Universidad Politécnica de Catalunya

TFG: Computacion

Comparación de la explicación obtenida con modelos de explicabilidad para diferentes métodos de Reinforcement Learning.

Autores
Javier Rivera Hernandez

Directores
Sergio Álvarez Napagao
Javier Vázquez Salceda
Lorenza Pinto Paola (GEP)

June 21, 2022

Abstract

The growing use of AI technologies to provide products and services to humans has raised concerns on how these technologies take some decisions that may heavily impact the people's lifes (such as the granting of a bank loan, the health diagnosis from some medical imagery, or the access to a job position).

In order to address those concerns, many institutions in Europe (leaded by the European Commission) are promoting the creation of Trustworthy AI technologies. One of the ways to make theses technologies trustworthy is by making all results provided by them understandable by humans. Explainability is a way to provide explanations on the decisions made by a given AI model or algorithm, and it is specially used in technologies such as Deep Learning and Reinforcement Learning.

This project is part of an ongoing research on the explainability of agents trained through Reinforcement Learning. The main objective is to check if, given a fixed environment and explainability model, different types of agent training algorithms produce different explanations (about their behavior). This hypothesis will be validated or refuted by training agents using different reinforcement learning methods and comparing their policies, based on metrics defined in the project, using the same explainability model.

Resumen

El creciente uso de las tecnologías de Inteligencia Artificial para proveer productos y servicios destinados a humanos ha aumentado el interes en como estas tecnologías toman decisiones que pueden tener un grave impacto en la vida de las personas (como por ejemplo la concesión de un prestamo bancario, el diagnóstico sanitario a partir de imagenes medicas, o el acceso a un puesto de trabajo).

Con la intención de abordar estos problemas, muchas instituciones en Europa (lideradas por la Comisión Europea) estan promoviendo la creación de tecnologías de IA de confianza (TRustworthy AI). Una de las formas de hacer estas tecnologías confiables consiste en hacer que todos los resultados que proveen sean entendibles por los humanos. La explicabilidad es una manera de proveer explicaciones a las decisiones tomadas por un modelo de IA o un algoritmo, y son especialmente usadas en las tecnologías de Deep Learning y Reinfocement Learning.

Este proyecto forma parte de la investigación en curso acerca de la explicabilidad de agentes entrenados con Reinforcement Learning. El objetivo principal es comprobar si, dado un entorno fijo y un método de explicabilidad, diferentes tipos de algoritmos de entrenamiento producen explaciones diferentes (acerca de su comportamiento). Esta hipótesis será validada o refutada entrenando agentes con diferentes métodos de Reinforcement Learning y comparando sus políticas, basandose en métricas definidas en el proyecto, usando un modelo de explicabilidad común.

Resum

El creixent ús de les tecnologies d'Intel·ligència Artificial per proveir productes i serveis destinats a humans ha augmentat l'interès en com aquestes tecnologies prenen decisions que poden tenir un greu impacte a la vida de les persones (com per exemple la concessió d'un préstec bancari, el diagnòstic sanitari a partir d'imatges mèdiques o l'accés a un lloc de treball).

Amb la intenció d'abordar aquests problemes, moltes institucions a Europa (liderades per la Comissió Europea) estan promovent la creació de tecnologies de confiança de IA (TRustworthy AI). Una de les maneres de fer aquestes tecnologies fiables consisteix a fer que tots els resultats que proveeixen siguin entenedors pels humans. L'explicabilitat és una manera de proveir explicacions a les decisions preses per un model d'IA o un algoritme, i són especialment utilitzades en les tecnologies de Deep Learning i Reinfocement Learning.

Aquest projecte forma part de la investigació en curs sobre l´explicabilitat d´agents entrenats amb Reinforcement Learning. L'objectiu principal és comprovar si, atès un entorn fix i un mètode d'explicabilitat, diferents tipus d'algorismes d'entrenament produeixen explacions diferents (sobre el comportament). Aquesta hipòtesi serà validada o refutada entrenant agents amb diferents mètodes de Reinforcement Learning i comparant les seves polítiques, basant-se en mètriques definides al projecte, usant un model d'explicabilitat comú.

Contents

Chapter 1

Introduction

1.1 Description of the problem and its context

1.1.1 Explainability of Reinforcement Learning methods

Explainability is the application of methods and techniques to Artificial Intelligence models and algorithms with the aim of making the results of these models/algorithms understandable by humans. This makes it possible to obtain explanations on why an AI model/algorithm has arrived at a certain decision, allowing humans to confirm existing knowledge, validate an AI model, or generate new assumptions.

Nowadays this is a hot topic, since some political entities as the European Commission have given attention to the creation of a regulatory framework for AI technologies that make decisions without an explanation, decisions that may affect the life of European citizens.

In particular, explainability is more crucial in Reinforcement Learning[1], because it is a kind of AI models that learns without any human intervention.

This project will explore the effect of using different Reinforcement Learning algorithms in the explanations obtained by a fixed explainability method in a fixed environment.

1.1.2 Context

This is a thesis work for the Bachelor degree in Computer Engineering at the Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya - BarcelonaTECH (UPC). The work is made with the collaboration of the High Performance Artificial Intelligence group (HPAI) from the Barcelona Supercomputing Center (BSC) and the Knowledge Engineering and Machine Learning Group (KEMLG) within the Intelligent Data sciEnce and Artificial Intelligence Research Center (IDEAI-UPC). The work has Sergio Alvarez Napagao and Javier Vázquez Salceda as advisors.

[1]We provide definitions to some terms such as Machine Learning, Reinforcement Learning or Policies in §2.1.

1.1.3 Problem to be solved

This project is part of the investigations about the explainability of Reinforcement Learning agents. The principal objective is to verify if, given a fixed environment and an explainability method, different agent learning algorithms produce different explanations (about its behavior). This hypothesis will be validated or refuted through the training of agents using varying kinds of Reinforcement Learning methods and comparing the resulting policies. These policies will be compared using metrics defined within the project, using the same explainability model for all the training algorithms.

About the agent learning algorithms, we will explore in the model-free family of algorithms. More precisely, we will choose one of each methods of the two approaches within model-free algorithms (Policy Optimization and Value Optimization).[2]

To run the tests there are different kinds of environments that could be selected for this work, such as ViZDoom [1], Neural MMO [2], Pommerman[3] or SMAC [4]. But we have selected the CartPole environment from OpenAI[3] as this was the environment used in the previous research (see §1.2.2 below), and this will allow us to compare the results of our research with the previous ones.

1.1.4 Stakeholders

The main stakeholders that will have direct interaction with this project will be the author and the advisors of this thesis (Sergio Alvarez Napagao and Javier Vázquez Salceda) who also are researchers in the Artificial Intelligence field. This work will also be used by the scientific community who gets access to the study realized in this project and can use this information to validate or learn about Reinforcement Learning systems and also for those who want to do further studies in this area.

1.2 Justification

1.2.1 Interest on Explainability

As it was previously stated, allowing humans to understand why physical or virtual agents trained with black box methods, (such as Neural Networks), have made a decision, increases the confidence that they have in these agents and therefore the user experience. Moreover, if these agents work in a sensitive field as the medical, the decisions made by these agents are crucial and have strong ethical consequences, because there are human lives in danger. This implies that these decisions must be analyzed and verified, without giving blind confidence to an agent that decides without any kind of justification beyond a mathematical justification. That is why the research in this topic is critical.

1.2.2 Previous Work

This thesis continues the research line of some previous works. One of them is the thesis of Antoni Climent Muñoz [5], which explores different explainability approaches that try to

[2]In §2.3 we discuss the different families of learning algorithms including the model-free ones.
[3]We describe the CartPole environment in more detail in §2.2.1.1.

apply to a Reinforcement Learning agent that plays a computer game called CartPole. This computer game is the environment of the Reinforcement Learning agent. The explainability method selected by Antoni Climent Muñoz consists in creating a MDP of the CartPole environment after discretizing the environment to then obtain predicates that explain the policy learned by the agent.

This thesis will use the explainability method used in [5]. The principal difference that this project will have in comparison with his research is that this thesis will focus on using a broader variety of agent training algorithms since in [5] only Deep Q Network Agent (DQN) is used.

Initially we were going to explore other famillies of Reinforcement Learning methods as Inverse RL or Imitation RL, but finally we decided to explore other families of RL methods due to our incapacity to get games played by an expert of CartPole.

Also, the thesis of Bartomeu Perelló Comas [6] deserves a mention because he extended the research line of explainability applied to Reinforcement Learning methods trying to apply the same explainability methods that will be used in this project with DQN in another environment called ViZDoom.

In order to explore new approaches that enable us to increase our knowledge and understanding of this topic, this thesis will try to analyze how changing the agent training algorithm affects the explication obtained.

1.3 Scope

1.3.1 Objective

The main objective of this thesis is to continue the research on the explainability applied to Reinforcement Learning algorithms. Since the effect of changing the environment and the explainability method has been investigated before, this thesis will focus on exploring how changing the agent learning method change the explication obtained for a fixed environment and explainability method. In order to have a clearer image of the process to handle this objective, it will be split into sub-objectives.

1.3.2 Sub-Objectives

The sub-objectives are the following ones:

- **O1:** Study the state-of-the-art and identify the current methods used in Reinforcement Learning, Neural Networks, and Machine Learning and how to get explainability on each of these fields.

- **O2:** Choose an environment: After researching some possible options, discretize the environment. This sub-objetive finally was simplified to just use CartPole because of its simplicity and to the fact that it is already discretized in [5].

- **O3:** Understanding how the explainability method explained in previous projects works to implement it.

- **O4:** Select two or three agent learning algorithms: After researching about them, select some of them to implement them. In the course of the project after learning about the different families of RL algorithms and having some setbacks, we decided that finally we will choose for each of the two approaches within model-free methods.

- **O5:** Select metrics to compare the policies obtained: Research possible metrics to compare agent policies and select one quantitative and one qualitative.

- **O6:** Finally, after selecting a metric, implementing the explainability method and the learning algorithms, compare the different policies obtained and get conclusions. In the course of the project we have seen that the problem was not to implement the explainability method that was already implemented. Instead, the problem was to integrate the learning methods in the explainability method.

1.3.3 Requirements

Since this is a research project that involves an exploratory process, the functional and non-functional requirements will depend on the initial explorations and decisions of what kind of environment, agent learning algorithms, and explainability method will be chosen.

In §4.2 we explain how we decided which agents will be implemented, and in§4.1 which environment. The explainability method selected is explained in [5].

1.3.4 Risk and Problems

In the course of every project always appear problems or delays in the pre-established planing. So consider them before dealing with them could really make the difference to not lose a lot of time. In the case of this project, there are three main problems:

1.3.4.1 Time

Time is one of the main problems, because of the complexity of the problem and the necessity of learning and understanding properly how to use Reinforcement Learning algorithms. So maybe a huge amount of time is needed to learn about the topic. Another possible issue related to the time is the amount of time that will be expended training the model since the agent has to train multiple times in a non-trivial environment until it obtains the required performance. Also, complications implementing this agent could lead to more delay.

