *each algorithm* m *in* $\mathcal{M}_k$ **do**
  $\boldsymbol{\theta}^{2,m}_{k+1} = Upd^m(\mathcal{T}_k, \boldsymbol{\theta}^1_k, \boldsymbol{\theta}^{2,m}_k)$
  $\boldsymbol{\theta}^{2,m}_{1:k+1} = \boldsymbol{\theta}^{2,m}_{1:k} \frown \boldsymbol{\theta}^{2,m}_{k+1}$
end
while D^{MAX}_{KL} *does not converge to a maximum* **do**
  define new $\boldsymbol{\theta}^1_{k+1}$
  for *each algorithm* m *in* $\mathcal{M}_k$ **do**
    $MIS^m = MIS^m_k(\mathcal{T}_{1:k}, \boldsymbol{\theta}^1_{1:k}, \boldsymbol{\theta}^{2,m}_{1:k+1}, \boldsymbol{\theta}^1_{k+1})$
    $\boldsymbol{\theta}^{2,m}_{k+2} = Upd^m(\mathcal{T}_k, MIS^m, \boldsymbol{\theta}^1_{k+1}, \boldsymbol{\theta}^{2,m'}_{k+1})$
    $PD^m = PD(\boldsymbol{\theta}^{2,m}_{k+2})$
  end
  for *each algorithm* m *in* $\mathcal{M}_k$ **do**
    for *each algorithm* m' *in* $\mathcal{M}_k$ **do**
      $D^{m,m'}_{KL} = D_{KL}(PD^m \| PD^{m'})$
    end
  end
  $D^{MAX}_{KL} = \max_{\substack{m \in \mathcal{M}_k \\ m' \in \mathcal{M}_k}} (D^{m,m'}_{KL})$
end
return $\widehat{\boldsymbol{\theta}^1}_{k+1}$

adjustments needed to correctly estimate the policy parameters at step $k+2$ given the history of the previous values and trajectories.

Therefore, given $(\,\cdot\,)_{1:k}$ the history of values $(\,\cdot\,)$ from step 1 to step k, the method takes as input the set of possible algorithms $\mathcal{M}_k$ at step k, the history of known parameters of our agent $\boldsymbol{\theta}^1_{1:k}$, the history of parameters of the opponent agent $\boldsymbol{\theta}^{2,m}_{1:k}$ for every algorithm m in $\mathcal{M}_k$, the trajectories generated with those parameters $\mathcal{T}_{1:k}$ and the threshold ζ.

With this data, we need to compute the expected parameters of the opponent agent $\boldsymbol{\theta}^{2,m}_{k+1}$ at step $k+1$ for each algorithm m, computed with the known algorithms' update rules, and we write $\boldsymbol{\theta}^{2,m}_{1:k+1} = \boldsymbol{\theta}^{2,m}_{1:k} \frown \boldsymbol{\theta}^{2,m}_{k+1}$ the concatenation of $\boldsymbol{\theta}^{2,m}_{1:k}$ and $\boldsymbol{\theta}^{2,m}_{k+1}$. Then, for every algorithm m, we compute the

Multiple Importance Sampling ratios (see Section 2.6.1) for the parameters $\boldsymbol{\theta}^1_{k+1}$ and $\boldsymbol{\theta}^{2,m}_{k+1}$ at step $k+1$. Then, we use again the update method of each algorithm to estimate the opponent's parameters $\boldsymbol{\theta}^{2,m}_{k+2}$ for that algorithm m at step $k+2$. As presented before, due to the impossibility to generate trajectories with the parameters at step $k+1$, we need to perform this operation including in the calculation the MIS ratios. Now, for every algorithm m, we need to compute the probability distribution of the estimated opponent's parameters at step $k+2$, expressed as $PD^m(\boldsymbol{\theta}^{2,m}_{k+2})$.
Finally, for every pair of algorithms (m,m'), we compute the Kullback-Leibler divergence $D^{m,m'}_{KL}$ and take its maximum value D^{MAX}_{KL}.
The exploration algorithm then chooses the values of $\boldsymbol{\theta}^1_{k+1}$ that maximize D^{MAX}_{KL} obtaining the parameters $\widehat{\boldsymbol{\theta}^1}_{k+1}$, in parameter space Θ, of the strategy to play the following step (see Algorithm 10).

We can now formulate the optimization as:

$$\widehat{\boldsymbol{\theta}^1}_{k+1} = \max_{\boldsymbol{\theta}^1_{k+1} \in \Theta} D^{MAX}_{KL}(\boldsymbol{\theta}^1_{k+1}) \tag{5.1}$$

where:

$$D^{MAX}_{KL}(\boldsymbol{\theta}^1_{k+1}) = \max_{\substack{\boldsymbol{\theta}^1_{k+1} \in \Theta \\ m \in \mathcal{M}_k \\ m' \in \mathcal{M}_k}} D_{KL}\Big(PD(\boldsymbol{\theta}^{2,m}_{k+2}) \| PD(\boldsymbol{\theta}^{2,m'}_{k+2}) \Big)$$

is the formula to obtain the maximum KL divergence given the set of policy parameters $PD(\boldsymbol{\theta}^{2,m}_{k+2})$ of parameters $\boldsymbol{\theta}^{2,m}_{k+2}$ dependent on parameters $\boldsymbol{\theta}^1_{k+1}$. Parameters $\boldsymbol{\theta}^{2,m}_{k+2}$ can be expressed as:

$$\boldsymbol{\theta}^{2,m}_{k+2} = Upd^m(\mathcal{T}_k, MIS^m_k(\mathcal{T}_{1:k}, \boldsymbol{\theta}^1_{1:k}, \boldsymbol{\theta}^{2,m}_{1:k+1}, \boldsymbol{\theta}^1_{k+1}), \boldsymbol{\theta}^1_{k+1}, \boldsymbol{\theta}^{2,m'}_{k+1})$$

in which $Upd^m(\,\cdot\,)$ is the update function of the opponent agent using algorithm m.

