PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF COMPUTER SCIENCE
POSTGRADUATE PROGRAMME IN COMPUTER SCIENCE

LEARNING HOW TO ACT IN A NORMATIVE ENVIRONMENT THROUGH OBSERVATION

LEONARDO ROSA AMADO

Advisor: Prof. Felipe Meneguzzi

Porto Alegre
2021

LEARNING HOW TO ACT IN A NORMATIVE ENVIRONMENT THROUGH OBSERVATION

ABSTRACT

Most societies are driven by social norms, which directly affect how people in such societies behave. When entering a new society, one must learn how to comply with its norms to fit in. Such learning becomes more challenging in realistic environments in which there is uncertainty in the outcomes of actions. Reinforcement learning algorithms are capable of learning how to act in environments with uncertainty, however, such algorithms require a reward function indicating which behaviors are more desirable. We propose an approach to learn how to act in a normative multi-agent system by observing actions of a given agent that is member of such system, without an explicitly defined reward function. Our goal is to learn a monolithic reward function that configures normative behavior. To accomplish such task, we aim to use execution traces of an agent to teach our new agent how to comply with the environment norms.

Keywords: Reinforcement Learning, Apprenticeship learning, Norms, Normative Multi-agent system.

LEARNING HOW TO ACT IN A NORMATIVE ENVIRONMENT THROUGH OBSERVATION

RESUMO

A maioria das sociedades segue normas sociais, as quais afetam diretamente o comportamento dos seus integrantes. Quando ingressamos em uma nova sociedade, temos que aprender a respeitar suas normas sociais para nos encaixarmos. Esse aprendizado fica mais desafiador quando lidamos com situações reais, onde se há incerteza sobre o resultado das ações. Algoritmos de aprendizado por reforço são capazes de aprender como agir em ambientes onde se há incerteza. Porém, para esses algoritmos convergirem, eles necessitam uma detalhada função de recompensa. Nesse projeto, apresentamos uma abordagem para aprender como agir em sistemas multi-agentes regidos por normas, observando ações de agentes que pertencem a esse sistema. Para realizar esse aprendizado, nós pretendemos usar essas observações para ensinar um novo agente a respeitar as normas desse ambiente.

Palavras Chave: Aprendizado por reforço, Aprendizado por reforço inverso, Normas, Sistemas multi-agentes normativos.

LIST OF ACRONYMS

IJCNN – International Joint Conference on Neural Networks

AAAI – Association for the Advancement of Artificial Intelligence

ICML – International Conference on Machine Learning

IJCAI – International Joint Conference on Artificial Intelligence

AAMAS – International Conference on Autonomous Agents and Multiagent Systems

MDP – Markov Decision Process

SARSA – State-Action-Reward-State-Action

MAXENT – Maximum entropy inverse reinforcement learning

MLIRL – Maximum Likelihood Inverse Reinforcement Learning

DEEPIRL – Maximum entropy deep inverse reinforcement learning

IRL – Inverse Reinforcement Learning

MMP – Maximum-margin Prediction

MLE – Maximum Likelihood Estimation

FCNN – Fully Convolution Neural Networks

MAS – Multi-agent system

NORMAS – Normative Multi-Agent System

NMDP – Normative Markov decision process

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF SYMBOLS

LIST OF TABLES

CONTENTS

1. INTRODUCTION

Social norms are informal rules that govern the behavior of members of a society [34]. Different societies often have different social norms, one example would be people walking slowly in the right side of the pavement so other people can walk fast on the left, whereas in other societies this does not apply. Human agents often learn norms by observing other people within a society following such norms. In order to avoid potential social penalties, agents have an incentive to both follow known norms and learn new norms as they evolve. However, imagine the scenario of an agent being introduced to a new society, with a new set of social norms. When this agent is first introduced to the new society, it may not know how to act as a member of this society. To learn how the society works, the agent may either start acting with no regard for the unknown social norms and learn them whenever it receives a social punishment or reward. Alternatively, this agent may observe the behavior of other agents and infer how it should act, aiming to avoid receiving punishments for violating unknown norms. This latter alternative is often a more desirable mechanism to learn social norms.

Inverse reinforcement learning is a technique used to teach an agent how to perform in an environment based on observations of other agents behaving in such environment. The agents acting in the environment are following an unseen reward signal, aiming to maximize their well-being in the environment. The goal of inverse reinforcement learning is to learn how to replicate such behavior without direct access to such reward signal, thus implicitly learn the set of rules governing the observed behavior.

Imagine that an agent can observe other agents following the norms by comparing what the agent itself would do in a situation and contrasting it with the other members of society. Such as the previous example of walking in the right side of the pavement, the agent can learn how to behave by observing everyone walking slowly on the right side and leaving space so that other agents can walk fast on the left side. In current approaches [17], the agent is given a norm penalty , where authors apply reinforcement learning algorithms to behave in an environment where they do not know the norms.

Perhaps elucidate how this is encoded in the reward function

We aim to develop an approach to learn how to behave in a stochastic norm-driven environment by looking at how other agents interact with such environment. To model such approach, we investigate the following techniques. First, we assume the domain model to be a markovian decision process, as it is able to represent stochastic environments [7]. This environment must be a multi-agent system where we can introduce a learning agent that infers how to behave in such society. Second, we investigate norms, as there are governing norms in this scenario,in which agents are sanctioned for violating them. Third, we assume that the acting agents use reinforcement learning techniques, as their ability to perform in stochastic scenarios with unknown constraints [38]. Finally, we investigate inverse reinforcement learning algorithms, as they have the ability to learn reward functions from observed data. We believe our approach could be used for real world applications, such as robots interacting with human norms.

1.1 Publications

Despite having no publications about the specific subject of our proposal, during the two-years of our Ph.D. program, we published the following works:

- A full paper at IJCNN 2018 with the titled: "Goal Recognition in Latent Space" [4];
- A full paper at AAAI workshop on Plan, Activity, and Intent Recognition titled: "An LSTM-Based Approach for Goal Recognition in Latent Space" [6];
- A full paper at ICML/IJCAI/AAMAS 2018 Workshop on Planning and Learning (PAL-18) titled: "LSTM-Based Goal Recognition in Latent Space"; [5]
- A journal paper at The Knowledge Engineering Review, titled: "Q-Table compression for reinforcement learning" [3].
- We submitted a demo to AAMAS titled: "LatRec: Recognizing Goals in Latent Space". Acceptance notification is scheduled to February 25.

1.2 Proposal Outline

This proposal is divided as follows. In Chapter 2, we provide background on machine learning, markov decision process, reinforcement learning, inverse reinforcement learning and social norms. In Chapter 3 we detail our approach, establishing the research questions and the general goals, and provide a formal definition of the problem to apply our approach. In Chapter 4, we discuss the activities and work plan regarding the remaining two years of our Ph.D. program. In Chapter 5, we survey related work. In Chapter 6, we discuss our preliminary results. Finally, we conclude this proposal.

2. BACKGROUND

In this chapter, we introduce the fundamental background knowledge to the main topics discussed in this proposal. First, we provide a quick overview of the basics on machine learning. Second, we explain the formal definition of markov decision processes. Third, we detail Reinforcement learning concepts and two algorithms. Third, we explain the idea behind Inverse Reinforcement Learning, and detail the algorithms studied in this proposal. Finally, we quickly summarize an overview of Social Norms.

2.1 Machine Learning

Machine learning can be described as the field of study that gives computers the ability to learn without being explicitly programmed [35]. Machine learning studies the construction of algorithms capable of learning and build models from data. Mitchell provides a formal definition of machine learning as follows [20]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." There are three main types of machine learning algorithms [31]:

- Supervised Learning.
- Unsupervised Learning.
- Reinforcement Learning.

In supervised learning, the objective is to infer a function or model using labeled data for training [22]. This labeled data is a set of training examples containing the expected output given a set of features. In Machine Learning, features are values that we use to represent data. For example, the features of a pixel could be the RGB value of this pixel. Using the training examples, a supervised learning algorithm produces an inferred function. After the training, the learned function should be able to compute the expected output given a set of feature values. For example, suppose we are trying to learn a function capable of predicting how much a house costs, based on the size of the house. To train the algorithm, we were given a few examples of house pricing, containing the price of the house and the size of the house. In this example, the size of the house is our feature value, and the price of the house is the output we want to predict. We train the supervised learning algorithm using the examples of house pricing provided. The algorithm learns a function capable of predicting the house, based on its size. This learned function is called learned model.

The idea of unsupervised learning is to infer a function to describe a hidden pattern on unlabeled data. Similar to supervised learning, in unsupervised learning the algorithm is given a training data set. However, the data in this data set does not have an expected output. A usual

task of unsupervised learning is to cluster the data into different categories. For example, suppose we are given a data set containing multiple instances of a category of flowers. Each instance contains the characteristics of a flower, such as color, size, number of petals. All these flowers are currently set in the same category. However, there are many differences between the flowers in this category, leading to the idea of dividing this category into two new sub-categories. Since the instances are not classified using these new sub-categories, we apply an unsupervised learning algorithm to cluster the instances into two sub-categories. After executing the algorithm, the instances are clustered and we are able know which flower belongs to which sub-category.

Unlike supervised and unsupervised learning, reinforcement learning operates in a different environment, without the explicit need of a training set. Usually, a reinforcement learning environment is modelled as a Markov Decision Process, where an agent acts and receives signals from the environment. The objective in reinforcement learning is maximize the expected reward by optimizing its behavior in the environment.

Machine learning tasks do not differ only based on the input given, but also on the desired output for each task. Some of the categorizations based on output are [10]:

- Classification task. In classification, the algorithm must determine which class an instance belongs to. The possible classes are already known beforehand.
- Regression task. Similar to classification, but instead of a discrete value, the algorithm tries to find a function that maps an input to a continuous value.
- Clustering task. In clustering, the objective is to find a hidden pattern in a set of inputs, dividing those inputs in multiple classes. Those classes are not known beforehand.
- Density estimation task. The objective is to find the distribution of inputs in some space.
- Dimensionality reduction task. The objective is to simplify inputs to a smaller representation, without losing information from the data.

2.2 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework used to model decision making in environments where the outcomes are only partially controlled by the agent [7]. MDPs are meant to be a a straightforward mathematical framing for interaction to maximize rewards in stochastic environments. The decision maker acting in an MDP is called an *agent*. This agent continuously interacts with an *environment* (i.e., everything outside the agent), which in turn returns a new situation to the agent. These situations are denoted as states. Furthermore, a reward signal is given when arriving at a state, which represents how desirable it is to be in such state. The acting agent aims to maximize the expected reward through its choice of actions. These interactions

happen through discrete (which can be extended to continuous cases) time steps, $t = 0, 1, 2, ..$. In Figure 2.1, we provide a simple illustration of the agent acting in an MDP [38, Chapter 3].

