**ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA**

Department of Computer Science and Engineering

ARTIFICIAL INTELLIGENCE

MASTER THESIS

NORM INFERENCE IN SOCIAL DILEMMAS: AN INVERSE REINFORCEMENT LEARNING APPROACH

Supervisor

Prof. Mirco Musolesi

Candidate

Veronica Biancacci

Accademic Year 2021/2022

For in this age of machine might

We must ensure we get it right

And build with care, and build with grace

An AI that's ethical and safe.

- ChatGPT

Abstract

Cooperation is an important tool for humans, crucial to reach optimal and ethical behaviour in many contexts. Multi-agent Reinforcement Learning techniques are an excellent instrument for studying the emerging cooperative behaviour of AI agents in different environments that can be simulated through games, which can be considered simplifications of the real world. Some of the most studied cases are Social Dilemmas, such as the Common Pool Resource Problem, where the cooperation or defection of the agents is crucial to the outcomes of the collective and the individuals.

The latest research has led to a good performance of the cooperation between AI agents. However, another critical characteristic agents need is Norm Inference, which is the ability to identify and understand the social norms that govern behaviour in a society. It is a serious aspect that must be considered when designing them since artificial learning agents will likely be embodied in our future society and will need to interact with both humans and non-human agents.

In this dissertation, an Inverse Reinforcement Learning (IRL) approach is used on the problem of Norm Inference in a Common Pool Resource problem, where the norm of private areas has been established. It is shown how it is possible to recover the expert policy that follows the norm through IRL and how the recovered reward function can be informative about the norm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With *Cooperative AI*, we refer to a class of problems that aims to solve cooperation in Artificial Intelligence systems. Cooperation is an essential tool for humans, necessary to reach optimal and ethical behaviour in many contexts. It is a well-known area, studied for decades, especially in game theory and economics. Lately, it also became a relevant subject of study in the Multi-Agent Reinforcement Learning Systems field since it is a good instrument for simulating games, which can be considered simplifications of the real world, and studying the emerging behaviour of AI agents in different environments. While traditional AI approaches often involve a single agent working in isolation to achieve a goal, Cooperative AI involves multiple agents working together to achieve a common or individual goal. Cooperation can help improve the AI system's overall performance and effectiveness.

Some of the most studied cases in Cooperative AI are *Social Dilemmas*, where the cooperation or defection of the agents is crucial to the outcomes of the collective and the individuals. In these settings, agents have the opportunity to improve their joint welfare by cooperating but are not easily able to do so.

Cooperative AI is an important field to investigate; it allows us to better understand cooperative interactions in different settings that resemble real-world situations and discover new solutions to present relevant problems. It is also an instrument that will allow us to achieve more moral and ethical AI agents. In the near future, artificial learning agents will likely be embodied in our society and will need to interact with both human and non-human agents. Designing agents capable of cooperating well with humans is a fundamental characteristic. A key aspect of these systems will be their ability to align with human values and norms.

Norm Inference refers to an AI system's ability to identify and understand the social norms that govern behaviour in a society. Norms are an essential part of human social interactions, and they help to ensure that people behave in ways that are accepted and expected by others in their community. AI systems that are able to detect norms can use this information to interact more effectively with other human or non-human agents in a given social context.

Social Norms are standard solutions used in human society to solve Social Dilemmas, such as the problem of Common Pool Resource. The problem of *Common Pool Resource* is a situation in which a group of individuals share a limited resource, such as a natural resource or environmental goods. It can be found in real-world examples, such as fishery stocks, water-management issues, and clean air rights. This can be difficult for the agents of this system to avoid over-exploiting the resource because of the greed to take as much of it as possible while it is still available and the fear that others will take all the resources. It can lead to resource depletion and, eventually, to conflicts among the individuals who share it. Cooperation and joint welfare are complex outcomes to achieve; for this reason, the Common Pool Resource problem is often referred to as the *Tragedy of the Commons*, from the homonym essay published in Science in 1968 by ecologist Garrett Hardinref [1]. However, human society has found cooperative solutions, often through social norms and rules. One solution to this problem is the creation of private areas and property rights, which can help ensure that the resource is managed sustainably.

In Cooperative AI, the main problem investigated is how to make agents more cooperative and less selfish. Many studies aim to find ways to achieve cooperation by training a population of agents in Social Dilemma problems so that the emerging behaviour of the society follows some norms that enable them to behave cooperatively. Lately, another aspect of the problem has also been investigated by researchers.

What happens when an external agent needs to enter a society where a norm is already established? Typically, the agent is trained to align with the values of other agents before it enters the new system, otherwise, its non-cooperative behaviour could threaten the balance and cooperation achieved in the group. The external agent, which has to be trained to gain knowledge about the system's norms, is not aware of the reward function needed to learn cooperative behaviour. To tackle this issue, we can use *Inverse Reinforcement Learning*, a method capable of learning the reward function of a given task through expert demonstrations.

The use of Inverse Reinforcement Learning to solve Norm Inference is an open research problem discussed by the community in recent years, but the only work, to our knowledge, and state of the art, to research and implement the problem is [18], which proposes a modified Inverse Reinforcement Learning algorithm to learn context-sensitive norms, called Context-Sensitive Norm Inverse Reinforcement Learning (CNIRL). It depends on the assumption that observations and contexts are separated in the environment, as we explain in Section 2.4.2. In our project, we instead rely on the Adversarial Inverse Reinforcement Learning algorithm to learn the context-sensitive norm from observations, relaxing the assumption, and we apply the problem of Norm Inference to a Social Dilemma, differently from [18].

In this dissertation, we investigate Norm Inference in a Common Pool Resource problem through Inverse Reinforcement Learning by learning a reward function and an optimal policy from expert demonstrations. The norm established is the one of private areas which enable the system's agents to learn cooperative behaviour. We further analyse the retrieved reward function to investigate which information about the norm it encodes and how it could be extracted and used on a random policy to reduce the number of norm violations.

Chapter 2

Background

This section reviews the underlying concepts and techniques upon which we build the approach presented in the next chapter.

2.1 Social Dilemmas

A *Social Dilemma* is a situation in which individual interests conflict with the collective good. In other words, it is a scenario where an individual's most beneficial choice differs from the most beneficial one for the group. These circumstances often arise when individuals must decide whether to act in their own self-interest (*defection*) or in the group's interest (*cooperation*).

One of the most famous Social Dilemmas is the *Prisoner's Dilemma* (PD), where two prisoners must decide whether to betray each other or cooperate. If they both defect, they receive a worse outcome than if they had cooperated. However, if one betrays the other while the other remains loyal, the betrayer receives the best possible outcome, while the loyal prisoner receives the worst. It creates a dilemma because the most beneficial choice for an individual is to betray the other prisoner, but the most beneficial choice for both prisoners is to cooperate. In this problem, mutual defection is a Nash Equilibrium.

SD	**C**	**D**
C	R, R	S, T
D	T, S	P, P

Prisoners	**C**	**D**
C	$3, 3$	$0, 4$
D	$4, 0$	$1, 1$

Table 2.1: Left: Canonical Payoff Matrix for Social Dilemmas. Right: Payoff Matrix for Prisoner's Dilemma. The two possible actions are cooperation (C) and defection (D)

In game theory, the *Nash equilibrium* defines a set of strategies for each player in a game such that no player has the incentive to deviate from their strategy. In other words, no player can improve their outcome by switching to a different strategy, given the other players' strategies.