Given n_j trajectories for learning step $j \leq k$ and assuming a fixed trajectory length $T(\tau)$, the Multiple Importance Sampling ratios for each timestep t is formulated as (see Equation (2.15)):

$$MIS^m_{k,t}(\mathcal{T}_{1:k}, \boldsymbol{\theta}^1_{1:k}, \boldsymbol{\theta}^{2,m}_{1:k+1}, \boldsymbol{\theta}^1_{k+1}) = \sum_{j=1}^{k} \sum_{i=1}^{n_j} \frac{\prod_{t'=1}^{t-1} \pi^m_{k+1}(\boldsymbol{a}_{j,i,t'} | s_{j,i,t'})}{\sum_{l=1}^{k} \tau_l \prod_{t'=1}^{t-1} \pi^m_l(\boldsymbol{a}_{l,i,t'} | s_{l,i,t'})}$$

and the set of the Multiple Importance Sampling ratios for all t is expressed as:

$$MIS_k^m = \{MIS_{k,1}^m, \ldots, MIS_{k,t}^m, \ldots, MIS_{k,T(\tau)}^m\}$$

Therefore, the expanded optimization formula is:

$$\widehat{\boldsymbol{\theta}^1}_{k+1} = \max_{\substack{\boldsymbol{\theta}_{k+1}^1 \in \Theta \\ m \in \mathcal{M}_k \\ m' \in \mathcal{M}_k}} D_{KL}(PD(Upd^m(\mathcal{T}_k, MIS_k^m(\mathcal{T}_{1:k}, \boldsymbol{\theta}_{1:k}^1, \boldsymbol{\theta}_{1:k+1}^{2,m}, \boldsymbol{\theta}_{k+1}^1), \boldsymbol{\theta}_{k+1}^1, \boldsymbol{\theta}_{k+1}^{2,m}) \parallel$$
$$PD(Upd^{m'}(\mathcal{T}_k, MIS_k^{m'}(\mathcal{T}_{1:k}, \boldsymbol{\theta}_{1:k}^1, \boldsymbol{\theta}_{1:k+1}^{2,m'}, \boldsymbol{\theta}_{k+1}^1), \boldsymbol{\theta}_{k+1}^1, \boldsymbol{\theta}_{k+1}^{2,m'})))$$

This maximization function can not be expressed in closed form and it is difficult to optimize with a gradient descent approach. Therefore, in our experiments, we convert the continuous space of the parameters into a discrete one and optimize over that set of parameters.