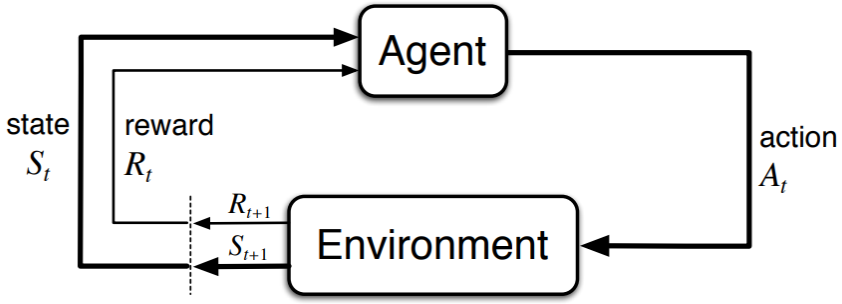


Figure 2.1: Agent-environment interaction in an MDP

In a finite MDP, the set of states S , actions A , and rewards R are all finite [27]. The actions are responsible for transitioning from one state to another. This transition between states define how the environment works. Thus, we define this transition as a probability of achieving state s' and receiving reward r, when applying action a in a state s. This process is called transition function, which is represented as:

$$p(s', r|s, a) \doteq PrS_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a \tag{2.1}$$

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \tag{2.2}$$

Change here to standard transition function

The function p defines the dynamics of the MDP. It is the framework mechanism do define the response for each agent action in the environment. This function returns a probability since in some MDP models, actions do not always yield the same results, hence it is possible to achieve different states executing the same action in the same state. In the literature, there are many ways to define the transition function, such as $p(s'|s, a)$ [31], which we use instead of the definition used in Equation 2.13. Episodic MDPs end when a goal G is achieved, which is a terminal state where there are no possible actions. Usually the purpose of an agent in an MDP environment is to maximize the expected signal. This reward signal is given by a reward function $R(s)$ that maps the states S of the environment to a reward. Thus the agent performance can be measured as the accumulated discounted reward when achieving a certain terminal state (or goal).

$$\sum_{t=1}^{\infty} \gamma^t R(s_t) \tag{2.3}$$

where γ represents a discount factor, a value that determines how valuable are future rewards. With the discount, we can formally define the MDP framework.

Definition 1 (Markov Decision Process) *An MDP is defined by the following tuple* $M = \langle S, A, P, \gamma, R \rangle$*, where* S *is a set of states,* A *a set of actions,* P *the transition function of the*

environment, γ the discount factor, and R a reward function that maps a reward to every state $s \in S$.

Having defined an MDP, we now move to the problem of solving an MDP. Solving an MDP consists of finding a *policy*.

A policy π is a function that maps any state in the domain to an applicable action in such state. A policy that always returns the action that provides the highest reward of each state is called optimal policy, denoted by π^* . However, to compute such policy we must estimate how desirable a state is. The notion of how good/desirable a state is, is defined in terms of future achievable rewards when following a certain policy. This estimate is called state-value function, or utility. These value functions represent either how good a state s is when following a policy π, denoted $v_\pi(s)$, or how good executing action a in a state s is $(Q_\pi(s,a))$. The *Bellman equations* [7, 8, 38] provably compute these estimates, as shown in Theorem 1.

Change so there is no more upper-case v

Add linear equation solution for bellman

$$v_\pi = R^\pi + \gamma P^\pi v_\pi \tag{2.4}$$

$$v_\pi = (I - \gamma P^\pi)^{-1} R^\pi \tag{2.5}$$

Theorem 1 (Bellman equations) *Given an MDP $M = \langle S, A, p(s'|s,a), \gamma, R\rangle$, and a policy $\pi : S \to A$. Then for all $s \in S$, $a \in A$, V_π and Q_π satisfy*

$$V_\pi(s) = R(s) + \gamma \sum_{s'} p(s'|s, \pi(s) V_\pi(s') \tag{2.6}$$

$$Q_\pi(s,a) = R(s) + \gamma \sum_{s'} p(s'|s,a) V_\pi(s') \tag{2.7}$$

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{2.8}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{2.9}$$

Bellman equation using expectation, Bruno did not ask specifically for this, but it is better in my opinion

Thus, with the value function of a state, we can now compute the value function of a policy. Policies with higher value functions are preferable, as their accumulative reward is higher. Therefore, a policy π is defined to be better than or equal to a policy π' if its expected return is greater or equal to that of π' for all states. Thus, $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$. Hence, an optimal policy π^* is better or at least equal to all other policies.

dont use p(s), only P

Theorem 2 (Bellman Optimality) *Given an MDP $M = \langle S, A, p(s'|s,a), \gamma, R\rangle$, and a policy $\pi : S \to A$. Then π is an optimal policy for M if and oly if, for all $s \in S$,*

$$\pi(s) \in arg\max_{a \in A} V_\pi(s) \tag{2.10}$$

$$\pi(s) \in arg\max_{a \in A} Q_\pi(s,a) \tag{2.11}$$

With Theorem 2, we can define the MDP problem. We define the MDP problem as finding an optimal policy for a given MDP M.

Definition 2 (MDP problem (optimality)) *Given an MDP $M = \langle S, A, p(s'|s,a), \gamma, R\rangle$, we want to compute a policy π^*, such that this policy is optimal, meaning that every value function of this policy is optimal. Thus,*

$$V_\pi(s) = \max_\pi v_\pi(s),$$

for all $s \in S$. There can be more than one optimal policy, to solve the problem we must compute only one.

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.12}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.13}$$

use this as optimal value function

With Theorem 2 and Definition 2 , we can use dynamic programming techniques to compute optimal policies in a complete MDP. First, we demonstrate the policy evaluation algorithm, which computes the value function values of a given policy π. The idea is that through random initialization of value functions of every state s, we can estimate the real value function of the policy the agent is following. To compute $V(s)$, we use a small variation of Equation 2.6, considering the probability of the policy choosing certain action, as to deal with uncertainty in the environment. Algorithm 1 shows the pseudo-code for the policy evaluation algorithm. Note that we use a parameter θ as a stopping mechanism for the algorithm. This parameter represents a measure of accuracy, so it usually is a number slightly greater than 0 (such as 0.05).

Check Delta just to be sure

Algorithm 1: Policy Evaluation

Input: π,the policy to be evaluated.
$M = \langle S, A, p(s'|s,a), \gamma, R\rangle$, the MDP tuple
θ, a small threshold for accuracy
Output: Values $V(s)$ for every $s \in S$
Initialize $V(s)$, for all $s \in S$, arbitrarily except that $V(terminal) = 0$;
While $\Delta > 0$:
  $\Delta \leftarrow 0$
  For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)(R(s') + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
return V

The purpose of policy evaluation is to compute the value function of a given policy. To solve an MDP, we must compute an optimal policy, for example by slight changing Algorithm 1.

Instead of following a given policy, we back the current best action of a state. Thus, instead of using $\sum_a \pi(a|s)$ in our $V(s)$ update function, we use max_a, which selects the action a with highest $V(s)$. We keep the random initialization of value functions, and the algorithm iteratively updates these value functions, selecting new actions as the values of $V(s)$ fluctuate. The algorithm stops when the highest difference in update is less than the parameter θ. After computing the value functions, we retrieve an optimal policy (given that the algorithm had enough iterations, which should be guaranteed with a small θ). After converging, we extract the optimal policy using Equation 2.10. This algorithm is called *Value Iteration*.

Check Delta just to be sure

Algorithm 2: Value Iteration

Input: $M = \langle S, A, p(s'|s,a), \gamma, R \rangle$, the MDP tuple.
θ , a small threshold for accuracy
Output: Deterministic policy, $\pi \approx \pi^*$
Initialize $V(s)$, for all $s \in S$, arbitrarily except that $V(terminal) = 0$;
While $\Delta > 0$:
  $\Delta \leftarrow 0$
  For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)(R(s') + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
For each $s \in S$:
  $\pi(s) = arg\max_a \sum_{s'} p(s'|s,a)(R(s') + \gamma V(s'))$
return π

With Algorithm 2, we can solve the MDP problem defined in 2. However, the Value Iteration algorithm requires a complete MDP to compute an optimal policy. In the next Section, we present a class of algorithms capable of computing an optimal policy without full knowledge of the MDP.

2.3 Reinforcement Learning

Reinforcement learning aims to learn from experience, by using obtained knowledge through sampling in an environment. A reinforcement learning algorithm is executed multiple times in a model, storing information about each execution with the objective of learning which actions in each state returned better rewards.

Most reinforcement learning algorithms assume that the environment can be represented as MDPs. When such is the case, they share most of the core components of the MDPs. However, different algorithms vary in what kind of information they need to learn a policy. In some models, it is not possible to know the exact outcomes of every action in every state. There are reinforcement learning algorithms that are capable of finding an optimal policy in an environment without the need of the transition function.

2.3.1 Q-Learning

Q-learning is a model-free reinforcement learning algorithm that does not need the transition function to learn the optimal policy [42]. The agent learns to act in the environment by testing the possible results of performing an action in a certain state, learning the utility of performing an action *a* in a state *s*, updating the value function $Q(s, a)$ (often called *Q-Value*). Thus, Q-Learning focuses on learning the value function of executing an action when the agent is in a particular state, rather than learning the value function of each state directly and then computing a policy, using Equation 2.9. However, in this scenario, we do not know the transition function $p(s'|s, a)$ of the environment. Instead of following a policy, Q-Learning backs-up the best action. The Q-Learning update function is then given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2.14}$$

where α is the learning function that defines how much the algorithm will be updated in each visit of such state-action pair. This function updates the previous state by using the $\max$ function, which backs-up the best action available in the actual state, returning the highest action-value attainable in such state. Unlike the Value Iteration algorithm, there is no random initialization of value function, since we do not know the transition function $p(s'|s, a)$. Thus, we need to approximate Equation 2.9 by exploring the environment.