1.3.4.2 Contingencies Implementing

Contingencies Implementing are another kind of problem. Exists the risk of delay due to the appearance of bugs implementing the agents. The possible sources of bugs implementing the agent are, for instance, the interpretation of the rewards or the discretization of the environment.

In the course of the project, the contingencies implementing have appeared integrating the agent in the explainability method and not discretizing the environment, because as it has been said in §4.1, we finally decided to use the discretization and explainability method described in [5].

1.3.4.3 Machine Learning Contingencies

There are some possible contingencies in the inherent nature of Machine Learning. For instance that some complex models might not work, or the precision of the agent is very low, and therefore we do not obtain conclusive results. Another problem that might happen is that although the agent works well and his precision is accurate, the results obtained are not good.

Ultimately, problems have occurred with the hyperparameter tuning process, to get an agent with a good performance that allow the model to have a good exploration of the universe of states and to converge to a optimal policy or value function. Also, to have policies that can be compared properly, we need two policies with a similar behavior, and we had contingencies tuning the hyperparameters to obtain similar policies.

Chapter 2

State of the Art

2.1 Terms and Definitions

In order to better understand this work, this section introduces the reader to some of the main Artificial Intelligence and Reinforcement Learning terms related to the work in this thesis.

2.1.1 Machine Learning

Machine Learning is a family of algorithms within the Artificial Intelligence field that can learn how to perform a task by itself without being explicitly programmed. These algorithms learn through experience by the use of data related to the task that is to be learned.

In the words of Tom M. Mitchell, Machine Learning is: "A computer program is said to *learn* from experience E concerning some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [7]

There are four different approaches to Machine Learning: Supervised Learning, Semi-Supervised Learning, Unsupervised Learning, and Reinforcement Learning, which is the one that we will focus on.

2.1.2 Reinforcement Learning

Reinforcement Learning [9] is a field of Machine Learning which focuses on how intelligent Agents should perform actions to maximize a reward signal obtained from the environment. The objective of Reinforcement Learning is that these Agents discover which actions yield the most reward by trying them out, considering that the actions may not only affect the immediate reward but also the next situation and therefore the subsequent rewards.

In order to achieve the optimal set of actions, the approach is to split the problem into states. These states represent the environment and the actions chosen by the Agent. Each

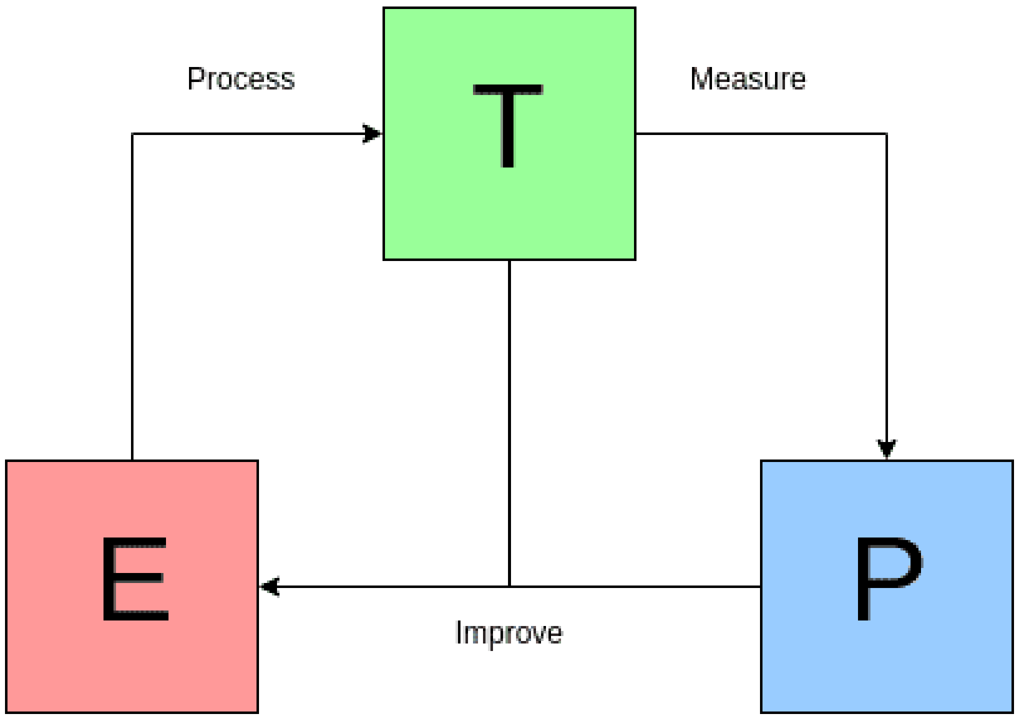


Figure 2.1: Diagram describing Mitchell's definition on ML. Image taken from [8].

time the Agent takes an action, a reward is assigned to it, and a new state representation is given. This is usually specified with a Markov Decision Process (MDP).

The main particularity of Reinforcement Learning with other Machine Learning algorithms is that in Reinforcement Learning we do not have initially a set of observations and their expected outcome, they are obtained through exploration on a trial-and-error basis. So to get a good performance, a balance between exploration and exploitation (of current knowledge) must be found.

Some important definitions of Reinforcement Learning will be explained:

2.1.2.1 Agent

An Agent is an entity that performs actions in a determined environment to resolve a problem. The situation of the Agent in the environment represents the state of the problem.

2.1.2.2 Markov Decision Processes

A Markov Decision Process (MDP) is a representation of the Agent throughout the resolution of the problem in the dynamic environment given. A MDP is represented with a graph

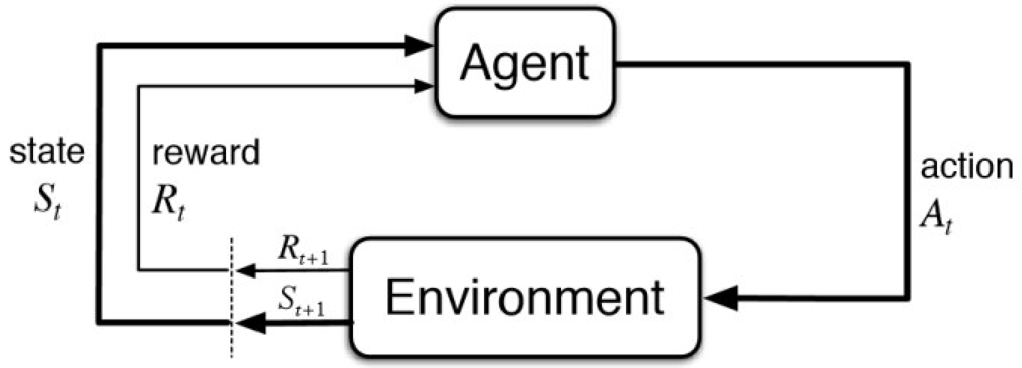


Figure 2.2: Diagram describing the RL cycle. Image taken from [10]

where each node represents a state and each weighted edge represents the probability of transition into a certain state given an action executed by the Agent.

2.1.2.3 Policy

A policy defines the learning agent's way of behaving at a given time. It is the set of stimulus-response rules. [9] Denoted by the π symbol, it is a map π: S $\rightarrow$ A where S is the set of states and A the set of possible actions. It also can be interpreted as the probability of going into a state given an action and an actual state $\pi = P(a|s)$.

2.1.2.4 Reward

The reward is the quantification of how well is performing the agent. At each step where the agent makes a decision, it receives a reward. The total or cumulative reward is the sum of the rewards of each time step until the game is finished. The cumulative reward in a time step t can be expressed as:

$$R_t = \sum_{i=0}^{t} r_i$$

Where i is the time step of the game.

We also want to define the total reward that will be achieved given a time step t and a sequence of previous states, actions, and rewards before the time step t. We will call this reward discounted reward.

In order to prevent infinite rewards in infinite-horizon environments, rewards are discounted by a factor of $\gamma \in$ [0, 1]. This is a mathematical convenience and also has an intuitive explanation. On one hand, it allows the expression to converge to a finite value, but it also explains that current and recent rewards are more important than past ones.

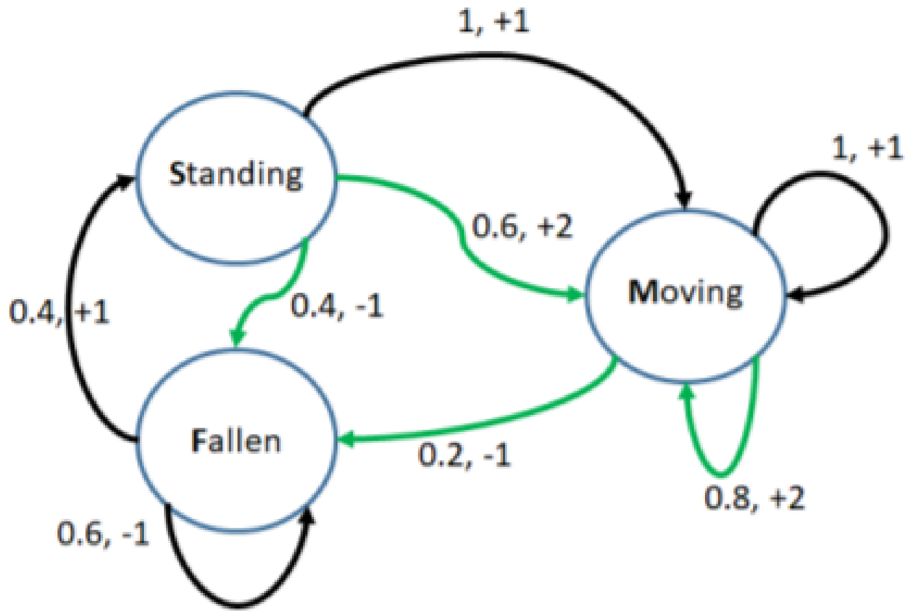


Figure 2.3: Diagram of a MDP example. Image taken from [11]

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+1}$$

The total discounted reward consists on the total discounted reward that will be achieved by the agent during the whole run. It is defined as:

$$R = R_0 = \sum_{i=0}^{\infty} \gamma^i r_i$$

2.1.2.5 Value Function

The Value function of state $s \in S$ associated to policy π is defined as the expected discounted reward that the agent will obtain if it follows policy π starting from state s. Formally:

$$V^{\pi}(s) = E_{\pi}(R | s_0 = s)$$

The optimal value function of state $s \in S$ is defined as the expected discounted reward that the agent will obtain if it follows an optimal starting from state s. Formally:

$$V^*(s) = \max_\pi V^\pi(s)$$

2.1.2.6 Action-Value Function

The Action-Value function of state $s \in S$ and action $a \in A$ asociated to policy π is defined as the expected discounted reward that the agent will obtain if it performs action a starting from state s and follows policy π from that point.

$$Q^\pi(s,a) = E_\pi(R|s_0 = s, a_0 = a)$$

The optimal action-value function of state $s \in S$ and action $a \in A$ is defined as the expected discounted reward that the agent will obtain if it performs action a starting from state s and follows an optimal policy from that point. Formally:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

2.1.2.7 Advantage Function

The Advantage function of state $s \in S$ and action $a \in A$ associated to policy π is defined as the expected increase on the discounted reward with respect to the expected discounted reward of the current state. Formally:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

The optimal advantage function of state $s \in S$ and action $a \in A$ is defined as the expected increase on the discounted reward with respect to the expected discounted reward of the current state. Formally:

$$A^*(s,a) = Q^*(s,a) - V^*(s)$$

2.1.3 Neural Networks

This area of AI tries to mimic the way human neurons work into a computational model. According to J. Schmidhuber: "A standard neural network (NN) consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons. [12]

Each activation is a function, usually a non-linear transformation. The process of learning consists in assigning the correct weights to the connections between neurons in order to make the Neural Network exhibit the desired behavior.