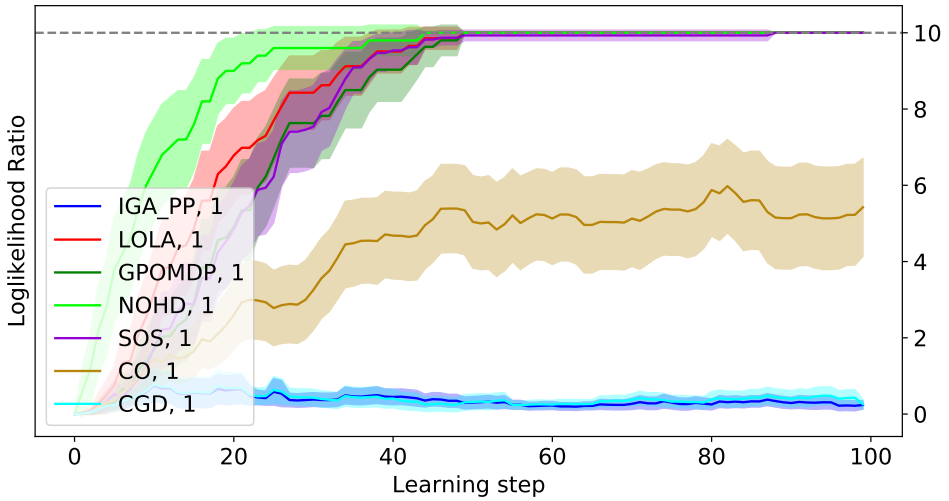
Chapter 6

Experimental Evaluation of Algorithm Identification

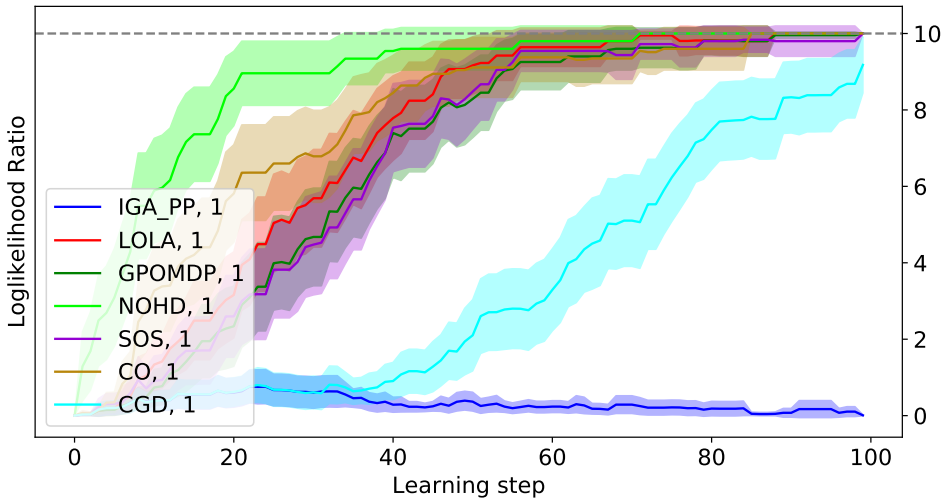
We have already presented in Chapter 4 some of the advantages of knowing the algorithm used by the opponent, enabling the agents to converge instead of diverging. In this chapter, we present the results obtained with the passive identification technique presented in Section 5.2 and compare those results with the active method.
The experiments are performed both in Matching Pennies (see Section 6.1) and in the continuous environment Gridworld 2 (see Section 6.2).
Then, we show the advantage of identifying the opponent's algorithm in Matching Pennies and how exploiting this information applying a Best Response strategy leads to obtaining a higher payoff (see Section 6.3). As before, the shaded region in the graphs represents the 95% confidence interval.

6.1 Identification in Bimatrix

We start applying the algorithm in Matching Pennies (see Section 4.1.1). In this scenario, agent 1 is the agent performing the identification and we assume it has a finite set of possible algorithms. Each algorithm, as described in Section 2.3.2, is defined with its starting parameters $\boldsymbol{\theta}_1^2$, its reward function $\boldsymbol{\omega}^2$ and its hyperparameters. To use passive identification as a baseline for the active identification, agent 1 plays a different random strategy at each learning step, while the opponent performs its algorithm. For the active method, because of the difficulties presented in Section 5.3, we convert the continuous space of the parameters into a discrete one and generate, at



(a)



(b)

Figure 6.1: Results of passive algorithm identification (a) and active algorithm identification (b) in Matching Pennies.

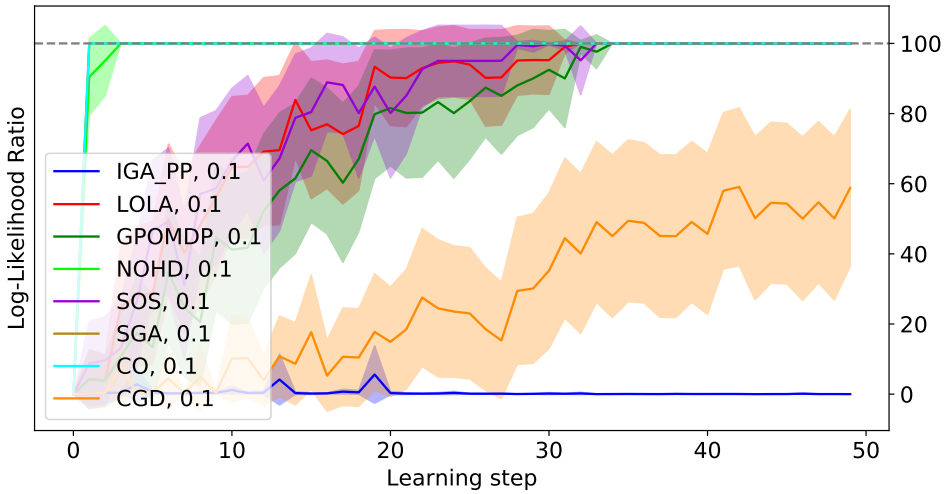
each learning step k, a vector of possible values for $\boldsymbol{\theta}^1_{k+1}$, linearly separated in the probability space, and perform the optimization over that set of values.
For the experiment, we run the simulation 20 times and each run is composed of 100 learning steps with 250 plays for each step. The discretization is performed linearly dividing the probability space of choosing Heads in 101 values in $[0, 1]$ and computing the corresponding policy parameters.

The results in Figure 6.1a show how the passive identification method (see Algorithm 9) can correctly discard many of the possible algorithms, leaving only algorithms which show similar behaviors. Nevertheless, using the active identification method presented in Algorithm 10, the identification of the algorithms is significantly faster and even very similar algorithms are differentiated (see Figure 6.1b).