A Q-learning agent uses a generic policy to explore an environment, and is often called the *exploration function* f. The exploration function is a policy used to mediate exploitation and exploration in the environment. Two examples of exploration functions in reinforcement learning are: $\epsilon-$greedy and optimistic estimate value function [39]. The ϵ-greedy policy uses a parameter ϵ $(0 \leq \epsilon \leq 1)$ to decide which action to perform using $Q(s, a)$. Hence, in state s the agent either chooses the action that has the highest Q-value $Q(s, a)$ with the probability of $1 - \epsilon$, or a random action otherwise. A larger ϵ means the agent will explore more random actions and exploit less known rewards. In many implementations, ϵ decreases when using state-action pairs that were already visited many times, to avoid excessive randomness. Thus, Q-Learning implementations often use a look-up table $N[s, a]$ to store how many times a state-action pair was visited. The optimistic estimate value function is an exploration function that forces the agent to visit new state-action pairs. If a state-action pair is completely new, then the exploration function sets its reward as the highest possible reward in the environment (R^+) when choosing the next action. Thus, usually this exploration function uses a threshold N_e, that is the number of times the agent is "forced" to explore each state-action pair. We can describe mathematically this function as:

$$f(Q(s, a), n) = \begin{cases} R^+ & if \quad n < N_e \\ Q(s, a) & otherwise \end{cases} \tag{2.15}$$

where $Q(s, a)$ is the Q-Value of state-action pair (s, a), and n the number of times this pair was visited.

With Equation 2.14 and an exploration function, we can implement the Q-Learning algorithm. We describe an implementation of Q-Learning in Algorithm 3. Note that instead of a policy, we are returning an action. This action is the best possible action in a given state, **considering** the exploration function ($arg\max f$).

Algorithm 3: Q-Learning step.

Input: percept, a percept indicating the current state s' and reward signal r'
Output: An action a.
Persistent: Q, a table of action value indexed by state and action, initially zero ;
N_{sa}, a table of frequencies for state-action pairs, initially zero ;
s, the previous state, initially null;
a, the previous action, initially null ;
r, the previous reward, initially null;
if $isTerminal(s')$ **then**
  $Q[s', None] \leftarrow r'$;
end
if s *is not null* **then**
  $N[s, a] \leftarrow N[s, a] + 1$;
  $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$;
end
$s \leftarrow s'$; $r \leftarrow r'$;
$a \leftarrow \arg\max_{a'} f(Q[s', a'], N[s', a'])$;
return a

Bruno said it is better to call step instead of algorithm, as it is only one execution

2.3.2 SARSA

With small tweaks to the basic Q-Learning algorithm we can achieve a new algorithm, the SARSA (State-Action-Reward-State-Action) [29]. SARSA is very similar to Q-Learning [31], with only a small change to the update rule. SARSA learns action-values relative to the policy it follows, while Q-Learning does it relative to the exploration policy it is following. Under some common conditions, they both converge to the real value function, but at different rates. The update rule for SARSA is represented as follows:

Say it can be greedy, change the text here

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma Q(s^{'}, a^{'}) - Q(s, a)) \tag{2.16}$$

Where a' is the action taken in state s'. The update rule is applied at the end of each $\langle s, a, r, s', a' \rangle$ quintuple. The SARSA algorithm acts on the environment and updates the policy based on actions taken. This behavior makes SARSA an on-policy algorithm, different from the Q-Learning algorithm that is off-policy, thus following generic exploration functions. SARSA learns the Q-Values associated

with following the policy itself, while Q-Learning learns the Q-Values associated with following the exploitation/exploration policy.

2.4 Generalization in reinforcement learning

Traditional reinforcement learning algorithms, such as Q-Learning, can struggle to converge when dealing with large state-spaces [18]. Even after some training, if the basic algorithm of Q-Learning is executed and a new Q-value is found, the algorithm follows the exploration function and explores the new Q-value, which can be a terrible choice. The problem comes from the fact that reinforcement learning algorithms are not able to generalize any information from the domain. If a new state is found, even though it is extremely similar to states already visited many times, the algorithm has no estimation of the utility of this state.

Function approximation is a viable option to generalize information from the domain, assuming that the reward function can be represented as a linear function. This means using other representation method for the value-function, instead of the Q-Table. The representation is an approximation since it is impossible to guarantee a true value function representation with only a linear function. [37]. To use function approximation, we compute a weighted linear combination of features that represent the state. In a scenario with *n* features, the algorithm must learn a weighted linear function of the set of features $f_1, ..., f_n$:

Check this later

$$\hat{U}_\theta(s) = \theta_1 f_1 + \theta_2 f_2 + ... + \theta_n f_n \tag{2.17}$$

$$\hat{v}_\theta(s) = \theta_1 f_1 + \theta_2 f_2 + ... + \theta_n f_n \tag{2.18}$$

A reinforcement learning algorithm can determine the weights for the parameters θ, approximating $\hat{U}$ (or $\hat{V}$) to the real utility function. Instead of having a large number of values in a table, we have a function based in a much smaller number of parameters, which is a big compression. With this approximated function, an agent is capable of guessing the utility of a state it has never visited.

No need, use just V

To apply this concept to reinforcement learning, we must adjust the parameters based on the real utility after each try. To do so, we must use an error function to determine how much we must change each parameter. We define $u_j(s)$ as the observed total reward from state s onward in the jth trial, then the error (loss function) is the half of squared difference of the predicted utility and the actual utility:

Perhaps dont use j

$$E_j(s) = \frac{(\hat{U}_\theta(s) - u_j(s))^2}{2} \tag{2.19}$$

changed U and u for v

$$E_j(s) = \frac{(\hat{v}_\theta(s) - v_j(s))^2}{2} \tag{2.20}$$

For the linear function approximation $\hat{U}_\theta(s)$, we have a simple update rule for each parameter:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s)), \\
\theta_1 &\leftarrow \theta_1 + \alpha(u_j(s) - \hat{U}_\theta(s))f_1, \\
\theta_n &\leftarrow \theta_n + \alpha(u_j(s) - \hat{U}_\theta(s))f_n.
\end{aligned}$$

Changing U for v

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha(v_j(s) - \hat{v}_\theta(s)), \\
\theta_1 &\leftarrow \theta_1 + \alpha(v_j(s) - \hat{v}_\theta(s))f_1, \\
\theta_n &\leftarrow \theta_n + \alpha(v_j(s) - \hat{v}_\theta(s))f_n.
\end{aligned}$$

Finally, using Equation 2.14 and the update rule described above, it is possible to derive an update rule based on the Q-Values of a domain. This update rule can be written as:

$$\theta_i \leftarrow \theta_i + \alpha(R(s) + \gamma \max_{s'} \hat{Q}(s', a') - \hat{Q}(a, s))f_i \tag{2.21}$$

2.5 Inverse Reinforcement Learning

In standard reinforcement learning we may not know R, but we have access to the immediate reward signal r. As long as we can model a reward function for a certain environment, we can learn a policy. However, small tweaks on the reward function can completely modify the behavior of an agent, and there are domains (specially real-world domains) where building a reward function is a complicated task, even for a domain expert [1]. Thus, Andrew Ng et al., introduced a series of algorithms called Inverse Reinforcement Learning [26]. The idea of IRL is that, by watching an expert behave in a environment using a reward function $R(s)$, we can learn an approximation $\hat{R}(s)$ of the reward function being used by this expert. Furthermore, we can use the learned reward function to train new agents, evaluating policy loss. This type of learning is called Inverse Reinforcement Learning [1].

In this section, we detail 4 algorithms computing $\hat{R}(s)$. First, we detail the standard IRL algorithm and display the formal definition of the IRL problem [26]. Second, we discuss a method for inverse learning that uses maximum entropy [45] (MaxEnt). Then, we detail the Maximum Likelihood Inverse Reinforcement Learning (MLIRL) [23], which uses the maximum likelihood estimation. Finally, we discuss the Maximum entropy deep inverse reinforcement learning (DeepIRL) algorithm [43], which is an extension of the regular maximum entropy IRL using neural networks.

2.5.1 Inverse Reinforcement Learning

The inverse reinforcement learning (IRL) problem can be characterized informally as follows [30]:

Given: 1) measurements of an agent's behavior over time, in a variety of states, 2) if needed, measurements of the sensory inputs of that agent; 3) if available, a model of the environment.

Determine: the reward function being optimized by that agent.

Briefly, the inverse reinforcement learning problem is the task of finding a reward function R that can explain certain observed behavior. We can formally define the IRL problem using the following definition, if we assume the state-space of our problem to be finite [26]:

Definition 3 (Inverse Reinforcement Learning Problem) *Given a finite state space S, a set of k actions $A = a_1, ...a_k$, a transition function describing the transition probabilities P[1], a discount factor $\gamma \in (0, 1)$, and a policy π; we wish to find the set of possible reward functions R such that π fits as an optimal policy in the MDP (S, A, P, γ, R). Thus the solution of an IRL problem is at least one singular reward function, capable of solving the MDP.*

According to Andrew et al., [26] there are two main sources of motivations for the IRL problem. The first arises from the potential use of reinforcement learning algorithms as computational models for animal and human learning [33, 42]. The idea to use reinforcement learning techniques in such scenarios are supported by physiological evidence that reinforcement leaning occurs in different species, such as bees and songbird vocalization [13, 25]. However, both works assume that the reward function is fixed and knows, which can be a strong assumption when modelling behaviour from animals, humans and other natural phenomena, in which we as humans consider unknown and only ascertainable through empirical investigation [26]. Thus, inverse reinforcement learning problems fit perfectly in systems which we ourselves can only describe through extensive empirical investigation. The second motivation comes from the task of building intelligent agents that can behave successfully in domains where the agent designer struggles to define a reward function. Such domains vary from walking in crowds [41], to efficiently drive cars in modern society, as it is hard to generate a numeric function that can be optimized to perform "well" in such scenarios. This is similar to already existing techniques such as Imitation Learning and Apprenticeship Learning. Thus, the main difference is that instead of just imitating the behaviour of the observed agent, IRL extracts the function that led the agent to behave in such way.

To solve the defined IRL problem, Andrew et al., provides a simple characterization of the set of all reward functions for which a given policy is optimal [26]. The characterization is defined by the following theorem:

[1] In this section, we use P_{sa} for the transition, as it is the notation used in Andrew et. at. original work. We keep this for simplicity if the reader wants to check some of the complete proof available in that work.