Social Dilemmas can be represented through matrix games. In *Matrix Game Social Dilemmas* (MGSD), the payoffs satisfy the following inequalities (this formulation from [2]):

1. R > P Mutual cooperation is preferred to mutual defection.
2. R > S Mutual cooperation is preferred to being exploited by a defector.
3. 2R > T +S Mutual cooperation is preferred to an equal probability of unilateral cooperation and defection.
4. either greed: T > R Exploiting a cooperator is preferred over mutual cooperation
   or fear: P > S Mutual defection is preferred over being exploited.

where R is the reward of mutual cooperation, P is the punishment arising from mutual defection, S (sucker) is the outcome obtained by cooperating against a defector, and T (temptation) is the the outcome obtained by defecting against a cooperator. Payoff matrices of the Prisoner's Dilemma are shown in table 2.1.

However, in real-life situations, Social Dilemmas are temporally extended and not one-shot games as in MGDS. For this reason, Sequential Social Dilemma models have been proposed [3].

In a *Sequential Social Dilemma* (SSD), the consequences of an individual's action do not depend only on their action at a current step but also on all the actions made previously by all the agents in the system.

A classic example of an SSD is the *Common Pool Resource* (CPR) problem. As already explained in chapter 1, if everyone uses the resource sparingly, it will be preserved and could be used for more time. However, if everyone acts in their self-interest and uses as much of the resource as possible, it will be depleted and no longer available for future use. Also in this problem, mutual defection is a Nash Equilibrium. Agents are prone to defect out of fear and greed. They are afraid of others defecting (taking all the resources before them) and want to maximise their own reward.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is an area of Machine Learning in which an agent learns, by trial and error, to reach a goal. The agent interacts with its environment by taking actions and receives feedback in the form of rewards and punishments. The feedback is used to correct its policy, which determines which action to take in each possible state. The goal of the agent is to find the optimal policy, which maximizes its expected cumulative reward R, defined as:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i \tag{1}$$

where r_i is the reward received at timestep i and γ is the discount rate, a parameter between 0 and 1, determining the importance of future rewards versus immediate rewards, used in continuing tasks.

Deep Reinforcement Learning includes all the RL algorithms that use Artificial Neural Networks to approximate the learnt policy or value function.

2.2.1 Markov Decision Process

RL is modelled as a Markov decision process. A *Markov decision process* (MDP) is a mathematical framework for modelling sequential decision-making problems under uncertainty. It is a discrete-time stochastic control process.

Formally, an MDP is defined as a tuple (S, A, P, R), where S is a finite set of states, A is a finite set of actions, P is a state transition function, R is a reward function, and γ is a discount factor. At each time step t, the MDP is in some state $s_t \in S$. The agent can choose an action $a_t \in A$, which will cause the MDP to transition to a new state $s_{t+1} \in S$ according to the state transition function P. The agent receives a reward $r_t \in R$ for its action, according to the reward function R. The agent's goal is to maximize its expected cumulative reward over time.

2.2.2 Sequential Social Dilemmas

Sequential Social Dilemmas (SSD) are defined as general-sum (simultaneous move) Markov games with each agent having a partial observation of the environment.

Formally, SSD is a tuple (M, Π_C, Π_D) where Π_C and Π_D are disjoint sets of policies that are said to implement cooperation and defection, respectively. M is a Markov game with state space S. Let the empirical payoff matrix $(R(s), P(s), S(s), T(s))$ be induced by policies $(\pi_C \in \Pi_C; \pi_D \in \Pi_D)$. A Markov game is an SSD when there exist states $s \in S$ for which the induced empirical payoff matrix satisfies the social dilemma inequalities reported in Section 2.1. [3]

2.2.3 RL Algorithms

Reinforcement Learning algorithms differ in how they approach learning, the kinds of problems they can solve, and their strengths and weaknesses. There is no agreed-upon set of categories for all RL algorithms, but some common ways of categorizing them include:

- *Model-based* vs *model-free*: in model-based Reinforcement Learning, the algorithm learns a model of the environment and uses it to plan its actions. In contrast, model-free Reinforcement Learning algorithms learn directly from experience without building an explicit model of the environment.

- *On-policy* vs *off-policy*: in on-policy Reinforcement Learning, the algorithm learns from the actions it takes in the environment. In off-policy Reinforcement Learning, the algorithm can learn from pre-recorded experiences independently of the agent's current actions in the environment.

- *Value-based* vs *policy-based*: In value-based Reinforcement Learning, the algorithm learns a function that estimates the long-term reward for each possible state-action pair. The optimal policy will be the one that chooses, at each timestep, the action with the highest value. In policy-based Reinforcement Learning, the algorithm directly learns a policy that outputs a probability distribution over the actions. Again, the optimal policy is the one that chooses the action with the highest probability.

We now introduce some RL algorithms that we will meet again in the next chapter.

2.2.4 Actor-Critic

Actor-Critic is a model-free, on-policy, policy-based RL algorithm.

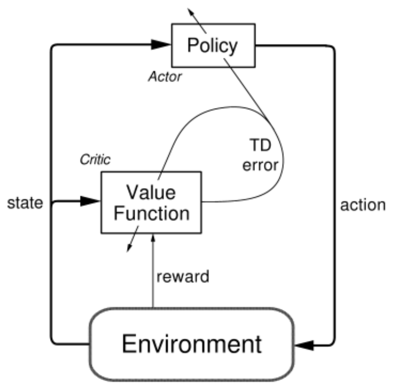


Figure 2.1: Actor-Critic structure.

Actor-Critic algorithms use two separate neural networks, the actor and the critic, to learn and make decisions. The actor is a policy network and learns a probability distribution over the actions for a given state, so it is responsible for learning and choosing actions. On the other hand, the critic is a value network, meaning it outputs an estimate of the expected return for a given state and action. It is responsible for evaluating the actions chosen by the actor and providing feedback. The actor then uses this feedback to adjust the probabilities in the distribution and improve the quality of its decisions.

The two networks are trained together using gradient descent. The critic network is trained to minimize the mean-squared error between its predictions and the target values, which are calculated using the Bellman equation. Meanwhile, the actor network is trained to maximize the expected return.

The *Bellman equation* describes the relationship between a state's value and its successor states. Formally, the Bellman equation for the action-value function Q can be written as follows:

$$Q(s,a) = E[R(s,a)] + E[max(Q(s',a'))] \tag{2}$$

where s is the current state, a is the current action, $R(s,a)$ is the expected reward for taking action a in state s, and s' is the next state. The equation states that the value of the current state and current action is equal to the sum of the expected reward for taking that action in that state and the expected maximum future return achievable from the next state and next action.

The Bellman equation is essential because it provides a way to update the action-value function based on the current estimate of the expected rewards and values of the next state. It allows Reinforcement Learning algorithms to iteratively improve their estimates of the action-value function, which will be used by the actor to find an optimal policy for taking actions in the environment.

It is an on-policy algorithm since it does an update at each step, using subsequent information (state, action, reward) received from the step just performed.

2.2.5 Soft Actor-Critic

Soft Actor-Critic (SAC) [4] is a model-free, off-policy, policy-based RL algorithm. SAC combines ideas from Actor-Critic and Maximum Entropy Reinforcement Learning.

The algorithm's name comes from the concept of "soft" policies. In a traditional Actor-Critic algorithm, the actor is typically trained to select actions according to a deterministic policy. On the other hand, a soft Actor-Critic algorithm is trained to select actions according to a stochastic (i.e., "soft") policy, allowing it to explore the environment and learn more effectively. It can improve performance, especially in complex environments with large action spaces.