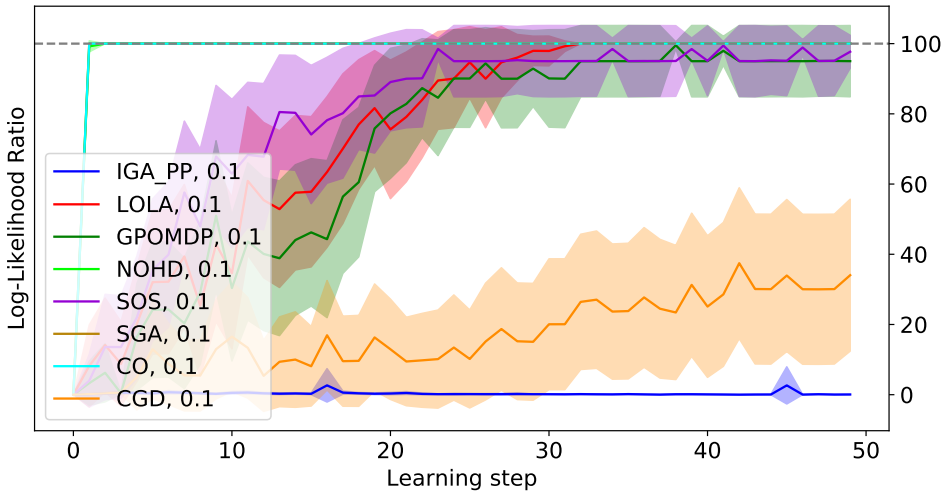
6.2 Identification in Gridworlds

As done before for the IRL methods, we perform the algorithm identification also in the continuous gridworld environment (see Section 4.1.2). We use Gridworld 2 for this experiment. As seen in the bimatrix scenario, agent 1 is the agent performing the identification and we assume it knows the set of possible algorithms of agent 2, its parameters θ^2_1 at the beginning of the learning and its reward function $\boldsymbol{\omega}^2$. For the passive identification, agent 1 plays a different random strategy at each learning step taken by a fixed pool of strategies, while the opponent performs its algorithm. For the active method, we choose, at each learning step k, a subset of possible strategy form the same fixed set as in the passive identification scenario, and perform the optimization over that set of strategies.
For the experiments, we run the simulation 20 times and each run is composed of 50 learning steps. In every learning step, 50 trajectories are generated and each trajectory is 15 steps long. The learning rates of the agents are both equal to 0.1. The set of possible strategies is created generating 100 random strategies and the subset is composed of 30 randomly chosen strategies.

The results in Figure 6.2a show how the passive identification method (see Algorithm 9) can correctly discard many of the possible algorithms, leaving only algorithms which show similar behaviors. In this environment,



(a)



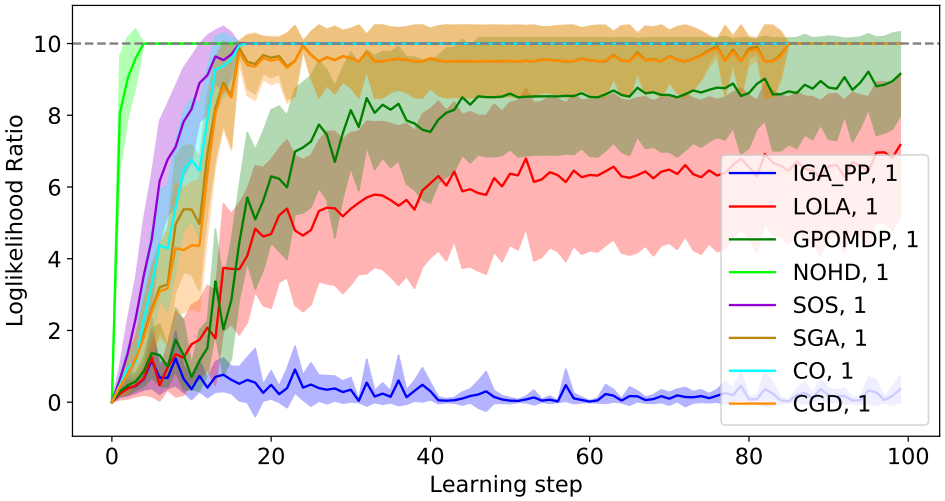
(b)

Figure 6.2: Results of passive algorithm identification (a) and active algorithm identification (b) in Gridworld 2.

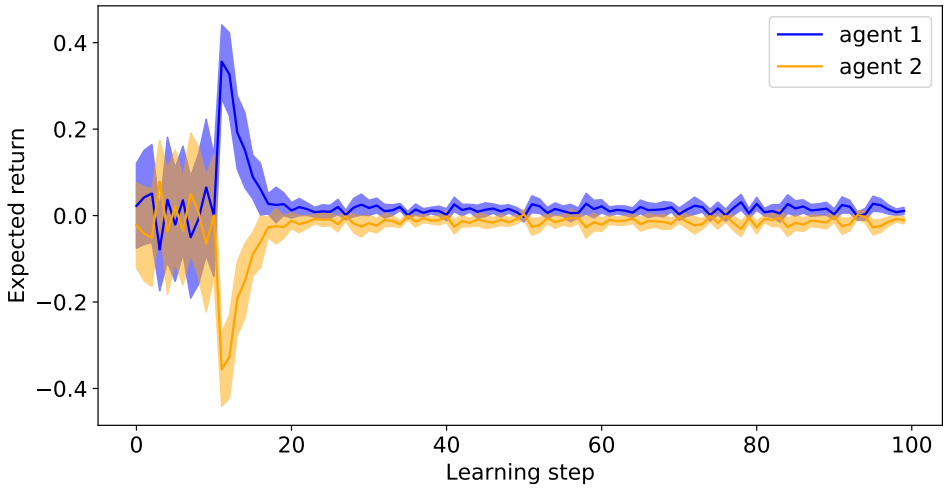
however, the active identification achieves similar results to the passive identification (see Figure 6.2b), probably due to the complexities of finding a good strategy for the exploratory purposes.