Theorem 3 *Let a finite state space S, a set actions $A = a1, ...a, k$, a transition probability matrix $\mathbf{P_a}$, and a discount actor $\gamma \in (0, 1)$ be given. Then the policy π given by $\pi(s) \equiv a_1$ is optimal if and only if, for all $a = a_2, ..., a_k$, the reward $\mathbf{R}$ satisfies:*

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_a})^{-1}\mathbf{R} \succeq \mathbf{0} \tag{2.22}$$

Theorem 3 defines the set of all possible answers for the IRL problem. The full mathematical proof of this theorem can be found in Andrew et, al,. original work [26]. This equation however is prone to degenerate solutions, such as constant reward functions (i.e., every state is rewarded equally) and a reward function that rewards 0 for all states. Furthermore, for almost every MDP there are too many reward functions that satisfies π, since very small alterations in a single reward of a state is enough to characterize a different solution. Therefore, to deal with such problems, Andrew et al. use Linear Programming to solve the IRL problem. First, we need a function that should be optimized and measures the quality of a reward function. A reward function solution should avoid deviating from the original policy π given. Thus, of all functions $\mathbf{R}$ satisfying Theorem 3, we aim to maximize:

$$\sum_{s \in S} (Q^{\pi}(s, a_1) - \max_{a \in A \setminus a_1} Q^{\pi}(s, a)) \tag{2.23}$$

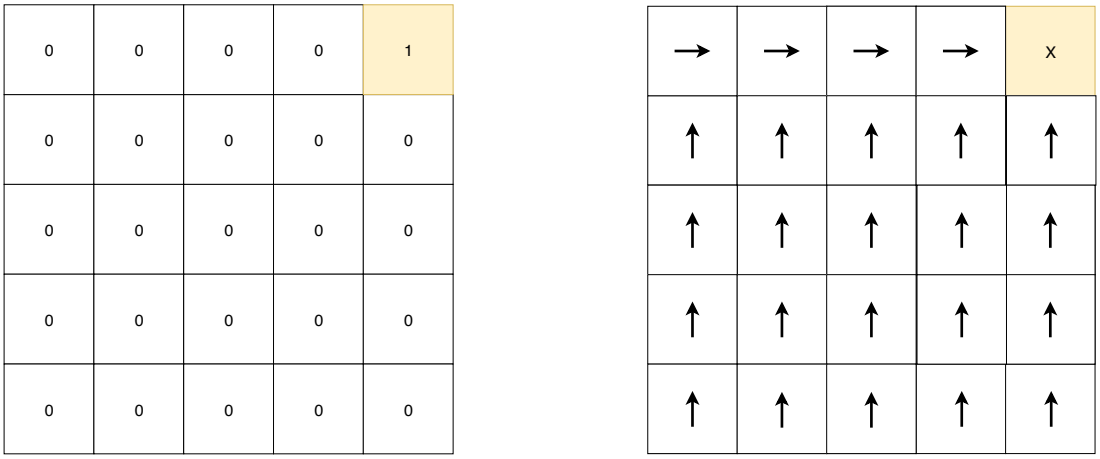
Therefore, we seek to maximize the sum of the differences between the quality of the optimal action and the quality of the next best action. Furthermore, Andrew et al. proposes the usage of a λ coefficient to increase the likelihood of choosing a simpler reward function (in this case by simpler we mean a reward function with smaller values). Putting it all together, this is our optimization problem:

$$\begin{aligned} &maximize \quad \sum_{i-1}^{N} min_{a \in \{a_2, ..., a_k\}}(P_{a_1}(i) - P_a(i))(I - \gamma P_{a_1})^{-1}R - \gamma||R||_1 \\ &s.t. \qquad (\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_a})^{-1}\mathbf{R} \succeq \mathbf{0} \quad \forall a \in A \setminus a_1 \\ &|R_i| \leq R_{max}, i = 1, ..., N \end{aligned} \tag{2.24}$$

where $\mathbf{P_a(i)}$ denotes the ith row of $\mathbf{P_a}$. This equation can be formulated as a linear program, and solved efficiently. To exemplify this, we show an example of an IRL problem.

Example 2.1 *Consider a 5x5 grid, with the following reward structure shown in Figure 2.2a. Each square represents a state, and the yellow square represents a* ***terminal state****, with the only positive reward of the environment. Our inverse reinforcement learning agent does not know the reward function, and only observe the policy described in Figure 2.2b. The acting agent can move in any of the four cardinal directions (North, East, South, West) in any of the states. If the agents moves in the direction of a wall, the agent will stay in the same state. To add uncertainty to the environment, there is a 30% chance that the agent will slide to one of the side states when trying to move (if the*

agent moves East, there is a 15% that the agent will move North and a 15% chance that the agent will move South).



(a) Ground truth reward function. (b) Observed policy

Figure 2.2: IRL example scenario.

To solve Example 2.1, we solve the linear programming problem we define in Equation 2.24. The result of solving such equation is shown in Figure 2.3. As we can see, the computed reward function is pretty close to the desired reward function.

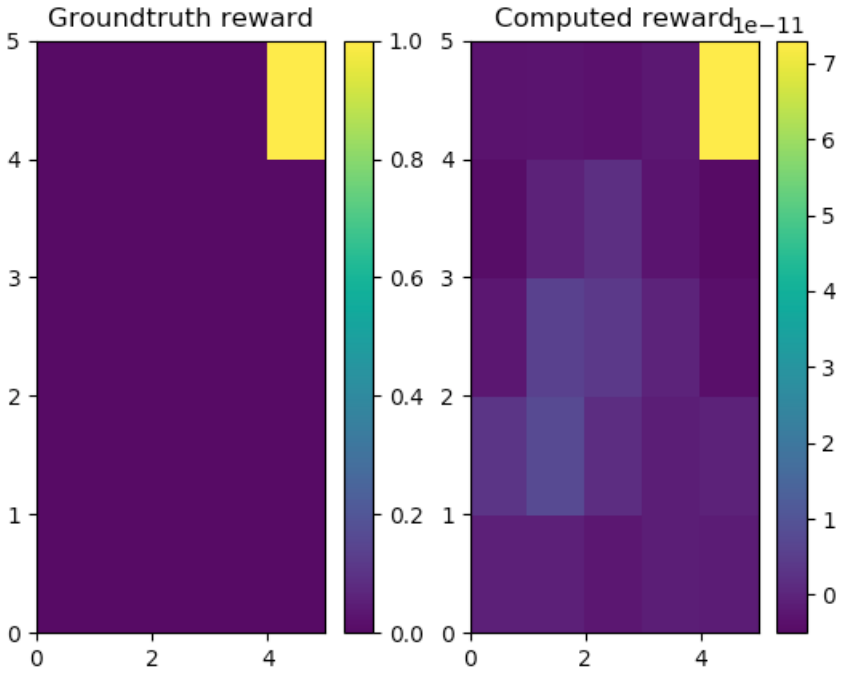


Figure 2.3: IRL example solution.

2.5.2 Maximum entropy Inverse Reinforcement Learning

Given the defined IRL problem, Brian et al. propose a new approach [45] to solve the IRL problem, by using the maximum entropy concept [15]. The goal in this approach is to compute a linear function, such as Equation 2.18, capable of making demonstrated behavior appear near-optimal. This function is assumed to be linear in the features, casting the IRL problem as a feature expectation problem. To compute such function, Brian et al. use a probabilistic approach to reason about uncertainty in the inverse reinforcement learning problem. Under the constraint of matching the reward function of the demonstrated behavior, they apply the principle of maximum entropy to resolve any ambiguities when choosing different reward functions.

Instead of using a policy of an observed agent, we now use the agent behavior ς as a trace of actions and states (i.e. a set of states s_i and actions a_i). We assume that the observed agent is trying to optimize a reward function $\mathbf{R}$, that linearly maps the features of each state, $\mathbf{f_{s_j}} \in \mathbf{R}$, to the estimated utility for visiting that state. Such as Equation 2.18, this equation is parameterized by reward weights θ. The combined reward of a trajectory is the sum of state rewards, that being the reward weight applied to the each feature of each state in trace ς, which can be written as the reward applied to the *feature counts* (the number of times the features appeared in the observed trajectory) $\mathbf{f}_\varsigma = \sum_{s_j \in \varsigma} \mathbf{f}_{s_j}$. Thus the reward value of a trajectory is:

$$reward(\mathbf{f}_\varsigma) = \theta^\top \mathbf{f}_\varsigma = \sum_{s_j \in \varsigma} \theta^\top \mathbf{f}_{s_j} \tag{2.25}$$

The same case we argued in the previous section applies here, where multiple reward functions can be answers in Theorem 3, such as degenerate solutions (e.g., all zeros). Ratliff et.al., cast this problem as a structured maximum-margin prediction (MMP) [28]. The idea is that they consider as a loss function the direct disagreement between the observed agent and a learned policy, and then learn efficiently a reward function based on this loss, using the structured margin method. However, this method struggles to converge when dealing with non-optimal observed behavior. To deal with non-optimal observed behavior and solve the problem of degenerate functions, Brian et. al. propose the usage of the maximum entropy principle.

Unlike previous work in IRL, maximum entropy inverse reinforcement learning considers a distribution over the entire class of possible behaviors instead of policies. Therefore, traces can have variable lengths for deterministic MDPs. Like distribution of policies, many different distribution of traces will match *feature counts* when demonstrated behavior is sub-optimal. The principle of maximum entropy affirms that, to best represent a current state of data, we must use the probability distribution which has the largest entropy. In the inverse reinforcement learning problem, this means that we can use this principle to resolve the ambiguities by choosing the distribution that does not show any additional preferences beyond matching feature expectations between an observed policy

and a learner's behavior. We show this process in Equation 2.26:

$$\sum_{Path\ \varsigma_i} P(\varsigma_i)\mathbf{f}_{\varsigma_i} = \tilde{\mathbf{f}} \tag{2.26}$$

The resulting distribution over traces is parameterized using our rewrad weights θ. With this model (Equation 2.27), plans with equivalent rewards have equal probabilities, and plans with higher rewards are highly more preferred.

$$P(\varsigma_i|\theta) = \frac{1}{Z(\theta)} e^{\theta^\top \mathbf{f}_{\varsigma_i}} = \frac{1}{Z(\theta)} e^{\sum_{s_j \in \varsigma_i} \theta^\top \mathbf{f}_{s_j}} \tag{2.27}$$

Given parameter weights, the partition function $Z(\theta)$, Equation 2.27 always converges for finite horizon problems and infinite horizons problems with discounted reward weights. With this equation, given demonstrated traces that are absorbed in a finite number of steps, the reward weights that maximize entropy must be convergent.

check this, not accurate

Furthermore, Brian et. al. describe the mathematical process for non-deterministic distributions. We do not detail these processes, as the mathematical background is quite dense as it is, and it is not entirely relevant to this proposal, as we do not propose to alter the maximum entropy algorithm, just use it. The maximum entropy IRL model (MaxEnt) resolves ambiguities in previous approaches, provides a convex, computationally efficiently method for computing the reward function weights.

2.5.3 MLIRL

Instead of using the Maximum Entropy concept to solve the Inverse Reinforcement Learning, Vroman et. al. cast the IRL problem as a maximum likelihood estimation problem (MLE). In maximum likelihood estimation, the goal is to find a set of parameters $\hat{\theta} \in \Theta$ given a mapping $f : \Theta \to \mathbb{R}$, the observed data ς[2], and the function $M(f)$ that generated the data. Since $\hat{\theta}$ stands for the approximated parameters set, θ stands for the parameters that generated the data. MLE computes $\hat{\theta}$ by using the following equation:

$$\hat{\theta} = arg\max_{\theta} P(\varsigma|M, f, \theta) \tag{2.28}$$

in some MLE problems, computing θ analytically can be quite simple. If f and the process M are known linear mappings, and enough data ς is provided, finding θ is solving a system of linear equations.