One critical method to encourage exploration is the use of the *Maximum Entropy* principle. This principle states that, when making decisions, an agent should maximize the entropy of its future actions in order to explore more of the state space and potentially discover better options. In the case of SAC, the agent will try to find a balance between exploiting known good actions and exploring new ones by maximizing both the expected cumulative reward R and the entropy of the policy.

The objective of the actor becomes:

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\sim p_\pi}[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] : \tag{3}$$

SAC is an off-policy algorithm, meaning it can learn from experience collected using a different policy than the one being learned. In practice, SAC uses a replay buffer to store past experiences, which are then sampled to train the networks. It allows the algorithm to learn from a diverse set of experiences and to improve sample efficiency.

2.3 Inverse Reinforcement Learning

The problem of *Inverse Reinforcement Learning* (IRL) in a Markov Decision Process consists of extracting the reward function given trajectories of an expert policy that is assumed to be optimal. The final goal is usually to use the extract function reward to train a new policy that behaves as the optimal one. For this reason, IRL is considered an approach of Imitation Learning, which is the more general problem of learning an optimal policy by imitating an expert behaviour, given a set of demonstrations.

More formally, we are given a set of expert demonstrations $D : \{\tau\}\tilde{}\pi^*$, from an unknown distribution, each in the form of $\tau = \{s_1, a_1, ..., s_T, a_T\}$, where s_t and a_t are the state and action at timestep t. We want to recover $c_\theta(\tau) = \sum c_\theta(s_t, a_t)$, an unknown reward function parameterized by θ, in a way that the agent can learn a near-optimal policy using the learned reward function.

It is considered to be an underdefined problem since 1) different optimal policies can explain a set of demonstrations, and 2) different reward functions can induce the optimal behaviour. [5]

In the next sections we introduce three algorithms: the Maximum Entropy IRL, the Generative Adversarial Imitation Learning, and the

Adversarial Inverse Reinforcement Learning.

2.3.1 Maximum Entropy IRL

The most popular IRL algorithm is the *Maximum Entropy Inverse Reinforcement Learning* (MaxEnt IRL) [6], which deals with the first ambiguity introduced in the previous section, by using a probabilistic behaviour model.

The MaxEnt formulation states that the probability of the trajectories under the expert is exponential in the reward:

$$p(\tau) = \frac{1}{Z} exp(c_\theta(\tau)) \tag{4}$$

It means that trajectories with maximal reward are exponentially most likely to be sampled by the expert under some reward function. To infer the reward function, we can then maximize the log-likelihood of the set of demonstrations with respect to the parameters θ of the reward function.

$$L(\theta) = \sum_{\tau \in D} log(p_{c_\theta}(\tau)) \tag{5}$$

$$= \sum_{\tau \in D} log(\frac{1}{Z} exp(c_\theta(\tau))) = \sum_{\tau \in D} c_\theta(\tau) - Mlog(Z) \tag{6}$$

with the partition function Z being $Z = \int exp(c_\theta(\tau))\, dx$. Then, the final objective to optimize is:

$$L(\theta) = \sum_{\tau \in D} c_\theta(\tau) - Mlog(\sum_{\tau} exp(c_\theta(\tau))) \tag{7}$$

We can use gradient descent to optimize it by computing the following gradient:

$$\nabla L(\theta) = \sum_{\tau \in D} \frac{dc_\theta(\tau)}{d\theta} - M \frac{1}{\sum_\tau exp(c_\theta(\tau))} \sum_{\tau} exp(c_\theta(\tau)) \frac{dc_\theta(\tau)}{d\theta} \tag{8}$$

The second term of the gradient can be simplified to avoid enumerating all possible trajectories, becoming:

$$= \sum_{\tau} p(\tau|\theta) \frac{dc_\theta(\tau)}{d\theta} = \sum_{s} p(s|\theta) \frac{dc_\theta(s)}{d\theta} \tag{9}$$

where $p(s|\theta)$ is the state visitation frequency and can be computed using a dynamic programming algorithm.

The MaxEnt IRL algorithm's main drawbacks are the assumption of known dynamics (the transition function) and the need to compute the state visitation frequencies. They could be approximated through interactions with the MDP, but using this algorithm in high dimensional state space is not feasible. Moreover, it requires in input hand-engineering features instead of row inputs and the recovered reward function does not generalize well, is not robust to environment's changes, and does not manage to infer the true intentions of the expert.

2.3.2 Generative Adversarial Imitation Learning

Let us now introduce a different algorithm that manages to recover the expert policy from the raw expert trajectories (state-action pairs, removing the need for hand-engineering features) and works on unknown dynamics.

Generative Adversarial Imitation Learning (GAIL) [7] is an Imitation Learning Algorithm based on adversarial learning and GAN's architecture [8]. Adversarial learning is a technique in Machine Learning where two models, the Generator G and the Discriminator D, are trained simultaneously. The Generator creates samples, in this case trajectories, that are intended to be similar to the expert's ones, while the Discriminator attempts to distinguish the generated samples from the expert trajectories. The two models are trained in an adversarial manner, with the Generator trying to produce samples that can fool the Discriminator and the Discriminator trying to correctly identify the generated samples. In

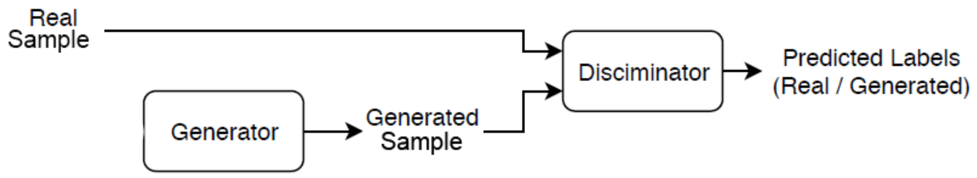


Figure 2.2: GAN structure. In our problem, the samples are the trajectories or the state-action pairs.

other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$min_G max_D V(D, G) = E_{x \sim p_data(x)}[log D(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{10}$$

Even if the Generator manages to recover the optimal policy, the Discriminator does not directly aim to recover the reward function since it outputs a value between 0 and 1 that classifies the input (state-action pair) as part of the expert trajectories distribution or not. It is not robust to the environment's changes and is unsuitable to use as a reward function. For this reason, GAIL is not considered to be an IRL algorithm.

2.3.3 Adversarial Inverse Reinforcement Learning

Adversarial Inverse Reinforcement Learning (AIRL) algorithm, differently from GAIL, is an IRL algorithm introduced in 2018 [9]. While the Generator learns the expert policy, the Discriminator is trained to learn the reward function.

One of the first works to show a direct connection between GANs and IRL proposes to cast the maximum likelihood optimization of the Maximum Entropy IRL in (5) as a GAN [10], as follows:

$$D_\theta(\tau) = \frac{exp\{f_\theta(\tau)\}}{exp\{f_\theta(\tau)\} + \pi(\tau)} \tag{11}$$

Moreover, AIRL switches from a trajectory-centric formulation to a state-action pair one to solve the problem of high variance estimates of the former formulations.