The neurons that get activated through the perception of the environment are called the input layer, whereas the layer that gives a final response to the inputs is the output layer. Those layers between the input and the output layer are the hidden layers.

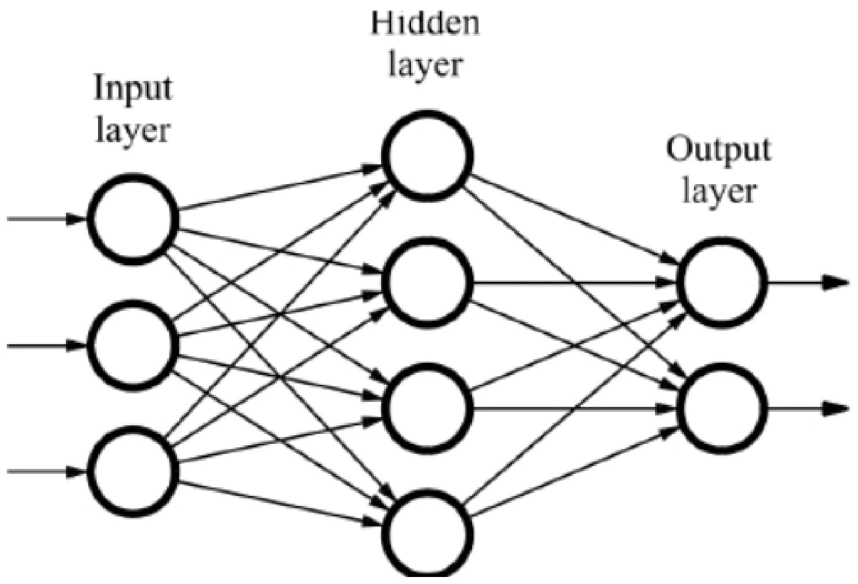


Figure 2.4: Diagram describing a Neural Network. Image taken from [13]

2.2 Environments

Reinforcement Learning has a lot of environments on which the agents could be trained. We will explain some of the most relevant ones included in the OpenAI library Gym.

2.2.1 Gym

Gym is a library of Python implementing a huge variety of environments that aims to provide the user with an easy interface and be a benchmark environment to evaluate the performance of the agents. Some of the environments that it includes are:

2.2.1.1 Cartpole

Cartpole is an environment provided by the Gym library. It consists of an inverted pole attached to a cart that moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 12 degrees from vertical, or the cart moves more than 2.4 units from the center.

The state of the game is represented by a tuple $\langle pos, vel, angle, velAtT\rangle$ where pos represents the cart position, vel is the cart velocity, $angle$ is the pole angle and $velAtT$ is the pole angular velocity.

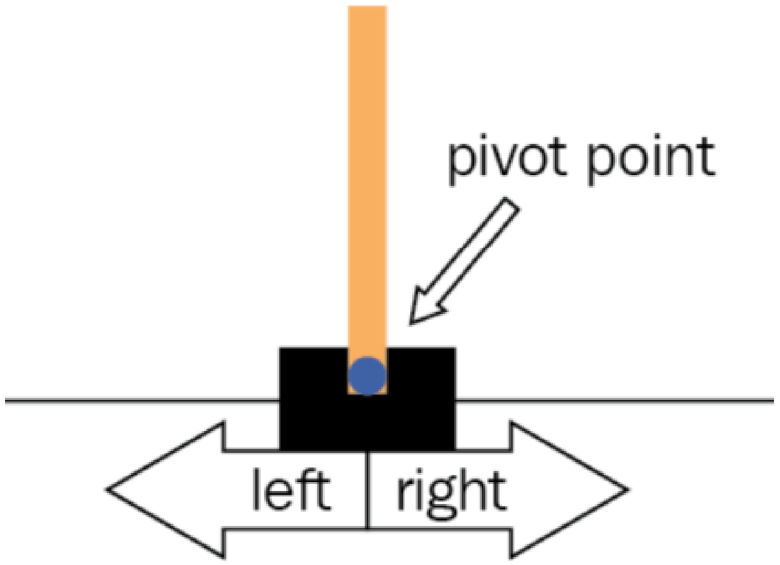


Figure 2.5: Cart Pole standing up []

2.2.1.2 Atari

There is a set of Atari 2600 environment simulated through Stella and the Arcade Learning Environment in the Gym library. The actions space of this environments are the 17 possible actions that you could play in the original controller plus the option of not pushing any button during this step. The observation space may be conformed by the RGB image that is displayed to a human player, a gray-scale version of this image or the 128 bytes of RAM of the console. The rewards depends on the environment selected.

2.3 Types of Agents

A basic classification of RL agents is between Model-Based and Model-Free agents.

Model-Based agents have access to a model of the environment, that means that the agent has a function that predicts state transitions and rewards, whereas the Model-Free agents do not.

Model-Based agents can plan by thinking ahead, seeing what would happen for a range of possible choices, and explicitly deciding between the options. But this usually is not available to the agent, in which case we have to use Model-Free agents.

Although Model-Free agents can not use the potential gains in sample efficiency from using a model, they tend to be easier to implement and tune.

Within the Model-Free agents, we will discern between on-policy and off-policy.

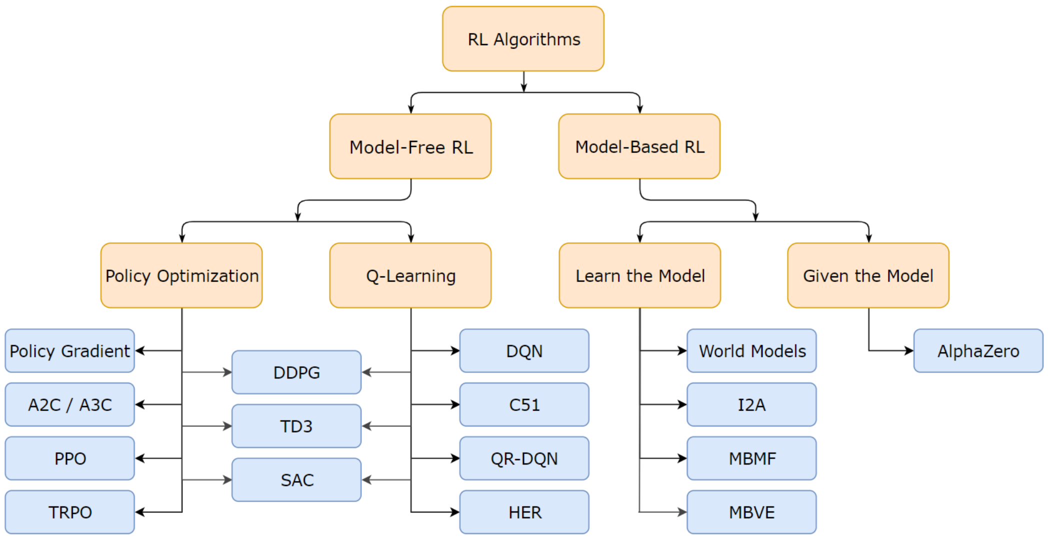


Figure 2.6: A non-exhaustive, but useful taxonomy of algorithms in modern RL [14]

The on-policy agents are those with learning strategies that base the learning procedure on the current policy of the agent. It means that each training step is performed taking into account the most recent policy, but it also means that past experience is hardly applicable to the current step.

The off-policy agents are those with learning strategies where the learning procedure does not depend on the current policy of the agent. It means that past experience can be perfectly used in the current training step, but the information obtained from the environment might not come from the most recent policy.

Now we will explain an example of an on-policy learning algorithm (PPO) and another of an off-policy one (DQN).

2.3.1 DQN

DQN (Deep Q-Network) [15] is a Q-learning off-policy learning algorithm that learns the Q^* function with a DL network using the Bellman optimality equation.

$$Q^*(s,a) = r + \gamma \max_{a'} Q^*(s',a')$$

Where s' is the state we transsition from state s after performing action a.

From the Bellman optimatility equation we can obtain a loss to train the model:

$$L(Q) = (r + \gamma \max_{a'} Q^*(s',a') - Q^*(s,a))^2$$

Once the Q^* function is properly learned, we can compute the policy as a softmax of the values of the Q^* function and sample the actions from that distribution.

2.3.2 PPO

PPO (Proximal Policy Optimization)[16] is an Advantage on-policy learning algorithm. In earlier Policy gradient methods, the objective function was something like:

$$\hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$$

Where θ is the policy parameters, $\hat{A}_t$ is the estimation of the Advantage at time t, $\hat{\mathbb{E}}_t$ is the empirical expectation over timesteps.

In PPO instead, we will use the ratio between the current and the old policy.

$$\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

The ratio will be clipped in order to restrict large policy updates.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

Where ϵ is a hyperparameter, usually 0.1 or 0.2.

Finally, the final objective function combines L^{CLIP}, an squared error loss $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$ and an entropy bonus S:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP} - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

Where c_1 and c_2 are weighting coefficients.

2.3.3 Inverse Reinforcement Learning

IRL (Inverse Reinforcement Learning)[17] is a technique in the field of RL where no reward function is given. Instead, it is inferred from the performances of an expert in the task to be learned. The idea is to mimic this behavior that is often optimal or close to optimal.

The main advantage of this method is that some environments or tasks that we want to learn do not have a known reward function, especially those tasks that consist in mimic human or animal behavior, and this method can work well in these tasks inferring the reward function from real performances. Another problem for which this method can be useful in certain domains where using a straightforward RL approach can not be optimal, such as driving.

On the other hand, this method requires gathering expert performances to train the model, which can be expensive or difficult. In our case, we decided to dismiss this method because we do not have access to an expert on the environment, and also we do not have time to master the environment and play the game ourselves.

2.4 Explainability Methods

2.4.1 Model-Agnostic

Nowadays, some model-agnostic explainability methods are applied to many kinds of ML and DL models. We can differentiate between local and global methods of these model-agnostic techniques.

Local methods are those that try to provide clarifications for a single decision of the model, whereas the global methods try to provide a general explanation that is valid for any instance.

2.4.1.1 LIME

Local interpretable model-agnostic explanations (LIME), as its name says, is a local method. It tries to explain which variables from the input caused the decision taken by the model. This method explains the prediction by replacing the original model f with a local interpretable model g (such as a linear model or a decision tree), derivated from the instance x to be explained. The interpretable model g learns from perturbing the original instance x and generating a set of instances X' where each instance $x' \in X'$ are random modifications of the variables x_i of the original instance. Once we have our perturbed set X', we train our model g with it, being the target of the prediction given by $f(X')$. As the input and the model are interpretable by a human, we can gather some intuition about which variables affect the output and how for a single instance.