For the IRL problem, $f(\theta)$ is the expert's reward we want to approximate, and the process M is computing the optimal policy the given MDP (without the reward function R). Computing

[2]In MLE problems, the notation for observed data is D. However, to keep consistent with the notation of the MaxEnt algorithm, we use ς.

M means finding the optimal policy in the given MDP, which can be done with the Value Iteration algorithm we detailed in Algorithm 2. Then, we can use the optimal policy to generate demonstrations in the MDP. It is intractable to compute analytically θ in the IRL problem, thus we need to estimate $\hat{\theta}$ using Equation 2.28.

Given the set of demonstrations ς extracted by observing an expert act in the environment, we aim to approximate θ_E, which are the weights used in the expert's reward function. The expert is assumed to be acting optimally in the environment (with respect to its reward function), but to allow non-optimal behavior or noise in the observations, the Maximum Likelihood Inverse Reinforcement Learning (MLIRL) uses a Boltzmann distribution for action selection. We detail this process using the following equation:

$$Q_{\theta_A}(s,a) = \theta_A^T \mathbf{f}(s,a) + \gamma \sum_{s'} p(s'|s,a) \otimes Q_{\theta_A}(s',a') \tag{2.29}$$

where θ_A is the current computed weights. The 'max' function in the standard Bellman equation is replaced with an operator that blends values via Boltzmann exploration [36]: $\otimes Q_{\theta_A}(s,a) = \sum_a Q(s,a) e^{\beta Q(s,a)} / \sum'_a Q(s,a) e^{\beta Q(s,a')}$. Thus, the Boltzmann exploration policy is $\pi_{\theta_A}(s,a) = e^{\beta Q(s,a)} / \sum'_a Q(s,a) e^{\beta Q(s,a')}$. Following this exploration policy, the log likelihood of the trajectories in ς being visited with the weights θ is $L(\varsigma|\theta) =$

$$\pi_\theta(s,a) = \sum_{i=1}^{N} \sum_{(s,a)\in\varsigma_i} log \pi_\theta(s,a) \tag{2.30}$$

hence, MLIRL seeks $\theta_A = arg\max_\theta L(\varsigma,\theta)$, which is the maximum likelihood solution. This function is optimized using gradient ascent (but we could use other optimization functions). Putting it all together,we describe the MLIRL algorithm in Algorithm 4.

Algorithm 4: Maximum Likelihood IRL

Input: MDP/r, features $\mathbf{f}$, trajectories ς, number M of iterations, step size for each iteration (t) $\alpha_t, 1 \leq t \leq M$
Output: Reward weights θ_M
Initialize: Choose random set of rewards weights θ_1
for t = 1 to M:
  Compute Q_{θ_t}, π_{θ_t}
  $L = \sum_i \sum_{(s,a)\in\varsigma} log(\pi_{\theta_t}(s,a))$
  $\theta_{t+1} \leftarrow \theta_t + \alpha_t \bigtriangledown L$
return θ_M

The MLIRL algorithm is well-behaved and produces a well-defined answer for finite-horizon settings. Vroman et. al. propose alterations in the MLIRL algorithm so it can estimate modular reward functions. We do not detail these modifications in this proposal.

2.5.4 Maximum entropy Deep Inverse Reinforcement Learning

The Maximum entropy IRL algorithm we detailed in Section 2.5.2 computes a linear reward function that can generate a reward signal for every state in the environment. To overcome the limitation, much research has been done to extend IRL model to non-linear functions [12, 16]. While in principle these approaches extends the IRL paradigm to the flexibility of non-linear reward approximation, they scale badly when dealing with large state-spaces [43].

Thus, Wulfmeir et al. propose the usage of neural networks [9] to approximate the expert reward function. Neural networks have yielded state-of-the-art performances across a variety of domains, such as reinforcement learning [21]. Their application in the IRL problem comes from their compact representation of highly non-linear functions. Furthermore, once trained neural networks provide favorable computational time once trained.

The proposed neural network are Fully Convolution Neural Networks (FCNN) , which receives a as input a set of features $\mathbf{f} = f_1, ..., f_n$ of size n representing the state, and outputs a set of reward $r_1, ..., r_n$, for each of the features of the state. Thus, the reward weights θ are in the hidden layers of the network, which can represent non-linear weights using multiple weights for each feature. The network structure is detailed in Figure 2.4.

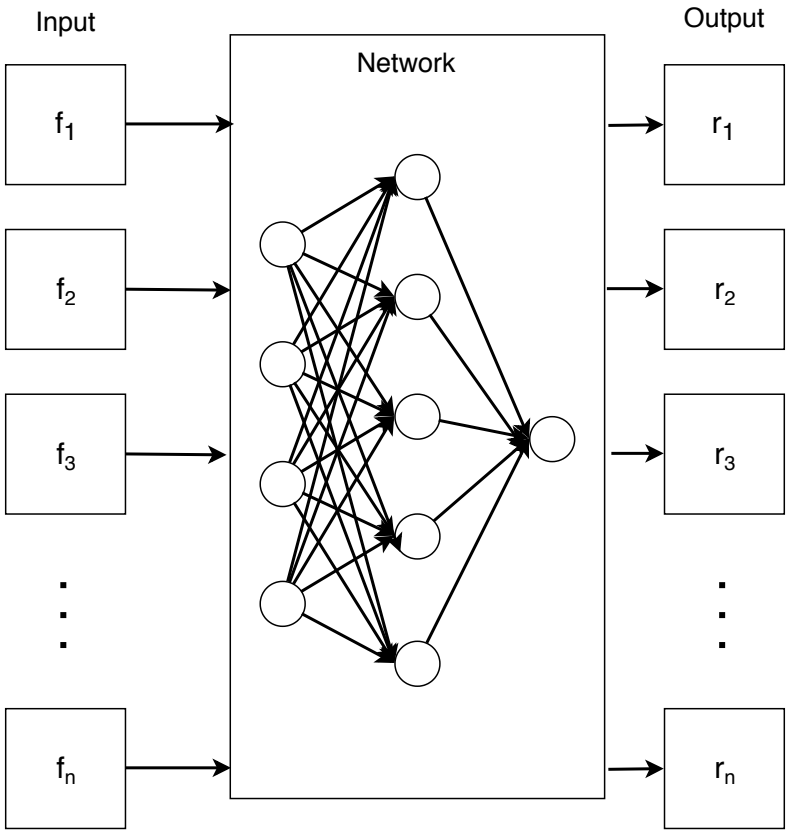


Figure 2.4: Agent-environment interaction in an MDP

We will not delve in the mathematical details of the Maximum entropy Deep Inverse Reinforcement Learning (DeepIRL) algorithm, as it would be necessary to detail neural network fundamentals, such as the backpropagation algorithm. DeepIRL outperforms MaxEnt in most scenarios, and performs best when dealing with life-long learning scenarios in the context of robotics, which inherently provide sufficient amounts of training data [43].

2.6 Social Norms

A norm can be defined as a rule for conduct which is generally complied by the members of a community. In multi-agent systems (MAS) , the researchers focus on social norms, since many aspects of MAS are borrowed from social concepts [32]. Social norms are generally composed of the following three main aspects:

1. Normative expectation of a behavioral regularity: The society expect that a given behavior is going to be followed by the agents of a society in a given circumstance.

2. Norm enforcement mechanism: When a norm is not followed, the violating agent could be subjected to a sanction. The sanction can be of any kind, such as monetary loss, physical or moral damage. In general, the sanction can be defined as a loss of utility.

3. Norm spreading mechanism: Agents with higher hierarchical level influencing other agents to follow a norm. Agents could also learn norms through imitation or learning mechanisms.

Researchers in multi-agent systems have studied how social norms could be implemented in agents. Norms are an interesting asset to MAS system as they help in maintaining social order and increase the predictability of behavior in society. A MAS driven by norms is a Normative Multi-Agent System (NorMAS). NorMAS can be defined as [11]: *A multi-agent system organized by means of mechanisms to represent, communicate, distribute, detect, create, modify and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfillment*. There is much research about applying norms in existing multi-agent systems, such as introducing it to BDI systems [19].

3. RESEARCH PROPOSAL

Thesis Statement: *We aim to develop an approach to learn normative rules and behavior in normative multi-agent scenarios. To accomplish this, we intend to formalize three distinct problems. (1) Learning how to act as an agent in a normative environment, avoiding violating norms. (2) Learning which features of the environment are responsible for triggering a norm violation. (3) Learning the sanction value of violating each norm.*

We propose an approach to compute a reward function in a normative environment by watching agents behave in such environment and use inverse reinforcement learning techniques to derive the normative rules. Our goal is to insert a new agent in this environment, capable of complying with norms, so that the agent passes as a new member of the environment.

This chapter is divided as follows. First, we detail the objectives of the research proposal. Second, we formalize the problem of learning how to act in a normative environment through observation. Finally, we formalize the problem of estimating sanction values and features of norms.

3.1 Objectives

3.1.1 Overall Objectives

We have five main objectives in this research: (1) formalize the problem of learning how to act in a multi-agent normative environment through observations; (2) develop an approach to act in the multi-agent environment through observation of an expert agent, using inverse reinforcement learning; (3) build an environment for experiments that is scalable in size, number of agents and number of norms; (4) design methods to extract the environment features that are used for each norm, and compute an estimate of the reward loss caused by the sanction of violating such norms; (5) experiment with the developed approaches in other scenarios and compare when using different inverse reinforcement learning algorithms.

3.1.2 Specific Objectives

Based on the overall objectives, we now define the specific objectives we aim to investigate in this proposal, by dividing them into five parts, as follows.

1. Formalize the problem of learning how to act in a multi-agent normative environment through observations. To do so, we assume the environment to be stochastic, and that the actions and transitions of state can be modelled as an MDP.

2. Develop an approach to act in the multi-agent environment through observation of an expert agent, using inverse reinforcement learning. The first step to make any inference about the norms applied in the environment, is to retrieve a policy capable of minimizing norm violation and preferably, with low policy loss. We aim to test using different available inverse reinforcement learning algorithms, prioritizing polices that do not violate any norm of the environment.

3. Build an environment for experiments that is scalable in size, number of agents and number of norms. To properly test our approach, we must design an environment to test its ability to consistently retrieve a near-optimal policy, and the scalability of the system, regarding agents, norms and state-space. In Chapter 6, we propose a simple scenario for early tests.