6.3 Best response with identified algorithm

In this section, we show how an agent performing algorithm identification can make use of this knowledge to exploit the strategy of its opponent.
We consider the game of Matching Pennies and both the agents use IGA-PP as the update algorithm. Since agent 1 performs algorithm identification, we use the same characteristics presented in Section 5.2. Moreover, since the goal of this experiment is to obtain a high reward, after ten steps of algorithm identification, we compute a Best Response to the estimated strategy of agent 2.
If the agent has not yet identified the opponent's algorithm, the agent computes the Best Response with respect to the algorithm with the highest likelihood at each time step. To compute the Best Response, we perform one step of gradient descent optimization over the expected return, given the estimated parameter θ^2_{k+1}, the set of trajectories at step k and the Importance Sampling due to the different probability distribution for both the agents.
The result, shown in Figure 6.3a, represents the identification of the opponent's algorithm, while Figure 6.3b shows how the expected return of the agent increases when it starts using the Best Response strategy and how the other agent, using IGA-PP, quickly converges to the Nash Equilibrium. Despite this, every time it shifts from the Nash Equilibrium, the agent is ready to exploit the opponent's behavior, increasing its reward.



(a)



(b)

Figure 6.3: Results of the identification (a) and expected return (b) of a Best Response strategy after the identification of opponent's algorithm.

Chapter 7

Conclusion

In this document, we expanded the usage of the LOGEL algorithm presented in [48] to the Multi-Agent environment. Furthermore, we proposed two other techniques to recover the reward function of an opponent using algorithm-specific optimization methods and presented the results for the IGA-PP algorithm. We presented a comparison between the techniques performed on the same agents following IGA-PP while modeling the opponent as GPOMDP or IGA-PP. We saw the impact that the correct assumption on the opponent's algorithm has on the estimation of its reward function and the overall learning process.
Then, we presented a method for a new approach in the field of opponent modeling. With this technique, the information recovered from the opponent is the algorithm it uses to update its policy parameters. Identifying the correct algorithm among a set of possibilities can be used to exploit the future behavior of the other agent, being "one step ahead" in a competitive environment or facilitating a collaboration if in a cooperative scenario. Then, we expanded this method providing an exploration method to facilitate the identification of the opponent's algorithm. Finally, we presented the results obtained with both the methods and a possible application in a competitive environment.

7.1 Future work

There are many possible directions for future work, being this a new approach to the problem of opponent modeling. We have selected some of them in this non-exhaustive list:

- The main direction for future work regards the possibility of performing standard IRL to recover the parameters of the opponent's reward function while, at the same time, identifying the algorithm it is using. This is a naturally ill-posed problem since one information is needed to model the other, but we are confident that, under some assumptions, great results could be achieved.
- Another direction could be to overcome the assumption made about the knowledge of the policy parameters of the other agent at the beginning of the learning. Those could be estimated with Behavioral Cloning techniques and then estimated again at each learning step, taking into account the possible algorithms, to refine the estimation.
- One of the difficulties encountered was the limited resources which restricted our capability of performing extended experiments in more complex scenarios. A possible environment could be Littman's soccer presented in [31] or a variation we presented in Appendix A due to its cyclical behavior similar to Matching Pennies.
- The algorithm identification method could be extended to k-level reasoning algorithms. This could be an interesting approach to identify the level of modeling power of the opponent and use this knowledge to stop the recursive reasoning at the minimum higher level needed to correctly model the opponent.
- We considered only 2-agent environments. A possible development could be to expand these methods to n-agent environments.
- The last direction for future work presented is to apply this method with Deep Reinforcement Learning algorithms. The challenges are evident, but achieving good results in this scenario could lead to a wider application in the real world of these techniques.

Bibliography

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

[2] Stefano V Albrecht and Peter Stone. "Autonomous agents modelling other agents: A comprehensive survey and open problems". In: *Artificial Intelligence* 258 (2018), pp. 66–95.

[3] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

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

[5] David Balduzzi et al. "The mechanics of n-player differentiable games". In: *arXiv preprint arXiv:1802.05642* (2018).

[6] Jonathan Baxter and Peter Bartlett. "Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms". In: (Jan. 2000).

[7] Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 2005.

[8] Michael Bowling. "Convergence and no-regret in multiagent learning". In: *Advances in neural information processing systems*. 2005, pp. 209–216.

[9] Michael Bowling and Manuela Veloso. "Multiagent learning using a variable learning rate". In: *Artificial Intelligence* 136.2 (2002), pp. 215–250.

[10] Michael Bowling and Manuela Veloso. "Rational and convergent learning in stochastic games". In: *International joint conference on artificial intelligence*. Vol. 17. 1. Lawrence Erlbaum Associates Ltd. 2001, pp. 1021–1026.

[11] David S Broomhead and David Lowe. *Radial basis functions, multivariable functional interpolation and adaptive networks*. Tech. rep. Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[12] George W Brown. "Iterative solution of games by fictitious play". In: *Activity analysis of production and allocation* 13.1 (1951), pp. 374–376.

[13] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[14] George Casella and Roger L Berger. *Statistical inference*. Vol. 2. Duxbury Pacific Grove, CA, 2002.

[15] Sonia Chernova and Manuela Veloso. "Confidence-based policy learning from demonstration using gaussian mixture models". In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. 2007, pp. 1–8.

[16] Vincent Conitzer and Tuomas Sandholm. "AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents". In: *Machine Learning* 67.1-2 (2007), pp. 23–43.

[17] Robert H Crites and Andrew G Barto. "Improving elevator performance using reinforcement learning". In: *Advances in neural information processing systems*. 1996, pp. 1017–1023.

[18] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. *A survey on policy search for robotics*. now publishers, 2013.

[19] Jakob N Foerster et al. "Learning with opponent-learning awareness". In: *arXiv preprint arXiv:1709.04326* (2017).