2.4.1.2 Global Shapley

Shapley value is a concept from Game Theory that measures how much an individual player contributes to a game. To calculate it, for each coalition of players it is observed which outcome they achieve, obtaining the weighted average difference of score between the coalitions that include the player and those that do not. In ML, these players are the features of the model, and the score is the accuracy or any metric of how well the model is performing.

$$\xi^{(j)} = \sum_{S \in \mathcal{P}(N \setminus \{j\})} \frac{|S|!(|N| - |S| - 1)!}{|N|!} (v_{S \cup \{j\}}(X_{S \cup \{j\}}) - v_S(X_S))$$

Where $v_{S \cup \{j\}}$ is the value function for dataset X with the feature subset S plus the feature j. The value function, to be a global method has to be the accuracy or the inverse mean absolute error.

The main disadvantage of this method is that a model must be trained for every one of the coalition features and also that the time complexity of this method is exponential. So the best way of implementing this method is to approximate the global importance by sampling some of these feature coalitions.

2.4.2 Rationalization

Rationalization consists in feeding a DL network with examples of human gameplay along with the natural language explanation of the decisions taken. After training the model, we can obtain a translation of the internal state-action of an agent into a natural language text.

The main advantage of this method is that it provides the user a natural language description of the decision taken by the agent, but on the other hand, it needs lots of human players to obtain the gameplays and the explanation feedback to create the dataset, which is beyond the scope of this Thesis.

2.4.3 Traces

One way of analyzing the agent's behavior is seeing executions of its policy in the environment and studying its performance. The main problem with this method is that it is unfeasible to analyze hours of agents' performances.

The optimal way of implementing these methods is by defining metrics or approaches to select the traces that have relevance to defining the agent behavior, reducing the time spent by the analysts.

One of the metrics that can be used is the importance of the state, which is defined as the difference between the discounted reward values for the best and the worst action choice in this state. Then, a certain number of states before and after the important state is displayed.

Another option is to look for critical states, which are defined as states for which choosing a random action gets a significantly worse reward than choosing a specific one.

2.4.4 Explainability from a Markov Decision Process

This is the method implemented in [5], and it consists in creating an MDP graph by running several times the agent interacting with the environment and representing the states using predicates. First of all, we need to discretize our environment to have a finite amount of states in our MDP graph. Then running the agent multiple times we can obtain for each state that the agent visits the probability of taking a certain action and the probability of reaching a state after taking such action. Finally, we will replicate the method in [5] to obtain the predicate from the questions "What will you do when you are in x this state?", "When will you perform x action?" or "Why did not you perform x action on y state?".

Chapter 3

Project Management and Methodology

3.1 Project Planning and Management

The length of this project is about four months, it started at the beginning of February and it is supposed to be completed at the end of June. The amount of ECTS credits assigned to this project is 18, and every credit is supposed to be the equivalent of 30 hours of work, making a total of 540 hours. This corresponds to an average of 4.5 hours a day.

In the next section will be discussed how to manage the time during the elaboration of this thesis. To do better planning and handle efficiently time, the first of all will be to analyze the objectives of the projects to split them into smaller tasks and their sub-tasks. Once this is done, we will be able to plan how much time these tasks will last and when will be expected to be finished.

3.1.1 Task Description

The project has been divided into the following tasks. There is also Table 1 that summarizes these tasks.

3.1.1.1 Project Management

This task consists in organizing the development of this thesis and will be broken into the next sub-tasks.

- **Context and Scope:** In this section of the project, it will be provided the contextualization and the justification of why this topic is relevant to be researched and the objectives that will be covered through the thesis.

- **Planning:** Through this section, it will be explained how we will manage the time to do different tasks and the resources that will be needed in order to accomplish the objectives respecting the time and resource dependencies between tasks.

- **Budget and Sustainability:** The number of resources and the total cost of doing the project will be covered in this section. Also, making a budget and analyzing the sustainability of this project will be done in these sections.

- **Meeting:** Every two weeks, meetings with the advisors will be done to discuss the current state of the project and the following tasks to be done. If it is needed due to difficulties or technical problems, more meetings will be scheduled. Finally we did the meetings every week as stated in the section 4.1.

- **Document Writing:** This task will consist in recording on the documentation every decision, topic researched, or implementation of everything that has been done during the development of the project.

- **Defence of the Project:** Finally, when all the work will be done, this project will be explained and defended in a presentation. This task will consist of preparing the presentation.

3.1.1.2 State of the art Analysis

In this task, we will focus on learning everything about the current techniques related to Reinforcement Learning and ML, and explainability techniques applied to Reinforcement Learning and ML. For that purpose, books, papers, and reports about different related topics will be read. Then, we will analyze the pros and cons of every method or technique.

The following topics will be mainly researched.

- **Agent Learning Algorithms:** Research about the different current algorithms to train an agent.

- **Explainability Methods:** Mainly, we will study and understand the explainability method used in [5]. But also we will learn about the current techniques used in ML and DL.

- **Reinforcement Learning Environments:** Different Reinforcement Learning environments will be analyzed to later select one of them.

- **Metrics to compare Explanations:** We will research the different techniques that could be used to numerically contrast if two explanations are different.

3.1.1.3 Implementing an Agent

Once the explainability method will be implemented and an environment has been selected, we will implement different agents with the agent learning algorithms researched in the previous task.

The task of implementing each agent will be decomposed into the next sub-tasks.

- **Coding the Agent:** Implementing the agent learning algorithm with Python.
- **Training and Tuning the Agent:** Training the agent in the selected environment and tuning the best hyperparameters to get a good performances.
- **Debbugging the Agent:** Detecting and correcting possible bugs and integrate it to the explainability method.
- **Analysing and Testing the Agent:** Analyze with some metrics the performance of the agent in the environment given.

3.1.1.4 Replicate explainability Method

This task will consist on implementing the explainability method used in [5] once we fully understood how it works.

3.1.1.5 Implementing metrics to compare explanations

Once the actual metrics and methods to compare two explanations have been studied, we will have to implement them. This will consists in two sub-tasks.

- **Coding the Metrics:** Implementing the metrics with Python.
- **Debugging the Metrics:** Detecting and correcting possible bugs and verify that it works properly.

3.1.1.6 Compare Results

Finally, after the different agents, the explainability methods, the environment, and the metrics have been selected and implemented, we will compare the different explanations obtained by each agent.

It will be split into the next couple of sub-tasks.

- **Applying metrics to each Agent:** Apply the metrics to each agent and obtain the results.

- **Comparing and Analysing the Results:** Compare and analyze the results between each pair of agents to conclude if the explanations converge or not.

3.1.2 Resources

In this prioject there will be three kinds of resources involved:

- **Human:** The human resources that will be needed to do this project are the two advisors that have the task of supervising and controlling the development of the thesis and myself, who will have the role of developer and tester.

- **Hardware:** The computer used for this project will be a laptop with 8Gb of RAM and a GTX 1060 GPU. It also could be needed an external computer (i.e. provided by Google) if more computational power is necessary.

- **Software:** The software used for this project will be Visual Studio Code as a programming environment in Python, GitLab as a control version operator, Gmail and Meet as a comunication tool, and finally, overleaf to write the documentation of the project.

  Also, as python libraries, we have used: gym, by OpenAI, because it is a great framework to use some environments as CartPole. Tensorflow, that is the one of the main frameworks to work with Neural Networks. Numpy, because is useful to operate with arrays and matplotlib, that can be used to obtain plots of our results.

The analysis of costs related to these resources is provided in Annex A.