Lastly, AIRL Discriminator is modified to recover a generalizable and robust reward function by disentangling it from the dynamics. The final Discriminator takes the following form:

$$D_{\theta,\phi}(s,a,s') = \frac{exp\{f_{\theta,\phi}(s,a,s')\}}{exp\{f_{\theta,\phi}(s,a,s')\} + \pi(a|s)} \tag{12}$$

with $f_{\theta,\phi}$ composed by a reward approximator g_θ and a shaping term h_ϕ as follows:

$$f_{\theta,\phi}(s,a,s') = g_\theta(s,a) - \gamma h_\phi(s') + h_\phi(s) \tag{13}$$

The Discriminator objective is the same used in the traditional GAN loss, while the policy can be updated using any policy optimization models that aim to maximize the expected cumulative reward, where the reward function is given by:

$$r(s,a,s') = log(D_{\theta,\phi}(s,a,s')) - log(1 - D_{\theta,\phi}(s,a,s')) \tag{14}$$

2.4 Norm Inference

2.4.1 Overview

Social Norms can be defined as the expectations, rules, or guidelines that govern the behaviour of individuals in a particular society or group. These norms are typically based on the society's shared values, beliefs, and customs, and they regulate individuals' behaviour to maintain social order and achieve cooperation. They can be explicit, such as laws and regulations, or implicit, such as unwritten rules of behaviour that are expected and accepted within a particular society. These norms are often enforced through social pressure, exclusion or some other kind of punishment. Overall, social norms play a crucial role in shaping the behaviour of individuals within a society.

Recognizing the norms that govern a society is a key ability an agent must master, especially if it has to interact in a human society, where norms are intrinsic in everyday activities.

The process of *Norm Inference* (also referred to in the literature as Norm Detection, Norm Recognition or Norm Identification) enables the agent to learn what is allowed, what is mandatory, and what is considered prohibited within a society. As the agent joins and leaves different open systems, the capability of recognizing the new norms that govern the society helps it derive new ways of achieving its goals by behaving in the way the other agents of the society expect it to do.

Norms are often classified by the Normative Multi-Agent systems community as follows:

- *Obligation norms*: they require the agent to perform a specific action or behaviour.
- *Permission norms*: they allow the agent to perform a specific action or behaviour. The presence of a permission norm usually corresponds to the absence of norms, as individuals can act freely.
- *Prohibition norms*: they forbid the agent from performing a specific action or behaviour.

Another critical aspect to consider is the Context-Sensitive characteristic of norms, which indicates that different contexts activate different norms. Usually, the context is intrinsic in the observation of the current timestep, so the agent should be able to detect, from observations, if and which type of norm is activated and choose the optimal behaviour to reach his goal without violating the norms.

2.4.2 Previous Work

A drawback found in previous work on Norm Inference is that some assumptions are usually presupposed to hold.

Let us examine which are the most frequent assumptions.

- The system must be closed: in the early research stage, some works suggested approaches that work only in closed systems. If the system is closed, it is easier for the agents to communicate in some ways and inform others of the norms or disclose other information. In open multi-agent systems, agents may have different internal architectures and concepts of norms, or they may not have a concept of norms at all.

- There must be deviant agents: according to Therborn [12], the detection of deviant agents is critical to the norms' learning. Many other works [11][12][14][18] assume the presence in the system of some agents who violate the norms. These *signalling actions* [17] are special events that help the agent identify behaviours that are discouraged (or encouraged) by the society. This assumption does not always hold, such as in optimal societies where all its agents follow the norms. In this case, it is not possible to recognise special events just by observing the agents' behaviour.

- The reward function must be visible: many works, such as the ones that assume the presence of deviant agents, also assume that the reward function is visible. Even if *signalling actions* exist in the system, observing the rewards or punishments that follow them would still be necessary to detect them. If the rewards cannot be accessed, it would be difficult for the agent to distinguish good from bad actions.

- The norms have structured, domain-specific representation and/or can be enumerated [15][16]: many works of Norm Inference of the past years had as their final goal the creation of an agent capable of detecting norms and storing them in its belief, by encoding them in structured representation. This representation can be domain-specific or represented in some formal/logical language. It can be limiting for different reasons; it does not generalise for different domains, and some norms may be difficult or impossible to express in formal languages or enumerate. It could be computationally expensive and may not scale well to large systems. Moreover, learning norms from observations in formal language is not always possible since it requires the agent to structure the information in a fixed representation that it may not be aware of.

- The context is already known: paper [18] assumes the context to be already known and separated from the observation. It would be a very convenient feature, but it is not likely to be present in the environment, and this separation could be challenging to accomplish.

The use of Inverse Reinforcement Learning to learn norms is still an open problem researched by the community. It has been discussed in recent years, but few works actually use it. As mentioned, [18] proposes a modified IRL algorithm to learn context-sensitive norms, called Context-Sensitive Norm Inverse Reinforcement Learning (CNIRL). As we explained, it assumes to have observations and contexts already separated. Moreover, its goal is to learn a policy which behaves following the norms without analysing the reward function to investigate the presence of possible information about the norms.

Chapter 3

Approach

3.1 Problem Setting

Let us assume an intelligent learning agent, referred to as *external agent*, wants to enter an open multi-agent system. The agents in this system could have different internal structures and concepts of norm from the external agent or among themselves, and they follow norms that the external agent is unaware of. We assume all the agents in the system to behave optimally, although the presence of a small percentage of deviant agents should not affect much the performance of the approach proposed in the next section.

The reward function is not observable, so the external agent can not identify *signalling actions* in case they are present. Instead, it can observe the interactions between the agents of the system. He can only access the *expert* trajectories (state-action pairs) and has no real concept of context or norm in his current architecture.

The problem investigated in this section is a Common Pool Resource appropriation problem and is represented through Sequential Social Dilemmas. The agents' goal is to gather resources from the environment. As we have already explained, without any norms that regulate the agents' behaviour, the problem will become a Tragedy of the Commons, with all the resources quickly depleted.

A norm capable of avoiding this outcome is the establishment of private areas. In this way, each agent will be in control of all the resources of the private area assigned to it. Without the fear of others taking the resources before it, it will learn to gather its resources in a renewable way. This norm is classified as a prohibition norm since it reduces the freedom of the system's agents, allowing them to take resources just from their own area.

With *renewable* or *sustainable* policy, we refer to a policy which menage to gather the resources until the end of the episode without depleting them all.

3.2 Proposed Approach

The proposed approach consists of performing Norm Inference in Social Dilemmas through Inverse Reinforcement Learning to obtain a policy that behaves according to the norm and a reward function that encodes information about the norm in the system and could help explain it.

More in detail, this project is organised as follows:

- Two initial policies are trained: one of the external agent, the other of the expert.
- An IRL algorithm is trained on expert trajectories, and the expert reward function and policy are obtained.
- The obtained reward function is compared with the external agent's one to extract information about the norm of the system.

3.2.1 Initial policies

The first part of the project consists of training the two initial policies in the same environment. The first policy corresponds to the external agent's one, which gathers resources without following any norm. The second corresponds to the expert policy, which instead follows the system's norm.

The algorithm used in this section is a Soft Actor-Critic RL algorithm, which manages, as explained in Section 2.2.5, to retrieve a soft policy. Choosing a soft policy can be advantageous in this environment since exploration could be critical in different situations, for example, when the agent ends up in empty areas, far away from the resources. Using a stochastic policy can help the agent avoid ending up in loops, going back and forth in the same area. It can also benefit the learning of the expert policy in the next step of the proposed approach since a soft policy can generate more than one optimal expert trajectory, which will help with generalization.

3.2.2 IRL Algorithm

The second part of the project consists of training the IRL algorithm to recover the expert reward function and policy.

The IRL algorithm chosen to infer the behaviour of the expert is Adversarial Inverse Reinforcement Learning. As explained in Section 2.3.3, it takes in input raw trajectories, made of a sequence of state-action pairs, and produces a policy and a robust reward function in output. Again, the policy optimization algorithm used to train the Generator network is the Soft Actor-Critic algorithm.

The popular Maximum Entropy IRL is unsuitable for our problem because of the unknown dynamics, the high dimensional state space and the weak reward function it retrieves. Similarly, the GAIL algorithm is unsuitable because, even if it can recover the expert policy, it cannot retrieve an actual reward function.