[20] Piotr J Gmytrasiewicz and Edmund H Durfee. "Rational coordination in multi-agent environments". In: *Autonomous Agents and Multi-Agent Systems* 3.4 (2000), pp. 319–350.

[21] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. "Model-free imitation learning with policy optimization". In: *International Conference on Machine Learning*. 2016, pp. 2760–2769.

[22] Junling Hu and Michael P Wellman. "Nash Q-learning for general-sum stochastic games". In: *Journal of machine learning research* 4.Nov (2003), pp. 1039–1069.

[23] Junling Hu and Michael P. Wellman. "Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm". In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML-98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 242–250. ISBN: 1558605568.

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

[25] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms". In: *Neural Computation* 6.6 (1994), pp. 1185–1201. DOI: 10.1162/neco.1994.6.6.1185.

[26] Alexis Jacq et al. "Learning from a Learner". In: *International Conference on Machine Learning*. 2019, pp. 2990–2999.

[27] Herman Kahn and Andy W Marshall. "Methods of reducing sample size in Monte Carlo computations". In: *Journal of the Operations Research Society of America* 1.5 (1953), pp. 263–278.

[28] George Konidaris. "Value function approximation in reinforcement learning using the Fourier basis". In: *Computer Science Department Faculty Publication Series* (2008), p. 101.

[29] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[30] Alistair Letcher et al. "Stable opponent shaping in differentiable games". In: *arXiv preprint arXiv:1811.08469* (2018).

[31] Michael L. Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine Learning Proceedings 1994* (1994), pp. 157–163. DOI: 10.1016/b978-1-55860-335-6.50027-1.

[32] Maja J Mataric. “Reward functions for accelerated learning”. In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 181–189.

[33] Maja J Matarić. “Learning in multi-robot systems”. In: *International Joint Conference on Artificial Intelligence*. Springer. 1995, pp. 152–163.

[34] Maja J. Matarić. “Reinforcement Learning in the Multi-Robot Domain”. In: *Robot Colonies* (1997), pp. 73–83. DOI: 10.1007/978-1-4757-6451-2_4.

[35] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. “The numerics of gans”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1825–1835.

[36] Andrew W. Moore and Christopher G. Atkeson. “Prioritized sweeping: Reinforcement learning with less data and less time”. In: *Machine Learning* 13.1 (1993), pp. 103–130. DOI: 10.1007/bf00993104.

[37] John Nash. “Non-cooperative games”. In: *Annals of mathematics* (1951), pp. 286–295.

[38] John F Nash et al. “Equilibrium points in n-person games”. In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.

[39] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, p. 2.

[40] Jing Peng and Ronald J. Williams. “Incremental multi-step Q-learning”. In: *Machine Learning* 22.1-3 (1996), pp. 283–290. DOI: 10.1007/bf00114731.

[41] Jan Peters and Stefan Schaal. “Policy Gradient Methods for Robotics”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006). DOI: 10.1109/iros.2006.282564.

[42] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural Networks* 21.4 (2008), pp. 682–697. DOI: 10.1016/j.neunet.2008.02.003.

[43] Matteo Pirotta and Marcello Restelli. “Inverse reinforcement learning through policy gradient minimization”. In: *AAAI Conference on Artificial Intelligence*. AAAI Press. 2016, pp. 1993–1999.

[44] Matteo Pirotta and Marcello Restelli. “Inverse reinforcement learning through policy gradient minimization”. In: *AAAI Conference on Artificial Intelligence*. AAAI Press. 2016, pp. 1993–1999.

[45] Dean A Pomerleau. "Efficient training of artificial neural networks for autonomous navigation". In: *Neural computation* 3.1 (1991), pp. 88–97.

[46] Rob Powers and Yoav Shoham. "New criteria and a new algorithm for learning in multi-agent systems". In: *Advances in neural information processing systems*. 2005, pp. 1089–1096.

[47] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. 1994. ISBN: 9780471619772.

[48] Giorgia Ramponi, Gianluca Drappo, and Marcello Restelli. "Inverse Reinforcement Learning from a Gradient-based Learner". In: *arXiv preprint arXiv:2007.07812* (2020).

[49] Giorgia Ramponi and Marcello Restelli. "Newton-based Policy Optimization for Games". In: *arXiv preprint arXiv:2007.07804* (2020).

[50] Giorgia Ramponi et al. "Truly Batch Model-Free Inverse Reinforcement Learning about Multiple Intentions". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2359–2369.

[51] Alfréd Rényi et al. "On measures of entropy and information". In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California. 1961.

[52] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[53] Claude Sammut et al. "Learning to fly". In: *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 385–393.

[54] Joe Saunders, Chrystopher L Nehaniv, and Kerstin Dautenhahn. "Teaching robots by moulding behavior and scaffolding the environment". In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. 2006, pp. 118–125.

[55] Florian Schäfer and Anima Anandkumar. "Competitive gradient descent". In: *Advances in Neural Information Processing Systems*. 2019, pp. 7625–7635.

[56] Sandip Sen, Mahendra Sekaran, and John Hale. "Learning to Coordinate without Sharing Information". In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*. AAAI '94. Seattle, Washington, USA: American Association for Artificial Intelligence, 1994, pp. 426–431. ISBN: 0262611023.

[57] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[58] Satinder P Singh, Michael J Kearns, and Yishay Mansour. "Nash Convergence of Gradient Dynamics in General-Sum Games." In: *UAI*. 2000, pp. 541–548.