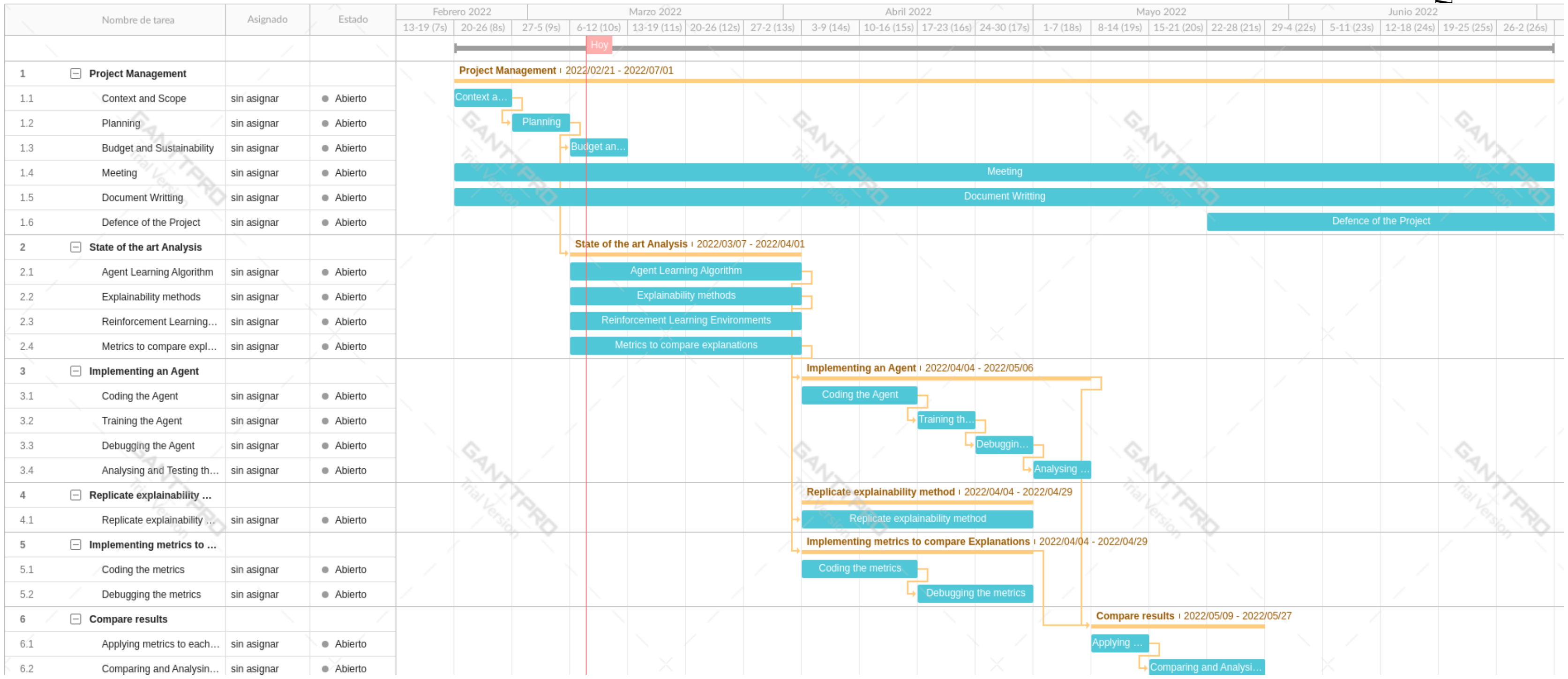
3.1.3 Tasks' Table and Gantt

ID	**Task**	**Time**	**Dependency**	**Resources**
T1	Project Management	**162 h**	-	PC
T1.1	Context and Scope	20 h	-	PC
T1.2	Planning	12 h	-	PC
T1.3	Budget and Sustainability	10 h	-	PC
T1.4	Meeting	10 h		PC, Meet
T1.5	Document Writting	90 h	-	Overleaf
T1.6	Defence of the Project	20 h	-	-
T2	State of the art Analysis	**40 h**	T1.2	PC
T2.1	Agent Learning Algorithm	10 h	T1.2	PC
T2.2	Explainability methods	10 h	T1.2	PC
T2.3	Reinforcement Learning Environments	10 h	T1.2	PC
T2.4	Metrics to compare explanations	10 h	T1.2	PC
T3	Implementing an Agent	**40h**	T2.1	PC, Gitlab, Visual Studio Code
T3.1	Coding the Agent	15h	T2.1	PC, Gitlab, Visual Studio Code
T3.2	Training the Agent	10h	T3.1	PC, Gitlab, Visual Studio Code
T3.3	Debbugging the Agent	15h	T3.2	PC, Gitlab, Visual Studio Code
T3.4	Analysing and Testing the Agent	10h	T3.3	PC, Gitlab, Visual Studio Code
T4	Replicate explainability Method	**40 h**	T2.2	PC, Gitlab, Visual Studio Code
T5	Implementing metrics to compare explanations	**40 h**	T2.4	PC, Gitlab, Visual Studio Code
T5.1	Coding the metrics	20 h	T2.4	PC, Gitlab, Visual Studio Code
T5.2	Debugging the metrics	20 h	T5.1	PC, Gitlab, Visual Studio Code
T6	Compare Results	**30 h**	T5, T3	PC, Gitlab, Visual Studio Code
T6.1	Applying metrics to each Agent	10 h	T5, T3	PC, Gitlab, Visual Studio Code
T6.2	Comparing and Analysing the Results	20 h	T6.1	PC, Gitlab, Visual Studio Code

Table 3.1: Original Table describing the tasks and the resources and time needed. [Self Made]

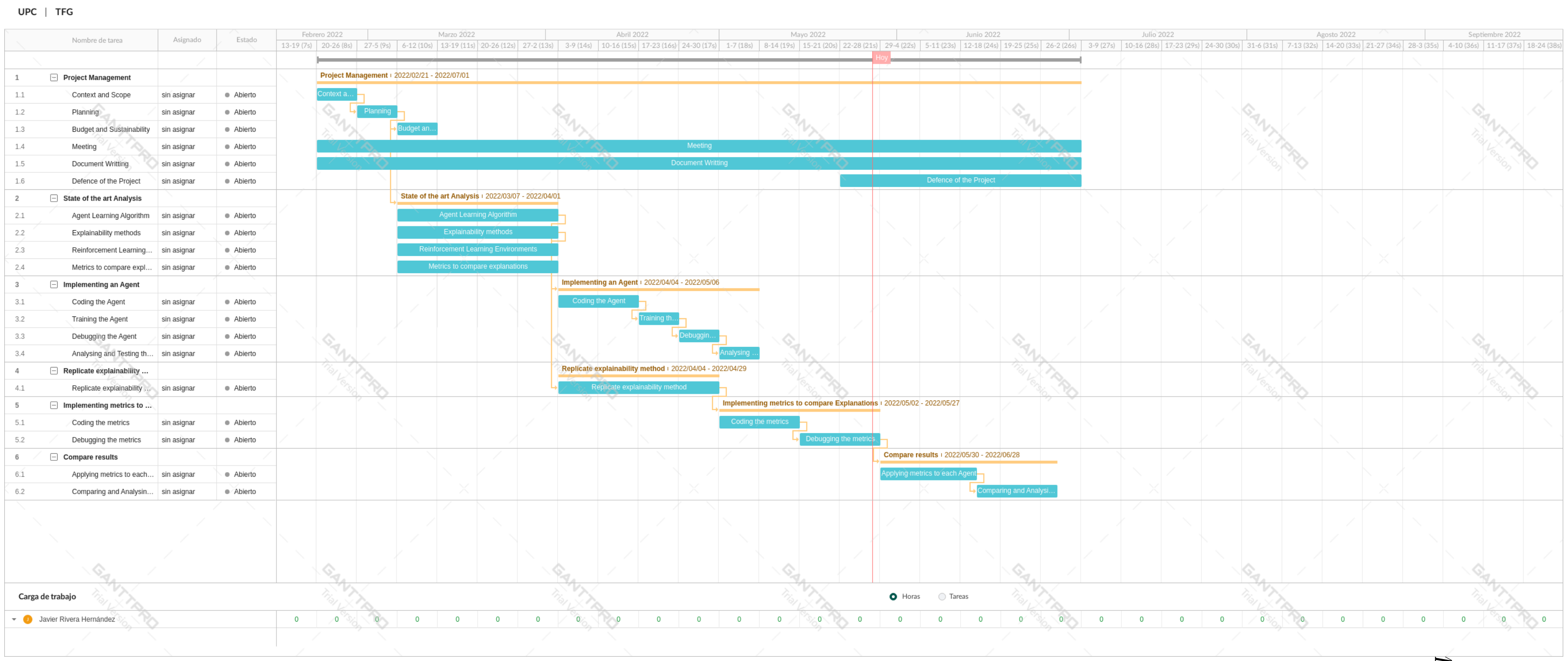
ID	Task	Time	Dependency	Resources
T1	Project Management	**162 h**	-	PC
T1.1	Context and Scope	20 h	-	PC
T1.2	Planning	12 h	-	PC
T1.3	Budget and Sustainability	10 h	-	PC
T1.4	Meeting	10 h		PC, Meet
T1.5	Document Writting	90 h	-	Overleaf
T1.6	Defence of the Project	20 h	-	-
T2	State of the art Analysis	**40 h**	T1.2	PC
T2.1	Agent Learning Algorithm	10 h	T1.2	PC
T2.2	Explainability methods	10 h	T1.2	PC
T2.3	Reinforcement Learning Environments	10 h	T1.2	PC
T2.4	Metrics to compare explanations	10 h	T1.2	PC
T3	Implementing an Agent	**40h**	T2.1	PC, Gitlab, Visual Studio Code
T3.1	Coding the Agent	15h	T2.1	PC, Gitlab, Visual Studio Code
T3.2	Training the Agent	10h	T3.1	PC, Gitlab, Visual Studio Code
T3.3	Debbugging the Agent	15h	T3.2	PC, Gitlab, Visual Studio Code
T3.4	Analysing and Testing the Agent	10h	T3.3	PC, Gitlab, Visual Studio Code
T4	Replicate explainability Method	**40 h**	T2.2	PC, Gitlab, Visual Studio Code
T5	Implementing metrics to compare explanations	**40 h**	T2.4, T3, T4	PC, Gitlab, Visual Studio Code
T5.1	Coding the metrics	20 h	T2.4, T3, T4	PC, Gitlab, Visual Studio Code
T5.2	Debugging the metrics	20 h	T5.1	PC, Gitlab, Visual Studio Code
T6	Compare Results	**30 h**	T5	PC, Gitlab, Visual Studio Code
T6.1	Applying metrics to each Agent	10 h	T5	PC, Gitlab, Visual Studio Code
T6.2	Comparing and Analysing the Results	20 h	T6.1	PC, Gitlab, Visual Studio Code

Table 3.2: New Table describing the tasks and the resources and time needed. [Self Made]



	Nombre de tarea	Asignado	Estado
1	⊟ **Project Management**		
1.1	Context and Scope	sin asignar	● Abierto
1.2	Planning	sin asignar	● Abierto
1.3	Budget and Sustainability	sin asignar	● Abierto
1.4	Meeting	sin asignar	● Abierto
1.5	Document Writting	sin asignar	● Abierto
1.6	Defence of the Project	sin asignar	● Abierto
2	⊟ **State of the art Analysis**		
2.1	Agent Learning Algorithm	sin asignar	● Abierto
2.2	Explainability methods	sin asignar	● Abierto
2.3	Reinforcement Learning...	sin asignar	● Abierto
2.4	Metrics to compare expl...	sin asignar	● Abierto
3	⊟ **Implementing an Agent**		
3.1	Coding the Agent	sin asignar	● Abierto
3.2	Training the Agent	sin asignar	● Abierto
3.3	Debugging the Agent	sin asignar	● Abierto
3.4	Analysing and Testing th...	sin asignar	● Abierto
4	⊟ **Replicate explainability ...**		
4.1	Replicate explainability ...	sin asignar	● Abierto
5	⊟ **Implementing metrics to ...**		
5.1	Coding the metrics	sin asignar	● Abierto
5.2	Debugging the metrics	sin asignar	● Abierto
6	⊟ **Compare results**		
6.1	Applying metrics to each...	sin asignar	● Abierto
6.2	Comparing and Analysin...	sin asignar	● Abierto

Figure 3.1: Original Gantt chart. [Own creation]



UPC | TFG

	Nombre de tarea	Asignado	Estado
1	**Project Management**		
1.1	Context and Scope	sin asignar	● Abierto
1.2	Planning	sin asignar	● Abierto
1.3	Budget and Sustainability	sin asignar	● Abierto
1.4	Meeting	sin asignar	● Abierto
1.5	Document Writting	sin asignar	● Abierto
1.6	Defence of the Project	sin asignar	● Abierto
2	**State of the art Analysis**		
2.1	Agent Learning Algorithm	sin asignar	● Abierto
2.2	Explainability methods	sin asignar	● Abierto
2.3	Reinforcement Learning…	sin asignar	● Abierto
2.4	Metrics to compare expl…	sin asignar	● Abierto
3	**Implementing an Agent**		
3.1	Coding the Agent	sin asignar	● Abierto
3.2	Training the Agent	sin asignar	● Abierto
3.3	Debugging the Agent	sin asignar	● Abierto
3.4	Analysing and Testing th…	sin asignar	● Abierto
4	**Replicate explainability …**		
4.1	Replicate explainability …	sin asignar	● Abierto
5	**Implementing metrics to …**		
5.1	Coding the metrics	sin asignar	● Abierto
5.2	Debugging the metrics	sin asignar	● Abierto
6	**Compare results**		
6.1	Applying metrics to each…	sin asignar	● Abierto
6.2	Comparing and Analysin…	sin asignar	● Abierto

Project Management 2022/02/21 - 2022/07/01
State of the art Analysis 2022/03/07 - 2022/04/01
Implementing an Agent 2022/04/04 - 2022/05/06
Replicate explainability method 2022/04/04 - 2022/04/29
Implementing metrics to compare Explanations 2022/05/02 - 2022/05/27
Compare results 2022/05/30 - 2022/06/28

Carga de trabajo
Javier Rivera Hernández

Figure 3.2: New Gantt chart. [Own creation]

3.1.4 Divergencies from the Original Planification

As it can be seen in the table and the Gantt of the previous section, we only can work in parallel implementing the agents and implementing the explainability method. Whereas the implementation of the metrics, which consists in integrating the agents in the explainability method once both agents exhibit similar behavior, has had to be done after implementing the agent and the explainability because they were previous dependencies that we did not consider originally. Right now we are finishing the integration of the agents in the explainability method, and after that, we will be able to obtain metrics about each agent and compare the results.

This divergence just affect the project planification in the fact that the phase of will be delayed one month from the original planification.