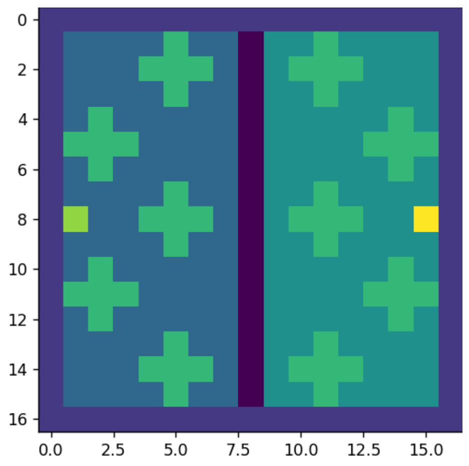


Figure 3.1: Environment's map.

Object	Value
Dividing wall	0
Framing wall	1
Left area	2
Right area	3
Apples	4
Left agent	5
Right agent	6

Table 3.1: Object's values in the environment.

3.2.3 Norm's Information in the Reward Function

In the last part of the project, the retrieved reward function is further analysed and compared with the one of the external agent to investigate how the new reward function can provide information about the system's norm.

With the extracted information, a supervised classifier is trained to recognise if a state-action pair will lead to a next state where the norm is violated.

3.3 Environment

The environment is inspired by the one presented in the paper [19] of Julien Perolat et al., where a similar game of Common Pool Resource appropriation is implemented as a Sequential Social Dilemma.

It is a partially-observed Markov game environment implemented as a 2D grid map. The experiments are implemented in a two-agents system.

The map consists of a 17x17 grid framed by a 1x1 deep wall. Apples and agents are represented by single pixels. Apples are structured in groups of 5, in a cross-like shape, referred to as trees. Two private areas, represented by different values, divide the map vertically into two equal regions, separated by another wall, as shown in Figure 3.1. The value of each object in the map is reported in the Table 3.1.

Apples can respawn with a probability proportionate to the density of nearby apples. It means that if all the apples on the map are gathered, they will not respawn anymore for the rest of the episode. The respawn probability p_t of an apple depends on the number n of present apples at the current timestep t in the 5x5 square centred around its location:

$$p_t(n) = \begin{cases} 0 & \text{if n} = 0 \\ 0.035 & \text{if } 0 < \text{n} < 3 \\ 0.065 & \text{if n} = 3 \\ 0.1 & \text{if n} > 3 \end{cases} \tag{15}$$

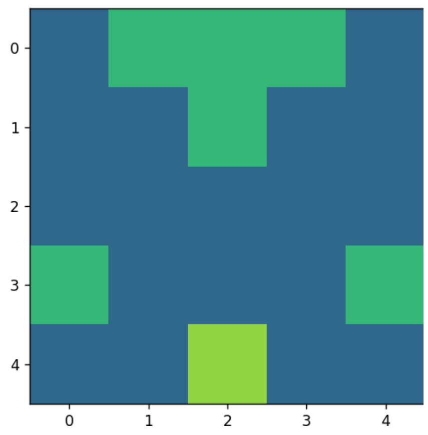


Figure 3.2: Agent's observation.

Agents observations $O(s, i) \in R^{5x5}$ of the current state s depend on the i^{th} agent's current position and orientation. The observation is a 5x5 window that extends 4 grid squares ahead and 2 grid squares from side to side. The agent's position in the observation is fixed at (4, 2). The observations of the two agents are equivalent, meaning that the right agent sees the values of the environment as if it was the left agent (the values of agents and private areas are exchanged in the right agent's observation). Observation's values are normalised between 0 and 1.

At each timestep, each agent can choose between 7 different actions: step forward, step right, step backwards, step left, rotate right, rotate left and stay still.

Chapter 4

Evaluation

4.1 Experiments

This section presents the conducted experiments along with the exact settings and parameters used for each of them.

4.1.1 Training of the Initial Policies

The preliminary step of our experiments consists in retrieving the initial policies of the external agent and the expert.

Both policies were trained for 400 episodes, each of 1000 steps. If all the apples are gathered before the 1000th step, the episode terminates earlier. The reward is 0 for each step and 1 if the agent picks a resource.

It was used a batch size of 32, a discount factor of 0.999, a replay buffer size of 10^5, 10^4 steps for the target network update, an Adam optimizer with a learning rate of 10^{-5} and a weight decay of 0.05 for each network. The entropy temperature hyperparameter was automatically tuned accordingly to [20].

Each model has 3 Convolutional layers of 3, 6, and 9 filters, each with a kernel size of 3, padding 'same', and ReLU activation function. A Flatten layer follows, with other 3 Fully-Connected layers of 64, 32, 18 neurons and the ReLU activation function. The last Fully-Connected layer outputs 7 values, one for each action.

The only difference between the two experiments is that in the first one, each agent can move in both areas and gather all the apples on the map, while in the second one, agents are permitted to move only in their private area.

4.1.2 Training of the IRL Algorithm

The main experiments of the project consist in retrieving the expert policy and reward function from expert trajectories using IRL.

In the original AIRL algorithm, as explained in Section 2.3.3, the model of the reward function and the one of the policy are trained simultaneously in an adversarial manner. Because the policy's training depends on the reward function's training, and we wished to evaluate and display the policy's performance during training on the final reward function, we replaced adversarial training with two consecutive training sessions.

Initially, the AIRL algorithm was used to train the reward function, where the expert trajectories served as real samples, and a random policy was used as the Generator. Then, the final policy is trained using the SAC algorithm on the retrieved reward function.

Two experiments were executed by training the model on two different expert policies. The first is a policy that follows the norm without behaving in a renewable way. It was trained to evaluate if the retrieved reward function can learn a sustainable policy even with expert trajectories that are not sustainable. The second experiment instead uses an expert policy that is already renewable. The policy used as the Generator for producing 'fake' trajectories and training the optimal policy with SAC is random.

A third experiment was run to test if using the external agent policy instead of a random one could improve the training of the models and the final performance. We experimented both, using this policy as the Generator to retrieve the reward function and as a starting policy in the training of the final one using the reward function retrieved in the first experiment.

The reward function is represented by two networks g and h, each with the same structure of the networks reported in Section 4.1.1 except for the number of neurons in the 3 Fully-Connected layers. The neurons' numbers, in this case, are 64, 32, and 18.

The reward function was trained for 15 episodes of 500 steps each. At each episode, 50 new samples were produced by the Generator and added to the old ones. It was used a batch size of 32 and two Adam optimizers with learning rates of 1e-3.

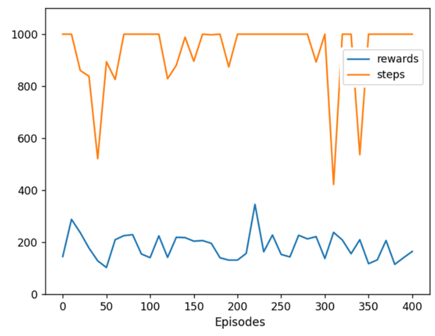
The policy was trained with the same hyperparameters reported in Section 4.1.1 but for fewer episodes, as it will be shown in the next section.

4.2 Results

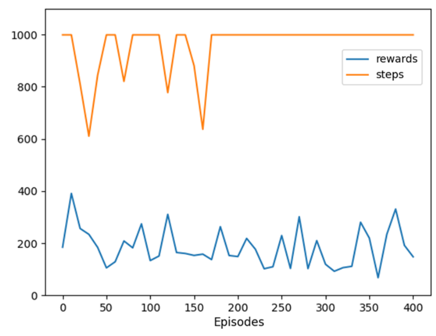
This section reports the obtained results of each experiment.