4. Design methods to extract the environment features that are used for each norm. Each norm is assigned to features of the environment. We aim to define which feature defines the violation of each norm. Furthermore, we aim to estimate the sanction of violating each norm. In Section 3.3, we detail our preliminary ideas on how to do so.

5. Experiment the developed approaches in other scenarios and compare when using different inverse reinforcement learning algorithms. Once we are satisfied with the approach results in our proposed scenario, we aim to use different scenarios of the literature, such as the traffic scenario [14].

3.2 Acting in a normative environment through observation

In this section, we introduce the problem of retrieving a policy in the stochastic scenario by observing other agents behave in such environment. First, we give a brief description of the new society scenario. Then, we provide a formal specification of the problem, as a reinforcement learning problem. Finally, we detail how we evaluate answers for this problem.

3.2.1 Scenario

Suppose a norm driven society where its members follow a rule of predefined norms, interacting with other members. Agents who follow the norms are positively rewarded and agents who disrespect the norms are reprehended by the rest of the community, leading to a negative reward.

We introduce a new agent to this society with no knowledge about the norms being enforced by such society. By just observing other agents, the new agent must be able to infer how to act in this society. If the agent correctly learns how to behave in such environment, it will violate any norm. thus, no sanction will be applied for the agent we introduced.

This scenario could use a set of modifications to increase its complexity. A more complex experiment could involve norms that the members of the community break and are not reprehended.

3.2.2 Problem specification

We model the environment as an MDP $M = \langle S, A, p(s'|s,a), \gamma, R \rangle$. This MDP represents a society, regulated by norms. The agent we aim to introduce in this MDP does not know the reward function R, and in a more extreme case. The scenario we described relies on having multiple agents, thus our MDP will have multiple agents acting aside from the one we must introduce. These agents will be following policies optimizing their gains in the environment, avoiding violating any norms. Therefore, our MDP must contain norms $\Omega = \{\omega_0, ..., \omega_n\}$ [40] . We define a norm ω as a tuple $\langle v, t_d, t_a, t_e \rangle$, with v being either:

- $O_{\alpha:\rho}\varphi \circ \Gamma$ (an obligation),
- $P_{\alpha:\rho}\varphi \circ \Gamma$ (a permission), or
- $F_{\alpha:\rho}\varphi \circ \Gamma$ (a prohibition),

where α, ρ are Γ terms, is a first-order atomic formula with constraints; t_d, t_a, t_e are, respectively, the time when v was introduced, when v becomes active and when v expires. Then, we define the society as an MDP, where actions can have stochastic behavior, extended as follows:

- A set of all possible states S.
- A set of n features $\mathbf{f}$ used to represent each state.
- A set of m agents Ag that are acting in the environment.
- A transition function $p(s'|s,a)$ that defines the mechanics of the environment.
- A set of n norms Ω that define how the agents must act.
- A set of punishments P that are applied when one of the norms is violated. $P(s,a)$ returns the sanction applied when executing action a in state s.
- A set of observations ς, retrieved from an agent from set Ag.
- A new agent I introduced to the environment.

This is very similar to Fagundes et.al. [14], approach to describe a Normative Markov Decision Process (NMDP) [14]. We choose this representation as it is simpler, specifically about the norms, as in Fagundes et. al. framework, a norm enforcement can alter the outcome of actions.

Given the definition of the environment, our agent I must learn a reward function R that describes the influence of set of norms N in the environmental reward function, using observation

ς (execution traces (s, a) or a policy) of the set of agents A. After learning the reward function R, we must train the agent with a reinforcement learning algorithm. Using this formalism, we define the problem of learning how to act in a normative environment through observations. We call this Normative Inverse Reinforcement Learning problem.

Definition 4 (Normative Inverse Reinforcement Learning Problem) *Given an MDP $M = \langle S, A, p(s'|s, a), \gamma \rangle$, with (possibly unknown) reward function R, a set of agents Ag, following a set of norms Ω in such MDP, and a set of observations ς, the solution to this Normative Inverse Reinforcement Learning problem is one or more policies $\overline{\pi}$ that minimizes the number of norms violated, avoiding punishments P. The quality of such policies are measured by introducing agent I that follows one of these policies.*

Given the problem of Definition 4, we can apply any IRL algorithm (such as the ones we detailed in Chapter 2, Section 2.5) to retrieve an estimate of the reward function that the observed agent is following in the set of observations ς. After estimating R, we feed the estimated reward function to a reinforcement learning algorithm, such as Q-Learning using ϵ-greedy exploration policy. Thus, we compute a policy $\overline{\pi}$ from the learned reward function, creating an agent that can now be inserted in the environment. We insert this new agent with the computed policy and measure how well he does in the normative environment. With this approach, we can solve the Normative IRL problem, however we must define ways to measure the quality of the policy found. We illustrate this approach in Figure 3.1.

3.2.3 Measuring the quality of the policy

Reinforcement learning algorithms generally measure the quality of the computed policy with the accumulated reward for solving a task. If the optimal policy is known, we can measure the quality of a policy by calculating the policy loss between the computed policy and the optimal policy. To measure the quality of our computed policy in the Normative IRL problem, we propose three different metrics: norm compliance efficiency, policy approximation, loss comparison.

The first metric, accumulated sanctions, measure the accumulated penalty through t steps of executing the agent in the MDP environment, following the computed policy $\overline{\pi}$. Thus, the accumulated sanction is given by:

$$P(\overline{\pi})_{i,t} = \sum_{i}^{t} P(s_t, \overline{\pi}(s_t)) \tag{3.1}$$

where i and t are the interval in which we sample penalties (usually t is the current or last state and i the first state). Equation 3.1 defines when a computed policy more efficiently complies with the norms than another policy.

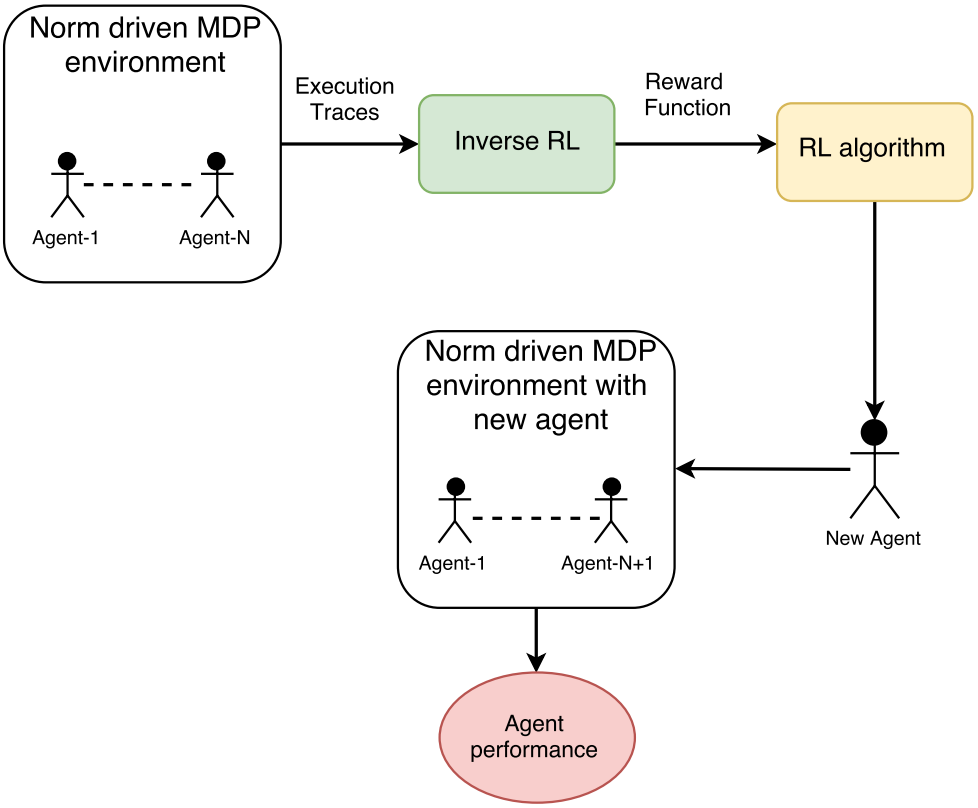


Figure 3.1: Approach overview.

Definition 5 (Norm compliance efficiency) *Given two Normative IRL problem solutions for a deterministic MDP, $\overline{\pi_1}$ and $\overline{\pi_2}$, $\overline{\pi_1}$ is a more **norm compliant solution** than $\overline{\pi_2}$, if:*

$$P(\overline{\pi_1})_{i,t} < P(\overline{\pi_2})_{i,t} \tag{3.2}$$

*when the interval $[i, t]$ contains all states to achieve the desired goal, starting from the initial. A solution **dominates** other solutions when $P(\overline{\pi_1})_{i,t} < P(\overline{\pi_2})_{i,t}$ for **any** interval $[i, t]$. Finally, we consider a solution $\overline{\pi}$ complete norm compliant if and only if, for all $s \in S$*

$$P(s, \overline{\pi}(s)) = 0 \tag{3.3}$$

Norm compliance efficiency alone is incapable of estimating how efficiently a task is solved. Thus, a norm compliant solution may still be uninteresting. We must define a measure to see if the solution solves the task efficiently. Usually we would use the accumulated reward, but since we do not know the true reward function of the environment, and only have an estimation (which can be very poor), we use the observation ς to estimate efficiency. The observations are a pair of state-actions (s_i, a_i). Since we assume that the observations comes from expert agents, we use these observations as an approximation of the optimal policy. Thus, to verify if our policy is efficient,

we verify the how much the observations ς describes our policy, similar to the concept used in the MaxEnt algorithm and the MLIRL algorithm.

Definition 6 (Policy approximation) *Given two Normative IRL problem solutions, $\overline{\pi_1}$ and $\overline{\pi_2}$ and a set of observations ς, $\overline{\pi_1}$* ***approximates more the original policy*** *than $\overline{\pi_2}$, if:*

$$\sum_{s \in S} I_{\varsigma}((s, \overline{\pi_1}(s)) > \sum_{s \in S} I_{\varsigma}((s, \overline{\pi_2}(s)) \tag{3.4}$$

where I is the indicator function, which checks if an element belongs to a set, in this case the observations ς.

Finally, if the ground truth reward function of the model is known (the reward function considering the norm penalties), we can use it to test our computed reward function. We assume that both rewards are represented by features with assigned weights, and rounded so the highest possible reward is 1. Then, we can measure the reward loss using any loss function.