3.1.5 Risk Management

During the course of the project there may appear obstacles that hinders its progress. The potential risks and obstacles have been described in previous sections. In this section we will present how can they be solved by introducing new tasks and adapting the planning to the current situation.

There are two critical tasks, implementing the agents and implementing metrics to compare explanations.

On one hand, implementing an agent needs programming and training. Both could lead to serious delay problems. If there appears bugs while coding, the solution will be to have an extra meeting with the advisors and exploring the possible causes of these bugs.

This will imply more testing and coding hours until the bugs are corrected. We suspect that this can increase the time to implement the agents in 10 hours.

This was stated before starting the implementation of the project, so afterwards, the bugs didn't appear coding the agents itself, they appeared integrating the agents in the explainability method. Although, the solution to the problem was the same as initially stated.

In the case having problems training the agent, we will use more powerful computers to speed up the time needed and also we will train the agents while doing other tasks as Replicate the explainability method. The extra computation power will be obtained demanding to HPAI and BSC some hours of computation in the supercomputer. We also assume that problems training the agents could lead to an increase of 10h of computation.

Finally, the problems training the agents were not related to a lack of computational power, they were related to tuning the hyperparameters and finding the right configuration to obtain a good performance and a behavior similar to the other agent in order to be comparable. The solution to this problem was increasing the time spent to train the agent.

On the other hand, it could happen that some bugs arise while coding the metrics. In this case extra meetings will be taken and hereafter, more hours of coding and testing will be dedicated, approximately 10h.

Still, the estimation of times dedicated in case of setbacks were almost similar for both cases.

3.2 Methodology

In this chapter, it will be discussed the selected methodology and the tools used to organize the development of this project. Finally, it will be discussed the methodology used to validate the results.

3.2.1 Scrum Methodology

A Scrum methodology consists in splitting the project into short periods of time called sprints. Each sprint usually has a 2-week duration, starts with a clear definition of short-term objectives for the period, and ends with a meeting with the stakeholders (in our case the directors of the project) to review and discuss if the objectives proposed at the beginning of this sprint phase are achieved. Moreover, the team will discuss the priorities and possible improvements to be applied in the next phase.

Along the project, this methodology was followed, but the sprints have had finally a week duration, because doing it like this, we can have better monitoring of the project and discuss as soon as possible the contingencies or problems that appear during the implementation.

3.2.2 Tools

The following tools were be used in order to organize the development of this project:

- GitLab platform of the HPAI group at BSC: This is a control version tool used to store the different versions of the code developed.

- Google Meet: This video chat platform will allow communication between the stakeholders of the project. It will be used to do the review of every sprint phase.

- Gmail: This email platform will be used to communicate via text more frequently with the stakeholders.

3.2.3 Validation

As stated above, some metrics and evaluation criteria will be explored and discussed to, first, validate if the explanations obtained are trustful and accurate, and on the other hand, to numerically or symbolically evaluate if the explanations obtained by different agent learning algorithms are consistent or divergent. The numerical or symbolical metrics will be chosen and discussed further. The criteria to validate the explanations will be, for instance, asking people who are experts or frequent users of the environment or use statistical analysis.

We decided to follow two approaches of metrics, qualitative and quantitative metrics. First of all, we need to quantify whether the policies that are going to be compared have similar behavior. In order to do it, we need to define some metrics that will be that both policies have a similar rate of success given a success score lower bound, and a similar mean and standard deviations of the score of both policies.

The quantitative metrics will consist of creating a Markov Decision Process (MDP) of the policy simulating some games and observing the actions selected by the policy given the discretized states of the environment defined in [5]. Once done, we can compare the MDP obtained by each policy and the difference in the actions selected by each policy as well as some environment metrics as the distance covered.

The qualitative metrics will consist in asking questions of the type: “What will you do when you are in x this state?”,“When will you perform x action?” or “Why did not you perform x action on y state?” and compare the answers given by each method.

Chapter 4

Implementation

This chapter provides an explanation of the process to implement the environment, the agents, the explainability method and the Metrics from the explainability module of [5]. This explanation will follow a chronological order.

4.1 Environment

One of the first steps was to decide the environment where to train and discretize the agents, as this is needed to apply the explainability method[1].

OpenAI's Gym library was our choice to implement the environment. Among all the options available in Gym, we do not choose any Atari game because they are more complex and would have been harsher to discretize than CartPole.

CartPole was the best choice since, in [5], it already exists a discretization, and it is more simple than Atari games, allowing us to focus on implementing the agents and comparing its explanations using an environment and an explainability method already made.

By default, "CartPole-v1" has a maximum of 500 steps per episode (game); we have not changed this.

```
import gym

env = gym.make('CartPole-v1')
```

4.2 Agent Training

The initial proposal was to compare two or three learning algorithms, but finally, because of temporal restrictions, we focused on the implementation of two algorithms.

[1]As we explained in §2.4.4, our explainability method is based on creating the MDP to represent its states with natural language predicates. We explain how we implement this explainability method in §4.3.

We considered implementing DQN, which was implemented already in [5], and also, we were interested in Inverse Reinforcement Learning; however, we reject the option of training an Inverse RL agent due to the disability of gathering good gameplays of CartPole in the right time.

With that in mind, we selected an alternative to IRL; as DQN is a Model-Free Q-Learning off-policy agent, the other model selected is PPO, which is a Model-Free Advantage on-policy agent, to have a representative of a more diverse variety of types of agents.

4.2.1 DQN

Deep Q-Network (DQN)[2] is the first agent selected to implement because in [5] it was the one implemented in the thesis. This eases the comparison of the results of our work with the ones from the previous work. But we did not implement DQN with the same architecture as him; we wanted to try to implement it by ourselves and increase the performance of the agent.

Our starting point was a DQN implemented for CartPole from the internet. While the DQN in [5] has three hidden layers with 16 neurons each, our DQN has three hidden layers with 512, 256, and 64 neurons, respectively. In both cases, the input and the output layer have the same width, and the activation functions are also the same, ReLU's[3] in the hidden layers and a linear activation in the output layer.

The memory buffer used to store previous states has a maximum size of 2000 tuples; each tuple consists of the initial state, the action performed, the reward obtained, the next state, and an indicator if the game finished at this step. The memory stores until it gets more tuples than the lower bound established in 1000 tuples. The model trained using the states stored in the memory.

```
def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))
        if len(self.memory) > self.train_start:
            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay
```

The list of hyperparameters of the model without taking into consideration the architecture of the neural network are the following:

- **Episodes:** Number of games we want the agent to play.

- **Discount Rate γ:** Increasing it gives more importance to future rewards, whereas decreasing it reduces it. We decided to leave it at a value of 0.95, which is pretty high, giving much importance to future rewards. We suspect this will work well because it is a simple environment, and the reward function is very straightforward.

[2]Deep Q-Network has been introduced in §2.3.1.

[3]ReLU is an activation non-linear function described as: $f(x) = x\ if\ x > 0, f(x) = ax\ otherwise$

- **Epsilon ϵ:** Probability of exploring a random state instead the selected by the model. ϵ_{decay} modifies the value of epsilon, but initially, it is set at a value o 1 to enforce the agent to explore the space of states.

- **Epsilon min ϵ_{min}:** Minimum value that epsilon can reach.

- **Epsilon decay ϵ_{decay}:** Once more tuples than train start fill the memory, each time we add another tuple to the memory, we multiply epsilon decay per epsilon until epsilon is lower than epsilon min. Initially, it had a value of 0.999, but we found exploring configurations that 0.9999 allows the model to search in the state universe and obtain a more robust policy that gets better performance and does not always move the cart to a side; although, it lasts more time to converge.

$$\epsilon = \epsilon \cdot \epsilon_{decay}$$

- **Batch size:** Determines how many tuples from the memory DQN uses to train.

- **Train Start:** Minimum number of tuples required in the memory to train the model.

- **Max Memory Size:** Maximum number of tuples stored in the memory; once the memory is full of tuples, the new ones replace the latest.

- **Learning Rate:** Determines how much neural net learns in each iteration.

```python
import tensorflow as tf
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam, RMSprop

def OurModel(input_shape, action_space):
    X_input = Input(input_shape)

    # 'Dense' is the basic form of a neural network layer
    # Input Layer of state size(4) and Hidden Layer with 512 nodes
    X = Dense(512, input_shape=input_shape, activation="relu",
    kernel_initializer='he_uniform')(X_input)

    # Hidden layer with 256 nodes
    X = Dense(256, activation="relu", kernel_initializer='he_uniform')(X)

    # Hidden layer with 64 nodes
    X = Dense(64, activation="relu", kernel_initializer='he_uniform')(X)

    # Output Layer with # of actions: 2 nodes (left, right)
    X = Dense(action_space, activation="linear", kernel_initializer='
    he_uniform')(X)

    model = Model(inputs = X_input, outputs = X, name='CartPole_DQN_model')
    model.compile(loss="mse", optimizer=RMSprop(learning_rate=0.00025, rho
    =0.95, epsilon=0.01), metrics=["accuracy"])

    return model


class DQNAgent:
    def __init__(self):
        self.env = gym.make('CartPole-v1')
        # by default, CartPole-v1 has max episode steps = 500
        self.state_size = self.env.observation_space.shape[0]
        self.action_size = self.env.action_space.n
        self.EPISODES = 1000
        self.memory = deque(maxlen=2000)

        self.gamma = 0.95    # discount rate
        self.epsilon = 1.0 # exploration rate
        self.epsilon_min = 0.001
        self.epsilon_decay = 0.9999
        self.batch_size = 64
        self.train_start = 1000

        # create main model
        self.model = OurModel(input_shape=(self.state_size,), action_space =
    self.action_size)
```

One decision, that differs from the typical DQN, is that once the model generates an output of the logits of each possible action, we do not apply a softmax to those values to obtain probabilities (Boltzmann policy); instead, we select the action with the highest value, which is supposedly the optimal action.

Once we have implemented the DQN, we must train it. The optimizer selected is RMSprop, which is used in the original DQN algortihm of DeepMind [15], with a learning rate of $2.5e{-}4$. Our loss function is $L(Q) = (r + \gamma \max_{a'} Q^*(s', a') - Q^*(s, a))^2$, which means that the neural network is trying to predict its output; nevertheless, as this is a simple environment and the Bellman Optimality function is recursive but is being satisfied, it finally converges.

We have explored the possibility of using a Double DQN or Dueling DQN [18] to improve the performance and the convergence, but we did not have enough time, and it was already working well, so it was not required.

The training loop simulates the number of episodes selected or until it reaches a game with 500 steps.

At each episode, we run a maximum of 500 iterations; each iteration, with a probability of $1 - \epsilon$, the agent performs a random action; otherwise, we let the neural network decide based on the Q-Function. Then, the agent performs the selected action, obtaining the new state, the reward, and if the episode finishes; hereafter, we store the tuple $(state, action, reward, new\_state, done)$ in the memory. Finally, if the episode finishes, we start another one; come what may, we retrain the model.