4.2.1 Evaluation of the Initial Policies

Figure 4.1 shows the results of the two initial policies during training. The *steps* function represents the number of steps the agent plays until the episode is terminated. If its value is 1000, the episode was played till the end, and the policy used is sustainable. If its value is less than 1000, the agent has depleted all the resources, and the episode was terminated earlier. In this case, the policy is not sustainable. The *rewards* function represents the number of resources collected during the episode by both agents.



(a) Policy without norm



(b) Policy with norm

Figure 4.1: The two plots report the total reward obtained and the episode's steps of the two initial policies. (a) represents the external agent policy which does not follow any norm. (b) represents the expert policy which follows the system's norm.

In both plots, when the policy is still random at training step 0, the reward amount is already quite high. It is because the random policy can not deplete all the resources and keep gathering them until the end of the episode, even without maximising the reward.

As soon as the training starts, again in both plots, the reward amount grows to drop shortly after in the following episodes. It can be explained by considering that the soft policy obtained at training step 10 needs more updates to learn to pick up apples perfectly but has already learned how to behave most of the time. The 'errors' it makes while it gathers apples give enough time for the resources to respawn before being all depleted. Soon after, the *rewards* and the *steps* function drop, indicating that the agent has ended up in the Tragedy of the Commons since it has become too fast in gathering the resources. Let us now explain the following steps.

Plot (a), corresponding to the system without the norm, shows how the agent does not manage to retrieve a sustainable policy and often gathers all the resources at once because of the fear that the other agent could take all the apples. Since the training is not stable, in some episodes, the resources are gathered in a renewable way. By analysing the trajectories of the agent during those training episodes, it is possible to observe that a drop in the performance has prevented the agent from depleting all the resources.

Plot (b) instead, corresponding to the system with the norm, shows how the agent learns, after some episodes, to gather resources in a renewable way as a result of the establishment of the norm of private areas and the absence of the fear of the other agent taking its resources.

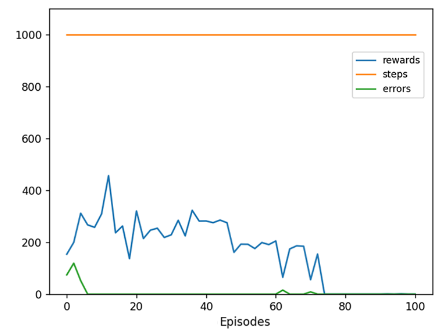
By analysing the trajectories, we discovered that the sustainable learned policy gathers resources mainly from the same 2 trees on one side of the map, sometimes picking up apples also from the other nearest trees, keeping the last tree on the other side of the map untouched most of the episode. It can be interpreted as an intentional behaviour to keep the policy sustainable by using the untouched trees as a recharge factor for the resources in the other trees. It is interesting to note how the achievement of this particular policy would not have been possible without the use of a soft policy.

4.2.2 Evaluation of the IRL Algorithm

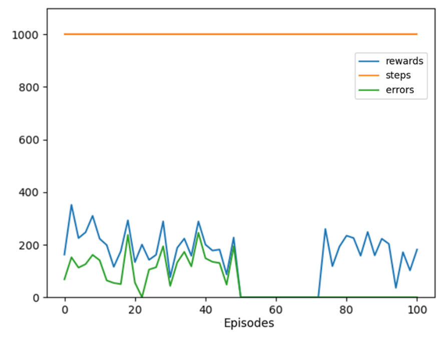
In this section we evaluate the ability of our approach to 1) learn the system's norm, 2) recover the expert policy, and 3) recover a sustainable policy.

Figure 4.2 reports the results of the first two experiments in terms of the total number of apples gathered (rewards), the number of norm violations (errors), namely the number of apples gathered in the wrong private area, and the episode's steps (steps). The first value is used to evaluate if the policy is maximizing its objective, the second to evaluate if it learns the norm, and the third to evaluate if it is a sustainable policy. An analysis of the trajectories is necessary to evaluate if the agent learns the exact same behaviour of the expert.

Plot (a) shows the results of the policy trained on the non-sustainable



(a) Policy on expert non-sustainable demonstrations



(b) Policy on expert sustainable demonstrations

Figure 4.2: The two plots report the total reward obtained, the episode's steps, and the number of violations of the policies trained on expert demonstrations with AIRL. (a) is trained on non-sustainable expert demonstrations, (b) is trained on sustainable expert demonstrations.

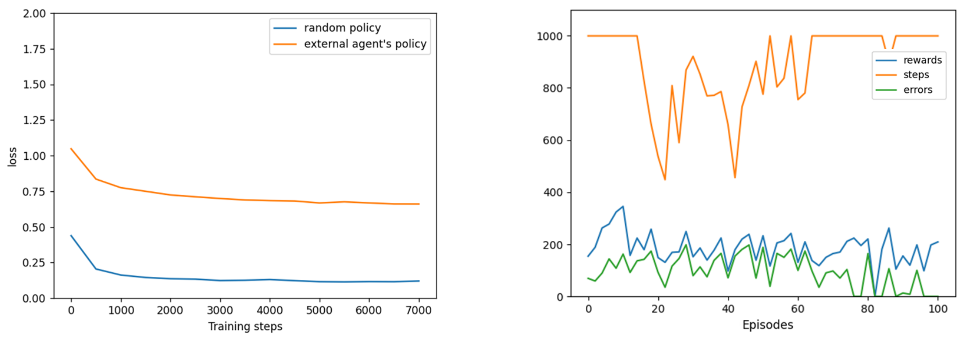
expert policy, while plot (b) shows the policy trained on the sustainable one.

It can be observed how the error function in both plots drops to 0, while the reward function, even if more unstable, behaves similarly to the previously explained ones, indicating that the agent has learned the norm. Further considerations on this Figure can be found in the discussion in Section 4.3.1.

The results of the last experiment, which uses the external agent policy as a starting policy for the training of the AIRL algorithm, are reported in Figure 4.3.

Plot (a) compares the loss of the reward function when trained with a random policy as the Generator (as in the previous experiments) and the one that uses the external agent policy instead. As it can be observed, when using the external agent policy instead of the random one, the Discriminator loss is overall higher, showing that a random algorithm that explores the whole environment is more effective for learning the reward function than a policy that explores the environment much less.

Plot (b) shows the results of the policy training on the reward function



(a) Comparison of Discriminators' loss (b) AIRL on expert demonstrations starting from external agent policy

Figure 4.3: (a) reports the values of the Discriminator loss during AIRL training on the non-sustainable expert demonstrations when the Generator is a random policy and when is the external agent policy. (b) reports the total reward obtained, the episode's steps, and the number of violations of the policy trained on the reward function obtained in experiment 1 starting from the external agent policy.

obtained in the first experiment, starting from the external agent policy. As we can observe, it has many difficulties in retrieving the optimal behaviour. It struggles to learn the norm in just 100 episodes. The original policy has reinforced through rewards the action of gathering apples in the wrong area. Unlearning this behaviour, which is already incorporated into the policy can be challenging for an RL agent. The agent also has difficulties learning a renewable policy since the original one is not sustainable, and it is difficult not to end up in the Tragedy of the Commons when it does not follow the norm.

Hence, these results show that, even if the external agent is already aware of the final goal and knows how to gather apples, this does not benefit the learning of the norm if part of its policy must be unlearned. It is more advantageous to learn the new policy from scratch.