[59] Richard S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *Machine Learning Proceedings 1990* (1990), pp. 216–224. DOI: 10.1016/b978-1-55860-141-3.50030-4.

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[61] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000, pp. 1057–1063. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf.

[62] Umar Syed and Robert E Schapire. "A game-theoretic approach to apprenticeship learning". In: *Advances in neural information processing systems*. 2008, pp. 1449–1456.

[63] Gerald Tesauro. "Extending Q-learning to general adaptive multi-agent systems". In: *Advances in neural information processing systems*. 2004, pp. 871–878.

[64] Paul Tseng. "Convergence of a block coordinate descent method for nondifferentiable minimization". In: *Journal of optimization theory and applications* 109.3 (2001), pp. 475–494.

[65] John N. Tsitsiklis. "Asynchronous stochastic approximation and Q-learning". In: *Machine Learning* 16.3 (1994), pp. 185–202. DOI: 10.1007/bf00993306.

[66] Karl Tuyls, Pieter JT Hoen, and Bram Vanschoenwinkel. "An evolutionary dynamical analysis of multi-agent learning in iterated games". In: *Autonomous Agents and Multi-Agent Systems* 12.1 (2006), pp. 115–153.

[67] Eric Veach and Leonidas J Guibas. "Optimally combining sampling techniques for Monte Carlo rendering". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 1995, pp. 419–428.

[68] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292. DOI: 10.1007/bf00992698.

[69] Michael Weinberg and Jeffrey S Rosenschein. "Best-response multiagent learning in non-stationary environments". In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 2004, pp. 506–513.

[70] Samuel S Wilks. "The large-sample distribution of the likelihood ratio for testing composite hypotheses". In: *The annals of mathematical statistics* 9.1 (1938), pp. 60–62.

[71] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3-4 (1992), pp. 229–256. DOI: 10.1007/bf00992696.

[72] Chongjie Zhang and Victor R Lesser. "Multi-Agent Learning with Policy Prediction." In: *AAAI*. Vol. 3. 2010, p. 8.

[73] Martin Zinkevich. "Online convex programming and generalized infinitesimal gradient ascent". In: *Proceedings of the 20th international conference on machine learning (icml-03)*. 2003, pp. 928–936.

Appendix A

Gridworld: Soccer

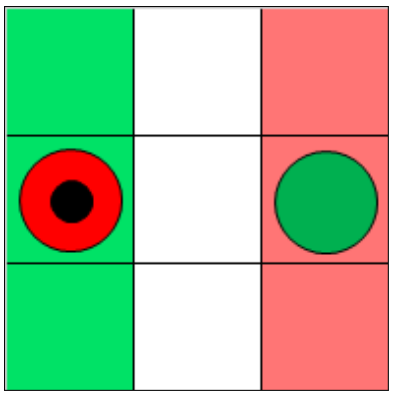


Figure A.1: Gridworld Soccer.

In Gridworld Soccer, two players are in a 3x3 grid in which the two lateral columns are the goals of the agents and each of them starts in the middle raw of the opposite column with respect to their goal. A visual representation can be seen in Figure A.1 where red represents agent 1 and green agent 2.
In each moment of the simulation each agent can take one of five actions: go north, go east, go south, go west or stay still. A ball is assigned randomly at the beginning of each episode to one of the players, represented by a black dot at the center of the agent. To obtain the reward associated with the goal, the agent have to move in one of the goal cells while possessing the ball. The interest of this environment is represented by the cyclical behavior that gradient descent algorithms generate like Matching Pennies, even if in a more complex environment, while algorithms like IGA-PP still converge to a Nash Equilibrium. This behavior happens when the agent with the ball

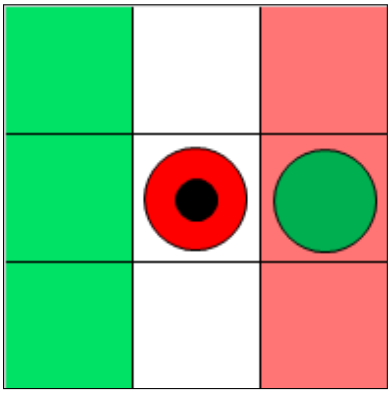


Figure A.2: Gridworld Soccer cyclical behavior point.

is in the middle cell and the agent without the ball is in its starting position, as shown in Figure A.2. In Figures A.3 and A.4, the probabilities of taking actions up, down and staying still are plotted for both the agents, using a continuous line for agent 1 and a dotted line for agent 2. It can be seen how gradient descent algorithms like GPOMDP diverge while IGA-PP converges to the Nash Equilibrium.