Once we reach 500 steps in an episode or run the selected number of games, we save our model on the disk.

```python
def run(self):
    for e in range(self.EPISODES):
        state = self.env.reset()
        state = np.reshape(state, [1, self.state_size])
        done = False
        i = 0
        while not done:
            self.env.render()
            action = self.act(state)
            next_state, reward, done, _ = self.env.step(action)
            next_state = np.reshape(next_state, [1, self.state_size])
            if not done or i == self.env._max_episode_steps-1:
                reward = 100
            else:
                reward = -100
            self.remember(state, action, reward, next_state, done)
            state = next_state
            i += 1
            if done:
                print("episode: {}/{}, score: {}, e: {:.2}".format(e, self.
    EPISODES, i, self.epsilon))
                print("Saving trained model as cartpole-dqn.h5")
                self.save("cartpole-dqn.h5")
                return
            self.replay()
```

The initial hyperparameters reached an episode with 500 steps, but when testing it, it seems that the agent only learned to go to the right, and in some games, it worked well, but many times it failed. So we needed to tweak the hyperparameters to obtain a more robust agent.

The first approach was to increase the ϵ_{decay} to increase the exploration in the first episodes, allowing the agent to explore new states and new ways of succeeding. After testing a few different values of ϵ_{decay}, we obtained a robust agent that almost always stands 500 steps per episode and moves to both sides.

4.2.2 PPO

The second agent implemented uses Proximal Policy Optimization (PPO)[4]. As we explained in §4.2, we selected PPO to have a representative of the on-policy agents.

Our PPO model is implemented by customizing the Model class of the Keras library. To implement PPO, we used the Actor-Critic approach, which consists of training two networks; one that outputs the action logits (actor) and another that outputs the estimation of the value function (critic).

The actor and the critic share part of the layers, two hidden layers with 512 neurons and relu activations. Then the actor network has an output layer of a dense layer with two neurons with a linear activation (one for each action), whereas the critic network has an output layer of a dense layer with only a neuron with a linear activation.

The memory buffer consists of lists for every element we want to store. Those elements are actions, values, states, an indicator if the episode finishes, and the probabilities of selecting

[4]Proximal Policy Optimization has been introduced in §2.3.2.

each action.

The list of hyperparameters of the model (without taking into consideration the architecture of the neural network) are the following:

- **Critic Loss Weight c_1:** Ponders the contribution of the error made by the critic to the loss function. We tried several values depending on the other hyperparameters.

- **Entropy Loss Weight c_2:** Ponders the contribution of the entropy to the loss function. Increasing it increases the exploration of the agent in front of the exploitation of the previous knowledge.

- **Entropy Discount Rate: $c_{2_{disc}}$:** Multiplies the entropy loss weight each step to reduce the exploration once the model has simulated enough steps.

$$c_2 = c_2 \cdot c_{2_{disc}}$$

- **Discount rate γ:** Increasing it gives more importance to future rewards, whereas decreasing it reduces it. We decided to leave it at a value of 0.99, which is pretty high, giving much importance to future rewards. We suspect this will work well because it is a simple environment, and the reward function is very straightforward.

- **Clip Value ϵ:** Constant that determines the interval at which is clipped the $r_t(\theta)$. It usually is set at 0.1 or 0.2. In our case we set it at 0.2, having that $r_t(\theta)$ is clipped between $(0.8, 1.2)$.

- **Batch Size:** Number of expiriences used to train the model.

- **Learning Rate:** Determines how much neural net learns in each iteration.

- **Number of Epochs:** Number of times the optimizer computes backpropagation to update the parameters of the neural network.

- **Number of Iterations:** Number of times we run the training process.

```
class PPO(keras.Model):
    def __init__(self, num_actions):
        super().__init__()
        self.env = gym.make('CartPole-v1')
        self.num_actions = num_actions
        self.dense1 = keras.layers.Dense(512, activation='relu',
                                         kernel_initializer=keras.
    initializers.he_normal())
        self.dense2 = keras.layers.Dense(512, activation='relu',
                                                  kernel_initializer=keras.initializers.he_normal())
        self.value = keras.layers.Dense(1)
        self.policy_logits = keras.layers.Dense(num_actions)

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return self.value(x), self.policy_logits(x)

    def action_value(self, state):
        value, logits = self.predict_on_batch(state)
        dist = tfp.distributions.Categorical(logits=logits)
        action = dist.sample()
        return action, value

    def act(self, state):
        action, _ = self.action_value(state)
        return action
```

Once we have implemented the PPO, we must train it. The optimizer selected is Adam, with a learning rate of $3e-4$. Our loss function is:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP} - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

```
def critic_loss(discounted_rewards, value_est):
    return tf.cast(tf.reduce_mean(keras.losses.mean_squared_error(discounted_rewards, value_est)) * CRITIC_LOSS_WEIGHT,
                   tf.float32)


def entropy_loss(policy_logits, ent_discount_val):
    probs = tf.nn.softmax(policy_logits)
    entropy_loss = -tf.reduce_mean(keras.losses.categorical_crossentropy(probs, probs))
    return entropy_loss * ent_discount_val


def actor_loss(advantages, old_probs, action_inds, policy_logits):
    probs = tf.nn.softmax(policy_logits)
    new_probs = tf.gather_nd(probs, action_inds)
    ratio = new_probs / old_probs
    policy_loss = -tf.reduce_mean(tf.math.minimum(
        ratio * advantages,
        tf.clip_by_value(ratio, 1.0 - CLIP_VALUE, 1.0 + CLIP_VALUE) * advantages
    ))
    return policy_loss


def train_model(action_inds, old_probs, states, advantages, discounted_rewards, optimizer, ent_discount_val):
    with tf.GradientTape() as tape:
        values, policy_logits = model.call(tf.stack(states))
        act_loss = actor_loss(advantages, old_probs, action_inds, policy_logits)
        ent_loss = entropy_loss(policy_logits, ent_discount_val)
        c_loss = critic_loss(discounted_rewards, values)
        tot_loss = act_loss + ent_loss + c_loss
    grads = tape.gradient(tot_loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return tot_loss, c_loss, act_loss, ent_loss
```

Where the actor network computes the logits of each action (that are converted to probabilities to select one), and the critic network estimates the Value function used to calculate the Advantage.

```
def get_advantages(rewards, dones, values, next_value):
    discounted_rewards = np.array(rewards + [next_value[0]])
    for t in reversed(range(len(rewards))):
        discounted_rewards[t] = rewards[t] + GAMMA * discounted_rewards[t+1]
    * (1-dones[t])
    discounted_rewards = discounted_rewards[:-1]
    # advantages are bootstrapped discounted rewards - values, using Bellman'
    s equation
    advantages = discounted_rewards - np.stack(values)[:, 0]
    # standardise advantages
    advantages -= np.mean(advantages)
    advantages /= (np.std(advantages) + 1e-10)
    # standardise rewards too
    discounted_rewards -= np.mean(discounted_rewards)
    discounted_rewards /= (np.std(discounted_rewards) + 1e-8)
    return discounted_rewards, advantages
```

Through this process, we create the memory buffer and execute steps of the environment until the memory has as many observations as the batch size; once we fill the memory, we calculate the advantages with the value function computed by the network; finally, for the number of epochs selected, we run backpropagation updating the parameters of the networks.

```
model = PPO(num_actions)
optimizer = keras.optimizers.Adam(learning_rate=LR)
train_writer = tf.summary.create_file_writer(f"./PPO-CartPole_{dt.datetime.
    now().strftime('%d%m%Y%H%M')}")
num_iters = 500
episode_reward_sum = 0
state = env.reset()
episode = 1
total_loss = None
for step in range(num_iters):
    rewards = []
    actions = []
    values = []
    states = []
    dones = []
    probs = []
    for _ in range(BATCH_SIZE):
        _, policy_logits = model(state.reshape(1, -1))
        action, value = model.action_value(state.reshape(1, -1))
        new_state, reward, done, _ = env.step(action.numpy()[0])
        actions.append(action)
        values.append(value[0])
        states.append(state)
        dones.append(done)
        probs.append(policy_logits)
        episode_reward_sum += reward
        state = new_state
        if done:
            if episode_reward_sum -1 != env._max_episode_steps-1:
                rewards.append(-100.0)
            else:
                rewards.append(100.0)
            state = env.reset()
            if total_loss is not None:
                print(f"Episode: {episode}, latest episode reward: {episode_reward_sum}, "
                    f"total loss: {np.mean(total_loss)}, critic loss: {np.mean(c_loss)}, "
                    f"actor loss: {np.mean(act_loss)}, entropy loss {np.mean(ent_loss)}")
            with train_writer.as_default():
                tf.summary.scalar('rewards', episode_reward_sum, episode)
            episode_reward_sum = 0
            episode += 1
        else:
            rewards.append(reward)
    _, next_value = model.action_value(state.reshape(1, -1))
    discounted_rewards, advantages = get_advantages(rewards, dones, values, next_value[0])
    actions = tf.squeeze(tf.stack(actions))
    probs = tf.nn.softmax(tf.squeeze(tf.stack(probs)))
    action_inds = tf.stack([tf.range(0, actions.shape[0]), tf.cast(actions, tf.int32)], axis=1)
    total_loss = np.zeros((NUM_TRAIN_EPOCHS))
    act_loss = np.zeros((NUM_TRAIN_EPOCHS))
    c_loss = np.zeros(((NUM_TRAIN_EPOCHS)))
    ent_loss = np.zeros((NUM_TRAIN_EPOCHS))
    for epoch in range(NUM_TRAIN_EPOCHS):
        loss_tuple = train_model(action_inds, tf.gather_nd(probs, action_inds), states, advantages, discounted_rewards, optimizer, ent_discount_val)
        total_loss[epoch] = loss_tuple[0]
        c_loss[epoch] = loss_tuple[1]
        act_loss[epoch] = loss_tuple[2]
        ent_loss[epoch] = loss_tuple[3]
    ent_discount_val *= ENT_DISCOUNT_RATE
    with train_writer.as_default():
        tf.summary.scalar('tot_loss', np.mean(total_loss), step)
        tf.summary.scalar('critic_loss', np.mean(c_loss), step)
        tf.summary.scalar('actor_loss', np.mean(act_loss), step)
        tf.summary.scalar('entropy_loss', np.mean(ent_loss), step)
```

The first one was that we could not save the model as with the DQN because we implemented the PPO as a custom model that inherits from the Keras model class. That kind of custom model can not use the default save function of the model class. Instead, we used the save_format argument 'tf' that saves the model with the SavedModel format.

```
model.save('./PPO_3/', save_format='tf')
```

The other setback was that initially, we did not get good results training the agent. We had to test different hyperparameter configurations multiple times to obtain an agent with acceptable results.

The first approach was to modify the critic and the entropy loss weight but only caused a slight improvement. Then we tried to reduce the learning rate and increase the number of epochs to help the model converge to give good results at every episode because it seemed that after several policy updates, the model started to have good results, but the following updates, it started to get poor results.

Increasing the entropy discount rate also helped improve the results because we initially allowed the agent to explore more states and try new ways of behaving in a state, having more stable policies.

4.3 Explainability algorithm

To implement the Explainability algorithm, we replicated the explainability method used in [5], which consists in creating an MDP to extract natural language predicates from this MDP.