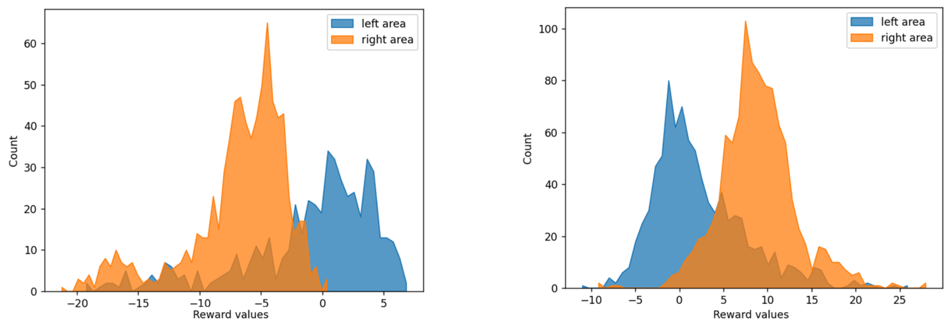
4.3 Discussion

4.3.1 Discussion on the Retrieved Policies

The drop to 0 of the error functions, shown in Figure 4.2, proves that the policies manage to learn the norm established to solve the Social Dilemma using Inverse Reinforcement Learning on the expert trajectories. The results were also evaluated by checking if the model can recover the expert policy and learn a sustainable one.

Regarding the policy of the first experiment, trained on the non-sustainable expert demonstrations, shown in plot (a), the *steps* function reveals that the policy never ends up in the Tragedy of the Commons. It means that it learns the renewable policy right away and never recovers the exact same policy of the expert. However, the agent's behaviour is similar to the expert's one. It gathers all the apples in its area quickly, then stops for some steps on a particular side of the map, giving enough time for the apples to respawn. It can be interpreted as intentional behaviour. The reward function retrieved by the IRL algorithm manages to reflect the real rewards of the original system. In this way, the agent is able to learn an optimal renewable policy really quickly. It is also important to note that the model learns the norm and the optimal policy in a few episodes. It is because rewards learned through IRL are derived from expert behaviour, which provides a more natural and informative signal for learning. It is also relevant to consider that even if learning this policy took much fewer iterations than the ones needed to train the policy on handcrafted rewards, each training step is computationally more expensive and more time-consuming. It is due to the forward step required to interrogate the reward function at each timestep to retrieve the correct reward.

In the second experiment, we can observe that the training is more unstable but still manage to learn a sustainable policy that follows the



(a) Distributions of the obtained reward function values

(b) Distributions of the differences between the two reward function values

Figure 4.4: (a) reports the rewards distributions obtained by running the external agent on the reward function retrieved in experiment 1 on the left and right areas. (b) reports the distributions of the differences between the corresponding rewards obtained by running the external agent on the reward function retrieved in episode 1 and on the reward function retrieved by running AIRL on the external agent demonstrations on the left and right areas.

norm. We also analysed the trajectories to investigate if it learned the expert policy. The learned policy consists in gathering the apples just from one tree. It can be considered similar to the expert's since it keeps the other trees untouched as a recharge factor, but it is far from being an optimal policy. The sustainable expert policy is more complex to learn than the non-sustainable one since it also makes significant use of the soft policy. In the expert demonstrations, different actions could follow the same state, meaning that the probability distribution over the actions is stochastic rather than deterministic. This stochasticity could be challenging to achieve since it could take more exploration, hence more training, for the agent to learn.

4.3.2 Reward Function Analysis

Lastly, we analysed the obtained reward function to investigate what information about the norm it encodes.

We examined the distributions of the reward values in the two private areas by generating 20 episodes using the external agent policy, which gathers resources in both areas, and collecting the rewards at each step from the retrieved reward function. The trajectories obtained by running the episodes were preprocessed by eliminating double state-action pairs to produce a set of tuples (state, action, reward) to analyse. We compared the obtained distribution by plotting the histograms of the rewards reported in Figure 4.4 (a). The left area was considered to be the one where is allowed to gather resources. As expected, the two distributions are quite different. The rewards obtained by the agent in the left area are higher, with most of them being positive values. The rewards obtained in the right area, instead, are lower and all negative.

In a second analysis, we compared the obtained reward function, which infers the norm, with the one of the external agent, which does not. In order to do that, we first trained the AIRL algorithm on the external agent trajectories to obtain a reward function that was more similar to the expert's, one in teams of magnitude and scale, compared to the handcrafted one where the reward values are all 0s and 1s. Then, we generated 20 episodes again using the external agent policy, preprocessed them, and collected the rewards from both functions for each step. We compared the distributions of the differences between corresponding rewards (rewards obtained from the same state-action pair) in the two areas of the map. Results are shown in Figure 4.4 (b). We can observe that the differences between the values of the two reward functions are higher in the right area, corresponding to the one where the norm is active and encoded in the expert reward function.

The two analyses prove again that the reward function obtained by the IRL algorithm is able to learn the norm of the system and encodes information about it. The external agent, unaware of the norm, could potentially extract this information from the reward function and use it to infer the norm.

To test this hypothesis, a supervised model, which we will refer to as Norm Classifier, was trained on labelled state-action pairs, with the label being:

$$label = \begin{cases} 1 & \text{if the pair lead to a state that violates the norm} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

The external agent, unaware that the system's norm is correlated to the two areas of the map, can not label the pairs based on the position of the next state on the map, and neither can access Figure 4.4 (b).

Instead, it can access the overall distribution of the differences between the values of two reward functions and use a threshold to choose the labels, as shown in Figure 4.5. If the difference between the two rewards obtained in a state-action pair is greater than the threshold, the next state is considered to violate the norm, and the label assigned to the state-action pair is 1, otherwise, 0 is assigned. It is because the two reward functions differ more in states where the norm is active since the expert function encodes it and the external agent function does not. In the examined problem, the established norm is a prohibition norm. As a consequence, the rewards of the expert function on states affected by the norm will be lower than the external agent's one to discourage the agent from violating the norm. If

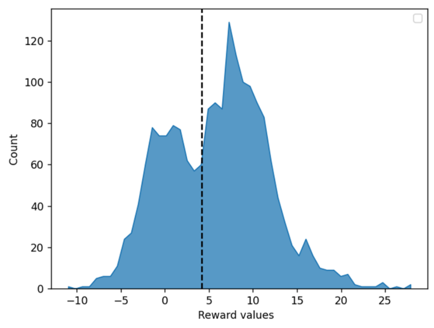
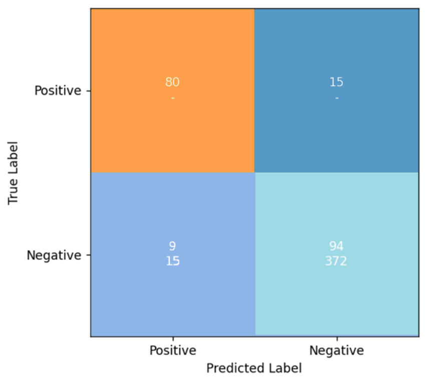
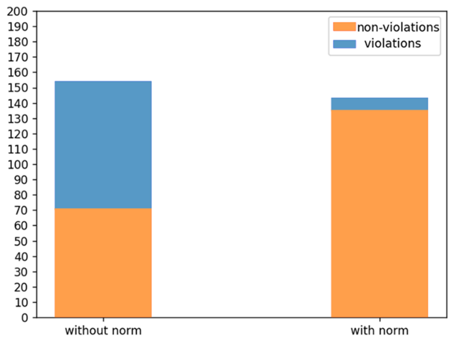


Figure 4.5: The plot reports the distribution of the differences between the corresponding rewards obtained by running the external agent on the reward function retrieved in experiment 1 and on the reward function retrieved by running AIRL on the external agent demonstrations, on the whole map. The threshold divides what are predicted to be the values corresponding to state-action pairs where the norm is active to the one where it is not active.