Definition 7 (Loss Comparison) *Given a Normative IRL problem solution, $\overline{\pi}$ and the true reward function R, which we can train a Q-Learning agent, we measure the loss functions:*

$$E(s)_{\overline{\pi}} = \frac{(\max_a Q(s,a) - \max_a \overline{Q}(s,a))^2}{2} \tag{3.5}$$

where Q is Q-Value function learned using the true reward function, and $\overline{Q}$ learned using the computed reward function. Thus, given two solutions, $\overline{\pi_1}$ and $\overline{\pi_2}$, $\overline{\pi_1}$ is ***closer to the real solution*** *than $\overline{\pi_2}$, if:*

$$\sum_{s \in S} E(s)\overline{\pi_1} > \sum_{s \in S} E(s)\overline{\pi_2} \tag{3.6}$$

With these definitions, we can properly evaluate the solutions we find when solving the Normative IRL problem. These are only the metrics we have defined for this proposal, as the metrics used as this work develops can change.

3.3 Estimating Sanction values and features

We have defined how we can find and rank solutions to the Normative IRL problem in Definition 4. However, solving the Normative IRL problem only gives us a learned policy that avoids violating norms by copying observed behavior. Such policy provides no explicit information about the norms of the environment or the sanction being applied when violating them. Given the recent trend of scrutable AI [24], we aim to be able to explain, in normative terms, why an agent behaves the way it does. Thus, we propose the problem of estimating which norms are being sanctioned in the environment, and call this problem norm estimation. To solve such problem, we propose that we need additional information of the environment. In this case, the reward function $R(s)$.

We assume that the agents Ag in the environment are trained using standard reinforcement learning techniques, such as Q-Learning, using a reward function $R(s)$ to achieve an optimal policy π^*. However, this numeric reward is independent of the *sanctions* applied when violating a prohibition norm. Thus, these agents are trained taking into consideration the sanctions of violating prohibitions, $R(s)_\Omega$. Here, the agent learns an optimal policy π^*_Ω, in which the agent considers the sanctions of violating the norms when computing such policy. Using inverse reinforcement learning, we learn an approximation of $R(s)_\Omega$ ($\overline{R}(s)_\Omega$), and an approximation of π^*_Ω ($\overline{\pi}$ by solving the Normative Inverse Reinforcement learning problem, observing an agent following π^*_Ω. Thus, we define the problem of norm estimation as follows.

Definition 8 (Norm estimation) *Given a Normative inverse reinforcement learning problem, a solution $\overline{\pi}$ for such problem, a reward $R(s)$ function of the MDP of the environment (therefore a policy π_1 that can be computed from $R(s)$) , which does not include the norm sanctions from Ω, the solution to the norm estimation problem is a set of norms $\overline{\Omega}$ that estimates the norms being sanctioned in the given MDP Normative IRL problem. The set of norms $\overline{\Omega}$ only contains violations that occur when following π_1, and not occur when following $\overline{\pi}$.*

To solve this problem, we propose estimating the sanction values by comparing $\overline{R}(s)_\omega$ with the environment reward function $R(s)$. The intuition here is that by training an agent with the reward function $R(s)$, we can compute the Q-Values Q of the MDP without the norm sanction. Thus, we can compare the estimated Q-Values $\overline{Q}$, computed using the approximated policy $\overline{\pi}$, with the Q-Values Q. Since both rewards are rounded to be in the interval of $[0, 1]$, if the difference between two Q-Values is greater than a threshold ϵ, we can assume that there is a norm being violated in that Q-Value. Furthermore, after identifying Q-Values with large differences, we can use the features of the state to determine what features are linked to the violation of a norm. Suppose that the state s has three binary features $f_1 = 1, f_2 = 1, f_3 = 1$, and a large difference between the computed Q-Value of the two policies. If a similar state s', has features $f_1 = 1, f_2 = 1, f_3 = 0$, we can assume that the punishment is tied to feature f_3. That is a very early and simple approach, as we obtain more detailed results, we will further develop this approach.

4. ACTIVITIES AND WORK PLAN

We now outline the activities and work plan that we aim to carry out throughout the next two years of our Ph.D. program at PUCRS, describing the work plan detailing the main activities for each semester. We do not set aside a semester for split-side doctoral program, but we consider this as a possibility.

Table 4.2 shows the execution of each activity according to the semester of the next four years.

1. **Survey literature:** this is a continuous activity that we will be doing through the entire program to be aware of new algorithms of inverse reinforcement learning;
2. **Implement our approach:** our current approach is still in the early stages. We must further develop strategies to decide which features are significant to each norm;
3. **Compute the estimate of a norm sanction:** our final goal with our approach is to compute an estimate of each norm sanction when such norm is violated. We aim to verify if this is possible and how much domain information is needed to achieve this;
4. **Further develop the test scenario:** our current test scenario is too simple, even if scalable. We must add intelligent agents to the system and further develop the multi-agent aspect;
5. **Survey others scenarios to test our approach:** we must survey the literature and search other interesting scenarios to properly validate our approach;
6. **Submit papers related to the proposal**: as we further implement our approach, we aim to submit our work to congresses and workshops (e.g., AAAI, AAMAS, PAL) in order to validate our work and receive valuable feedback from the scientific community.
7. **Write the Ph.D. thesis:** as part of the Ph.D. program schedule, we must write the thesis for submission; and
8. **Defend the Ph.D. thesis:** present our thesis to the committee;

	2018		2019	
Activity Number	First Half	Second Half	First Half	Second Half
1	X	X	X	X
2	X	X		
3	X	X		
4	X	X		
5		X	X	
6		X	X	X
7			X	X
8				X

Table 4.1: Activities and work plan for the next two years.

5. RELATED WORK

In this chapter, we survey literature related to our proposal. First we comment on an approach for choosing actions in a normative environment with uncertainty. Second, we survey recent work on norm emergence with reinforcement learning. Third, we discuss inverse reinforcement learning applications. Finally, we discuss ethical agents using reinforcement learning.

5.1 Choosing actions in normative environments with uncertainty

When acting in a normative environment, a agent must balance the desire to achieve a desired goals or to comply with the given norms. In [17], the authors addresses the question of acting in a environment where there is uncertainty regarding both the outcome of the action and if there is someone monitoring and enforcing the environment norms. Similar to our proposed approach, this work deals with stochastic environment and learn how to behave in a normative environment using reinforcement learning algorithms. The authors detail a parking environment where the agent must park following parking norms. In our approach we do not give to the agent the reward function, while in this work is explicitly defined what are the rewards for complying and disobeying one of the norms. We aim to discover the reward of complying or disobeying norms by observing agents that are already in the environment. Furthermore, we aim to use multi-agent scenarios.

5.2 Emergence of Social Norms

We proposed a scenario where we defined a set of norms being followed by the active agents. A more organic way to introduce norms is by Norm emergence. In [44], Yu et al., design a complex social network of agents where each agent interact with its neighbors, observing their actions but not the payoff of each action. The objective is to a social norm emerge as the agents choose to follow it based on the payoff. The authors use a collective reinforcement learning algorithm to learn a policy applied to all agents. As a first implementation we do not aim to use norm emergence, but to bootstrap norms in our environment. However, some techniques of norm emergence could be used to create a more organic environment to introduce our new agent.

5.3 Inverse reinforcement learning

An interesting application of inverse reinforcement learning is the use to teach a robot to navigate in crowds [41]. The authors use IRL algorithms to learn the reward function necessary to imitate the simulation of a human walking in a crowd. Using a simulated environment, the authors analyze the impact of changing features of the environment, in the final policy generated. Although

the scenario is a social one, there is no compromise with learning the set of norms. Additionally, the environment does not account the crowd as agents, making it a single agent environment.

5.4 Ethical Decision Making

Emerging AI system will be a part of human decision making even more as the technology progresses [2]. In [2], David et al., approaches the idea of using reinforcement learning algorithms as framework for ethical decision making. The authors formalize the ethical learning and decision-making problem as solving a partially observable Markov decision process (POMDP). Differently from our work, the authors build a reward function to induce the agent to follow ethical norms. In our approach, we aim to derive the reward function from agents following norms, where there is no explicit reward function.

6. PRELIMINARY RESULTS

In this section, we show our preliminary results using our approach for learning how to act in normative environments. The results are fairly simple, we intend to extend the experiments for the presentation of this proposal. First, we discuss the proposed scenario. Then we show our preliminary results.

6.1 Proposed scenario

For early experiments of our approach, we propose a simple, scalable scenario. The scenario consists of travelling grid, where trucks must retrieve and deliver cargo to specific points in the grid. In some sections of this grid, there are bridges, where a loaded truck must not cross. If the truck crosses the bridge while loaded, the truck is fined a sanction. The drivers must consider if crossing the bridge and receiving a sanction is more optimized than taking other paths.

To apply IRL techniques in this domain, we must model the scenario as an MDP. We model the map as connected graph, where the nodes can be a standard location or a bridge. The transition function is defined as connections of the graph. Furthermore, we use the following features to represent the internal state of an agent:

1. f_1, an integer feature that represents the current node;
2. f_2, a binary feature to describe if the agent is loaded with cargo;
3. f_3, a binary feature to signalize if the agent is currently on a bridge;
4. f_4, an integer feature signalizing that a delivery was completed;
5. f_5, an integer feature enumerating the number of available deliveries.

We illustrate this scenario in Figure 6.1, where an example of the delivery scenario is shown. Each node is one of the possible locations, and the yellow squared nodes are bridges. The green node A is the pick-up location where the truck can pick-up deliveries, and the red node is delivery location, where the truck unloads the cargo. An episode of such scenario ends with the agent unloading all the available cargo (thus, this is a terminal state).

The reward function of the environment is straightforward. When finishing every delivery (arriving at a terminal state), the reward signal is 1 (the highest reward in this scenario). Every other state is rewarded -0.1. To model the sanction for crossing a bridge while loaded, we use a norm ω, which applies a sanction of -0.9 if the agent is a bridge while loaded. Thus, we verify the features f_2 and f_3 to apply the sanction of this norm.