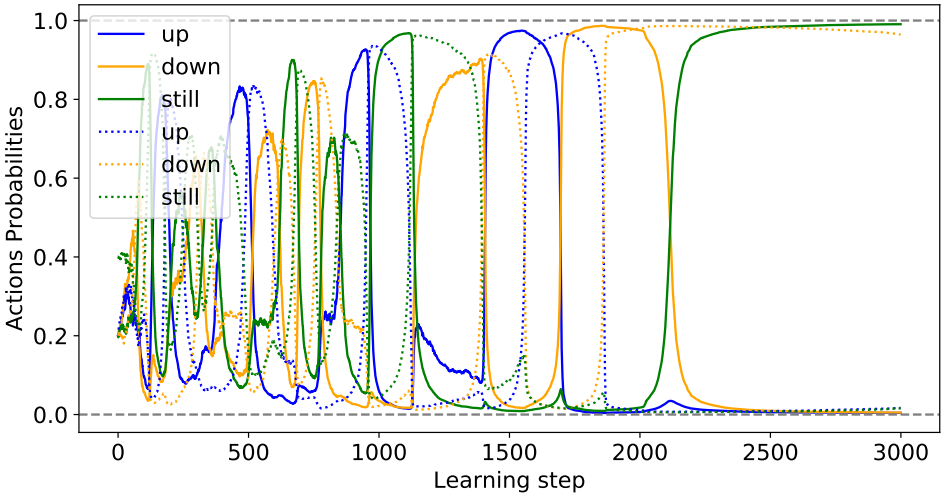


Figure A.3: Probabilities of the agents using GPOMDP in Gridworld Soccer

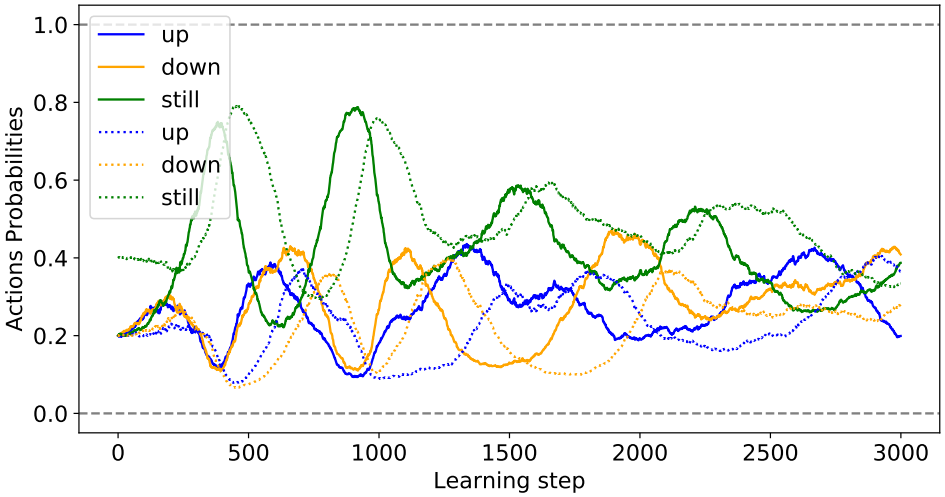


Figure A.4: Probabilities of the agents using IGA-PP in Gridworld Soccer

Appendix B

IRL in Bimatrix: LOLA

In this appendix we present the results in the same setting as in Section 4.2 with the single difference that the two agents instead of playing using IGA-PP they play following the update rule of LOLA.
As with IGA-PP, the results in Figure B.1 show that, using the omega estimation formula (3.3), the two agents do not converge and instead diverge due to the lack of the correct estimation of the other agent's $\boldsymbol{\omega}$ (see Figure B.1a and B.1c). Instead, using LOLA omega estimation formula (3.5), the two agents learn the correct opponent's reward function and converge to the Nash Equilibrium (see Figure B.1b and B.1d).

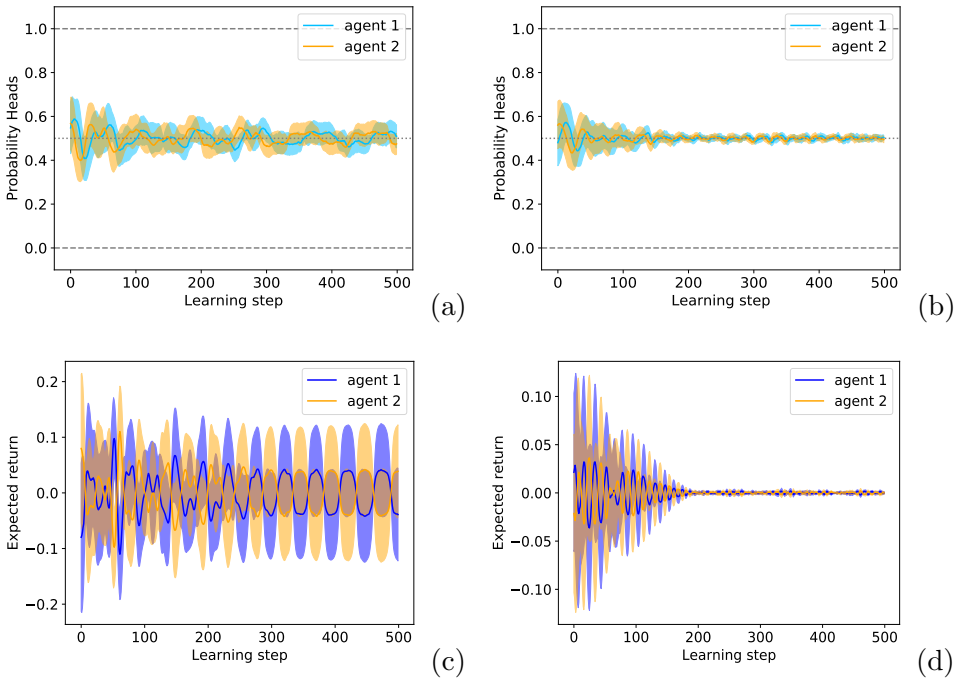


Figure B.1: Matching Pennies results. Probability of choosing Heads with IRL on GPOMDP (a) and on LOLA (b). Expected return with IRL on GPOMDP (c) and on LOLA (d).