(a) Confusion matrix of the Norm Classifier



(b) Resources gathered not using the Norm Classifier and using it.

Figure 4.6: (a) reports the confusion matrix of the predictions made by the Norm Classifier on a random policy. The values above represent the predictions made when a resource was gathered. The values below represent the predictions made at each step. (b) reports the number of resources gathered by a 1) random policy and 2) a random policy which does not take actions that are predicted as norm violations by the Norm Classifier.

instead, it was an obligation norm, the rewards would have been higher to encourage the agent to behave following the norm.

The model was trained on 40 episodes, preprocessed again by eliminating double state-action pairs to avoid over-represented samples in the dataset. The accuracy achieved on the test set is 0.9. To test this model, a random policy was run on the environment, and at each step, the Norm Classifier was interrogated if taking action a in state s was a violation of the norm.

The confusion matrix is reported in Figure 4.6 (a). The values on the first row correspond to the predictions made by the Classifier only when a resource was gathered. The values on the second row correspond to the predictions at each step, but the ones of the Positive True Label are missing. The reason is that the norm of private areas has no clear rules. It is clear that gathering apples in the area of the other agent violates the norm and that moving and gathering apples in its own area does not. It is unclear if moving around in the other agent's area is permitted.

In principle, it is, but since there is no gain in doing that, the expert never crosses the wall that separates the two areas. The external agent could infer from the expert observations that crossing the wall violates the norm. Hence, we decided to consider 0 and 1 predictions to be both correct for each state-action pair in which the agent moves around in the wrong area without gathering resources. For this reason, those samples were not considered in the calculation of the confusion matrix.

Figure 4.6 (b) reports a comparison between two bar charts, one representing the resources gathered by an agent which follows a random policy, the other by an agent that chooses a random action at each step and interrogates the Norm Classifier about the presence of the norm. If the Classifier predicts that the next state violates the norm, the agent chooses another action.

As we can observe in both figures, the trained model is not perfect. The main reason is that the labels themselves are not all correct. In Figure 4.4 is clear that a large number of entries of the two distributions share the same values, leading to a wrong assignment of the labels to the state-action pairs. Even though the Norm Classifier makes errors, we can show that a random policy, without any information about the norm, was able to move around in the environment, reducing the number of norm violations by a large amount just by relying on this simple classification model.

Chapter 5

Conclusion

5.1 Summary

We have approached the open research problem of Norm Inference by using Inverse Reinforcement Learning to retrieve a reward function and a policy that follows the expert behaviour in a Social Dilemma problem. We have shown through experiments how the retrieved policy manages to learn both the norm and a sustainable behaviour. We have further analysed the reward function to show how it encodes information about the norm. We have provided a simple example of how it is possible to extract and use this information on a random policy by training a classification model to recognise norm violations.

With future progress in the IRL field, it will be possible to retrieve better and better reward functions, which will automatically improve both our experiments' results and our proposed Norm Classifier.

This study aims to demonstrate that it is possible to use IRL for Norm Inference in Social Dilemmas, and even if the problem approached is rather a simple one and the performance could be improved, it is a starting point for future research.

5.2 Future Work

The problem discussed in this dissertation could be complicated by establishing more than one norm, and some considerations can be made.

The information that can be extracted from the retrieved reward function in case of more than one norm, using the approach described in this project, will be able to distinguish between obligation and prohibition norms, following the reasoning done at the end of Section 4.3.2. The Norm Classifier could be trained to predict three classes (obligation norm, permission norm, and prohibition norm), and based on the predictions, an agent could behave accordingly.

It could be argued, in principle, that it is unnecessary to distinguish every single norm from each other to teach an agent how to behave in the presence of norms and that it is only needed to distinguish between obligation and prohibition norms. However, this could not hold based on what the information about the norms wants to be used for.

Future research could bring new and better ideas to this approach which is still considered to be an open problem.

Bibliography

[1] Hardin Garrett. The Tragedy of the Commons. In *Science*, Vol. 162, No. 3859, 1243-1248, 1968.

[2] Michael W. Macy and Andreas Flache. Learning Dynamics in Social Dilemmas. In *Proceedings of the National Academy of Sciences*, Vol 99 (suppl 3), 7229-7236, 2002.

[3] Leibo J.Z., Zambaldi V., Lanctot M., Marecki J., Graepel T.. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 464–473, 2017.

[4] Haarnoja T., Zhou A., Abbeel P. and Levine S.. Soft Actor–Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, 2018.

[5] Andrew Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*, 1999.

[6] Ziebart B.D., Maas A., Bagnell J.A. and Dey A.K.. Maximum Entropy Inverse Reinforcement Learning. In *National Conference on Artificial Intelligence*, 2008.

[7] Ho J. and Ermon S.. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 4565–4573, 2016.

[8] [8] Goodfellow I. et al. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2672–2680, 2014.

[9] Fu, J., Luo, K., and Levine, S.. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[10] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models, abs/1611.03852, 2016.

[11] Bastin Tony Roy Savarimuthu, Stephen Cranefield, Maryam Purvis, and Martin Purvis. Identifying conditional norms in multi-agent societies. In *Proceeding of the international workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN@AAMAS 2010)*, 19–24, 2010

[12] Therborn G.. Back to Norms! On the Scope and Dynamics of Norms and Normative Action. In *Current Sociology*, Vol 50, 863-880, 2002

[13] Stephen Cranefield, Felipe Meneguzzi, Nir Oren, and Bastin Tony Roy Savarimuthu. A Bayesian approach to norm identification. In *22nd European Conference on Artificial Intelligence*, 622–629, 2016.

[14] Bastin Tony Roy Savarimuthu, Stephen Cranefield, Maryam Purvis, and Martin K. Purvis. Obligation norm identification in agent societies. In *Journal of Artificial Societies and Social Simulation*, 13(4), 2010.

[15] Bastin Tony Roy Savarimuthu, Stephen Cranefield, Maryam Purvis, and Martin K. Purvis. Identifying prohibition norms in agent societies. In *Artificial Intelligence and Law*, 21(1), 1–46, 2013.

[16] Malle B. F., Scheutz M., Austerweil J. L.. Networks of social and moral norms in human and robot agents. In *In Aldinhas Ferreira M. I., Silva Sequieira J., Tokhi M. O., Kadar E., Virk G. S. (Eds.), A world with robots: International conference on robot ethics: ICRE 2015*. Vol. 84, 3–18), 2017.

[17] Bastin Tony Roy Savarimuthu, Stephen Cranefield, Maryam A. Purvis, Martin K. Purvis. Internal Agent Architecture for Norm Identificatio. In *Lecture Notes in Computer Science* Volume 6069), 2009.

[18] Y. Guo, B. Wang, D. Hughes, M. Lewis and K. Sycara. Designing Context-Sensitive Norm Inverse Reinforcement Learning Framework for Norm-Compliant Autonomous Agents. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 618-625, 2020.

[19] Julien Perolat, Leibo J.Z., Zambaldi V., Lanctot M., Charles Beattie, Karl Tuyls, Graepel T.. A multi-agent reinforcement learning model of common-pool resource appropriation. In *Annual Conference on Neural Information Processing Systems (NIPS)* 2017.

[20] Tuomas Haarnoja et al.. Soft Actor-Critic Algorithms and Applications, arXiv:1812.05905, 2019.