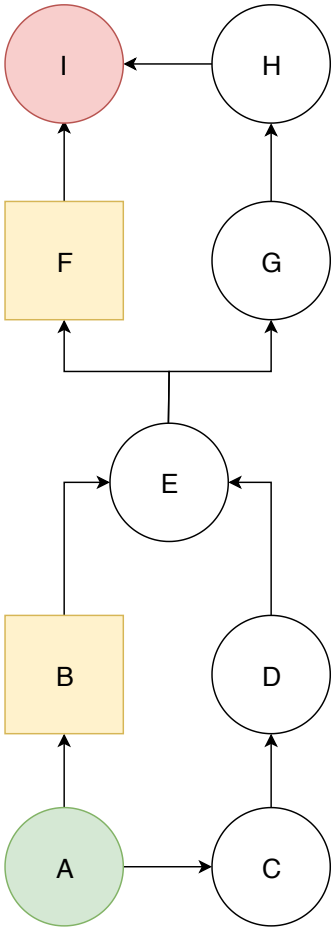


Figure 6.1: Delivery scenario

6.2 Preliminary experiments

To experiment our approach, we use three algorithms: standard inverse reinforcement learning; maximum entropy inverse reinforcement learning; and maximum entropy deep inverse reinforcement learning. We use the scenario illustrated in Figure 6.1, setting the agent to start at the node E with one delivery to make. We evaluate the solutions found, using the definitions of Chapter 3.

In Table 6.1, we show the results when applying the three different algorithms and evaluating them using our defined metrics: Norm compliance efficiency (Definition 5); Policy approximation (Definition 6); and Loss Comparison (Definition 7). The discount factor γ is set to 0.8 for all algorithms. The MaxEnt and DeepIRL algorithm are trained using 100 epochs. All algorithms were given full observability of the policy being followed by the expert (the optimal policy of the environment). As we can see, the standard IRL algorithm was unable to achieve a policy that respect the domain norm, violating it two times. Both MaxEnt and Deep IRL algorithms were able to achieve a norm

compliant policy. There is a small deviation from the optimal policy in the MaxEnt algorithm, only one action from the set of 19 states of the environment.

Algorithms	Norm compliance efficiency	Policy Approximation	Loss Comparison
IRL	-1.8(2)	4/19	10.70333
MaxEnt	0	1/19	5.12678
DeepIRL	0	1/19	4.80864

Table 6.1: Results in the delivery scenario.

Our experiments are still too simple, but we plan to provide more meaningful experiments during the presentation of our proposal. Furthermore, we must compute the norm estimation using the extracted reward function. We will extend this chapter as the implementation advances.

7. CONCLUSION

In this document, we introduced our thesis proposal, in which we developed an approach to learn how to act in normative multi-agent systems using only the observations of an agent that already belongs to the environment. The main contributions of this proposal are: first, the formal definition of the problem of acting in a normative environment through observations; second, the formal definition of the problem of computing which feature of the state are assigned to each norm, and the sanction for disobeying them; third, a simple scalable scenario to experiment our approach. We performed early experiments using 3 inverse reinforcement learning algorithms, aiming to introduce an agent capable of acting in the normative environment without violating any norms.

For the remaining two years of our Ph.D., we aim to fully develop our approach for measuring the sanction of a norm, and apply a set of refinements to our current scenario. First, we must fully develop our test scenario to fully enable multi-agent interactions. Second, we aim to develop agents in the scenario that function as observers, in which violation of the norms can only be sanctioned if they are witnessed by one the observers. Finally, we aim to test our approach in different scenarios, to properly validate the ability of our approach of learning complex reward functions.

BIBLIOGRAPHY

[1] Abbeel, P.; Ng, A. Y. "Apprenticeship learning via inverse reinforcement learning". In: Proceedings of the Twenty-first International Conference on Machine Learning, 2004, pp. 1–.

[2] Abel, D.; MacGlashan, J.; Littman, M. L. "Reinforcement learning as a framework for ethical decision making." In: AAAI Workshop: AI, Ethics, and Society, Bonet, B.; Koenig, S.; Kuipers, B.; Nourbakhsh, I. R.; Russell, S.; Vardi, M. Y.; Walsh, T. (Editors), 2016.

[3] Amado, L.; Meneguzzi, F. "Q-table compression for reinforcement learning", *The Knowledge Engineering Review*, vol. 33, 2018, pp. e22.

[4] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. "Goal recognition in latent space". In: Proceedings of the 31st International Joint Conference on Neural Networks, 2018.

[5] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. "Lstm-based goal recognition in latent space". In: In ICML / IJCAI / AAMAS 2018 Workshop on Planning and Learning (PAL-18), 2018.

[6] Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; Meneguzzi, F. "An lstm-based approach for goal recognition in latent space". In: In the AAAI 2019 Workshop on Plan, Activity, and Intent Recognition, Honolulu, USA., 2019.

[7] Bellman, R. "A Markovian Decision Process", *Indiana Univ. Math. J.*, vol. 6, 1957, pp. 679–684.

[8] Bertsekas, D. P.; Tsitsiklis, J. N. "Neuro-Dynamic Programming". Athena Scientific, 1996, 1st ed..

[9] Bhadeshia, H. K. D. H. "Neural networks in materials science", *ISIJ INTERNATIONAL*, vol. 39, 1999, pp. 966–979.

[10] Bishop, C. M. "Pattern Recognition and Machine Learning (Information Science and Statistics)". Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[11] Boella, G.; van der Torre, L.; Verhagen, H. "Introduction to the special issue on normative multiagent systems", *Autonomous Agents and Multi-Agent Systems*, vol. 17–1, 2008, pp. 1–10.

[12] Choi, J.; Kim, K.-E. "Bayesian nonparametric feature construction for inverse reinforcement learning". In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, 2013, pp. 1287–1293.

[13] Doya, K.; Sejnowski, T. J. "A novel reinforcement model of birdsong vocalization learning". In: NIPS, 1994.

[14] Fagundes, M. S.; Ossowski, S.; Cerquides, J.; Noriega, P. "Design and evaluation of norm-aware agents based on normative markov decision processes", *Int. J. Approx. Reasoning*, vol. 78, 2016, pp. 33–61.

[15] Jaynes, E. T. "Information theory and statistical mechanics", *Phys. Rev.*, vol. 106–4, May 1957, pp. 620–630.

[16] Levine, S.; Popovic, Z.; Koltun, V. "Nonlinear inverse reinforcement learning with gaussian processes". In: *Advances in Neural Information Processing Systems 24*, Shawe-Taylor, J.; Zemel, R. S.; Bartlett, P. L.; Pereira, F.; Weinberger, K. Q. (Editors), Curran Associates, Inc., 2011, pp. 19–27.

[17] Li, J.; Meneguzzi, F.; Fagundes, M. S.; Logan, B. "Reinforcement learning of normative monitoring intensities". In: 18th International Workshop on Coordination, Organizations, Institutions, and Norms, 2015, pp. 1–15.

[18] Menache, I.; Mannor, S.; Shimkin, N. "Basis function adaptation in temporal difference reinforcement learning", *Annals of Operations Research*, vol. 134–1, Feb 2005, pp. 215–238.

[19] Meneguzzi, F.; Luck, M. "Norm-based behaviour modification in bdi agents". In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, 2009, pp. 177–184.

[20] Mitchell, T. M. "Machine Learning". New York, NY, USA: McGraw-Hill, Inc., 1997, 1 ed..

[21] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. "Playing atari with deep reinforcement learning", 2013, cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[22] Mohri, M.; Rostamizadeh, A.; Talwalkar, A. "Foundations of Machine Learning". The MIT Press, 2012.

[23] Monica C . Vroman, M. L. L. "Maximum likelihood inverse reinforcement learning", 2014.

[24] Monroe, D. "Ai, explain yourself", *Commun. ACM*, vol. 61–11, Oct 2018, pp. 11–13.

[25] Montague, P. R.; Dayan, P.; Person, C.; Sejnowski, T. J. "Bee foraging in uncertain environments using predictive hebbian learning", *Nature*, vol. 377–October, 1995, pp. 725–728.

[26] Ng, A. Y.; Russell, S. J. "Algorithms for inverse reinforcement learning". In: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 663–670.

[27] Puterman, M. L. "Markov Decision Processes: Discrete Stochastic Dynamic Programming". New York, NY, USA: John Wiley & Sons, Inc., 1994, 1st ed..

[28] Ratliff, N. D.; Bagnell, J. A.; Zinkevich, M. A. "Maximum margin planning". In: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 729–736.

[29] Rummery, G. A.; Niranjan, M. "On-line q-learning using connectionist systems", Technical Report, 1994.

[30] Russell, S. "Learning agents for uncertain environments (extended abstract)". In: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, 1998, pp. 101–103.

[31] Russell, S. J.; Norvig, P. "Artificial Intelligence: A Modern Approach". Pearson Education, 2003, 2 ed..

[32] Savarimuthu, B. T. R.; Cranefield, S.; Purvis, M. A.; Purvis, M. K. "Identifying prohibition norms in agent societies", *Artificial Intelligence and Law*, vol. 21–1, 2013, pp. 1–46.

[33] Schmajuk, N. A.; Zanutto, B. S. "Escape, avoidance, and imitation: A neural network approach", *Adaptive Behavior*, vol. 6–1, 1997, pp. 63–129, https://doi.org/10.1177/105971239700600103.

[34] Scott, J.; Marshall, G. "A Dictionary of Sociology". Oxford University Press, 2009.

[35] Simon, P. "Too big to ignore: the business case for big data". New Delhi: Wiley, 2013.

[36] Singh, S.; Jaakkola, T.; Littman, M. L.; Szepesvári, C. "Convergence results for single-step on-policyreinforcement-learning algorithms", *Mach. Learn.*, vol. 38–3, Mar 2000, pp. 287–308.

[37] Sutton, R. S. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: Advances in Neural Information Processing Systems 8, 1996, pp. 1038–1044.

[38] Sutton, R. S.; Barto, A. G. "Introduction to Reinforcement Learning". Cambridge, MA, USA: MIT Press, 1998, 1st ed..

[39] Tijsma, A.; Drugan, M.; Wiering, M. "Comparing exploration strategies for q-learning in random stochastic mazes", 2016.

[40] Vasconcelos, W. W.; Kollingbaum, M. J.; Norman, T. J. "Normative conflict resolution in multi-agent systems", *Autonomous Agents and Multi-Agent Systems*, vol. 19–2, Oct 2009, pp. 124–152.

[41] Vasquez, D.; Okal, B.; Arras, K. O. "Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison". In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014, 2014, pp. 1341–1346.

[42] Watkins, C. J. C. H.; Dayan, P. "Technical note: q-learning", *Mach. Learn.*, vol. 8–3-4, May 1992, pp. 279–292.

[43] Wulfmeier, M.; Ondruska, P.; Posner, I. "Maximum entropy deep inverse reinforcement learning", *CoRR*, vol. abs/1507.04888, 2015.

[44] Yu, C.; Zhang, M.; Ren, F.; Luo, X. "Emergence of social norms through collective learning in networked agent societies". In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, 2013, pp. 475–482.

[45] Ziebart, B. D.; Maas, A.; Bagnell, J. A.; Dey, A. K. "Maximum entropy inverse reinforcement learning". In: Proc. AAAI, 2008, pp. 1433–1438.