In CartPole-v1, the representation of the state is four numbers. The first one represents the cart position, going from (-2.4, 2.4); the second one contains the cart velocity that goes from $(-\infty, +\infty)$; the third one represents the pole angle, going from $(-24°, +24°)$; and finally, the fourth number describes the angular velocity of the tip of the pole, going from $(-\infty, +\infty)$.

In [5] the following discretization is proposed: "Pole falling left/right", "Pole stabilizing left/right", "Pole standing up", "Cart moving left/right", "Cart positioned far left/right", "Cart near the middle" and "Stuck right/left".

After discretizing the environment, we ran several episodes of the game, gathering which states the agent reached and which actions did perform at each state. Once we have collected this information, we can obtain an MDP representing the agent's behavior because we know the probability of taking some action and how probable it is to reach a state after such action.

Since we got the MDP, we want to obtain an answer to the following three questions: "What will you do when you are in x this state?", "When will you perform x action?" and "Why did not you perform x action on y state?".

- For the first question, we chose the action with the mayor probability in a given state.
- For the second question, we returned the states with the highest probability of performing a given action.
- For the third question, it performs the contrary action, and if it does, it will provide the difference between both states. When this states are no known, it will search for nearby states.

4.4 Metrics

Before comparing the explanations of each agent, we needed both agents to have a similar performance and success rate. With a script, we played several episodes of CartPole to gather the number of times the agent exceeds the success score lower bound and its mean and standard deviation. We did the same for each agent. Then, with the appropriate statistical test, we ensured that it is statistically comparable.

The metrics consisted of the following:

- For the quantitative analysis, apart from analyzing the difference between the MDPs and the performance on some environment metrics as the distance covered, the average velocity or the averge angular velocity, we thought computing the Frobenius Norm $||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$ to quantify the dissimilarity between both MDPs.
- The implementation of the qualitative was done by comparing the answers to the questions implemented in [5].

Chapter 5

Testing and Results

In previous chapter we have described the implementation of the agents, the environment, and the integration of the explainability method to en agent and the environment. We also explained the implementation of some metrics that will be useful to this chapter, where we provide a descriptions of our testing setup and an analysis of the results obtained.

5.1 Agent Performance and Training

Before starting the experimentation, we set a seed for all the experiments because it is the correct way of comparing the results and making them replicable.

As explained in 4.4, in order to compare the explanations of both agents, we needed that they have a similar performance. We set the lower bound score at 450 steps per episode, which corresponds to not losing for 95% of the maximum steps per episode.

The first problem we faced was that the DQN has an outstanding performance (almost always survives $\approx$ 450 steps per episode) while the PPO usually got $\approx$ 300. So we tried to better adjust the hyperparameters to improve the performance of PPO. However, this seemed not to work, so momentarily we considered the alternative of decreasing the performance of the DQN policy. Fortunatelly, this was not necessary in the end, since we obtained the optimal configuration of hyperparameters of PPO to get a performance similar to DQN.

To obtain the optimal configuration, we realized that the initial batch size of 128 experiences was insufficient to get a good performance since we lack the experiences to generalize well. So we increased the batch size until we enhanced the score with a batch size of 2048.

Nevertheless, in some episodes, we obtained good results, but after more policy updates, the results started to get worse, getting a high variability in the score.

To solve this, we had the idea of rewarding the experiences where it gets 500 with +100 and the steps that fall before 500 with -100 to try that the agent training converges faster to not lose before the 500th step. It worked very well, obtaining scores $\approx$ 500, so we applied this idea to DQN, getting a model that always goes to the left and achieves a score of 500. As this is an agent that we can not compare with the PPO because 500 is a limit, and maybe the agent is not perfect, is optimized to get a score of 500, we trained another DQN with the same methodology, obtaining a model that achieves a score of ≈ 500 but behaves more naturally.

Finally, we have three agents to compare: the perfect DQN we used as a control; the DQN that behaves more naturally; and the PPO.

We tried to demonstrate statistically that our agents' scores and performance are equivalent.

So following Figure 5.8, our variable, the score obtained, is quantitative, and we wanted to test if there is a significant difference in the location of the scores obtained by each agent. As we had three agents and do not had a factorial design, we could use One-Way ANOVA, Kruskal-Wallis, or Median test.

We did not use ANOVA because our variables do not follow a normal distribution. To verify it, we used a Shapiro-Wilk normality test. Its null hypothesis is that the distribution does follow a normal distribution. As we can see in Figure 5.1, for PPO and DQN, the p-value is less than 0.05, so we reject the null hypothesis. In the case of the perfect DQN, we could not perform the Shapiro-Wilk test because all its values are the same, so it is not normal.

Also, we can not use Kruskal-Wallis because the scores of each agent do not follow a similar distribution (Figure 5.2) nor have an equivalent spread. To test it, we used the Levene test for each pair of agents. Its null hypothesis is that the agents are homoscedastic. As we can see in Figures 5.3, 5.4, and 5.5, the p-values are less than 0.05, so we reject the null hypothesis; the variances are not homogeneous.

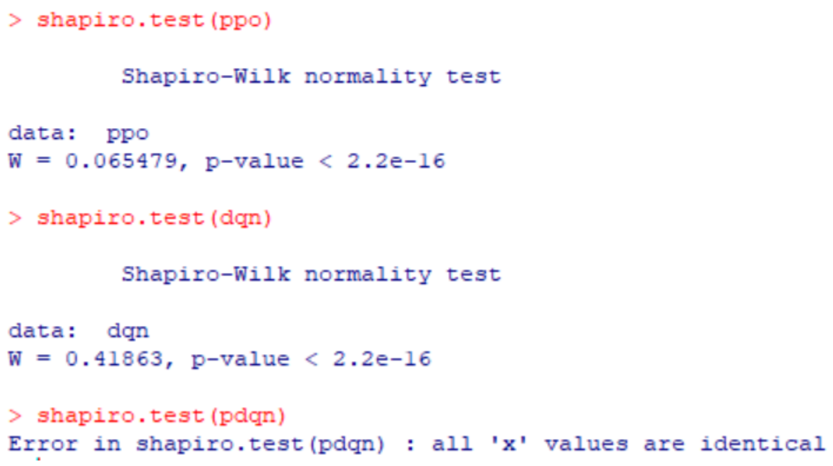
Therefore, we used the Median test because it is the test that we complied the following premises:

We have one-way data with two or more groups; the dependent variable is ordinal; the independent variable is a factor; our observations between groups are not dependent.

Its null hypothesis is that the medians in the scores of the agents are equal.

In the results of the Median test (Figure 5.6), we obtained a higher p-value than 0.05. That means we could accept the null hypothesis; the medians of the scores are equal.

We applied as Post-hoc analysis the pairwise Median test. It confirms that the median is equal between the scores of every agent pairwise (Figure 5.7). The results surprised us because a p-value of 1 is a strong affirmation of the equivalence in the medians between agents.



```
> shapiro.test(ppo)

        Shapiro-Wilk normality test

data:  ppo
W = 0.065479, p-value < 2.2e-16

> shapiro.test(dqn)

        Shapiro-Wilk normality test

data:  dqn
W = 0.41863, p-value < 2.2e-16

> shapiro.test(pdqn)
Error in shapiro.test(pdqn) : all 'x' values are identical
```

Figure 5.1: Shapiro-Wilk normality test of the scores of each agent [Own creation]

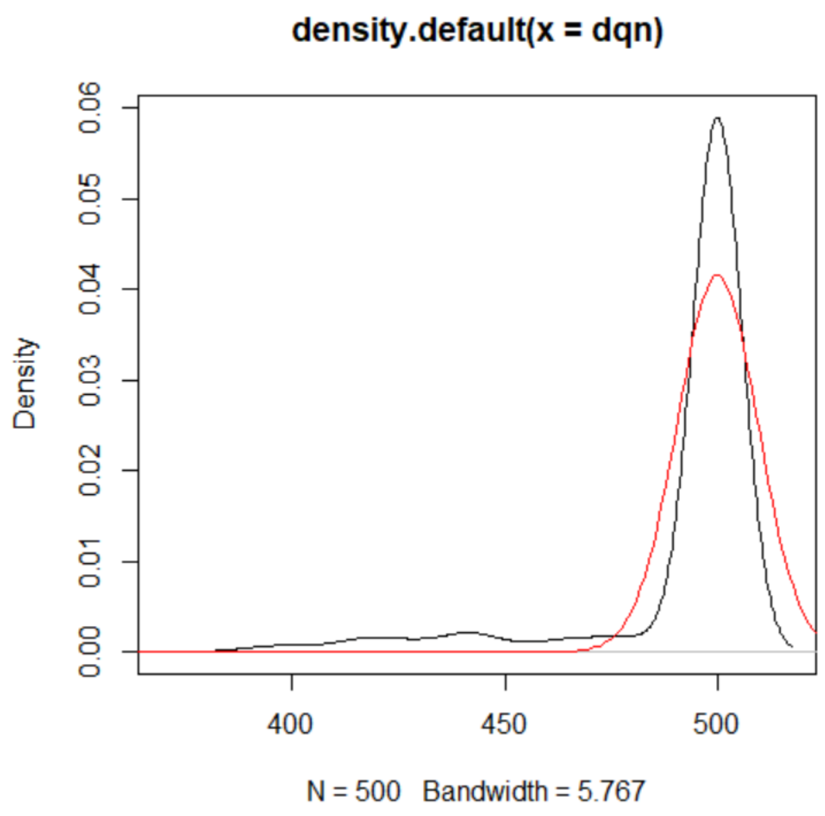
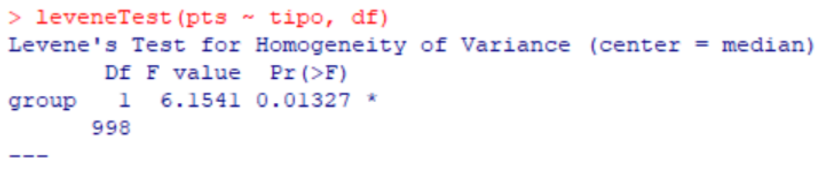
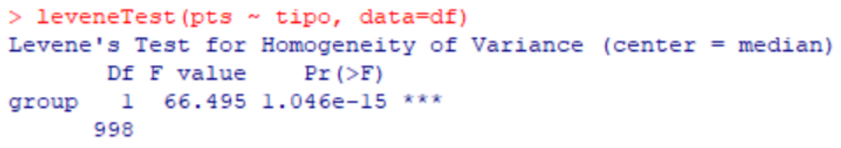


Figure 5.2: In red color: PPO scores, in black color: DQN scores [Own creation]



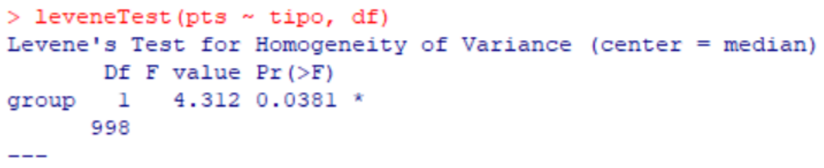
```
> leveneTest(pts ~ tipo, df)
Levene's Test for Homogeneity of Variance (center = median)
       Df F value  Pr(>F)
group   1  6.1541 0.01327 *
      998
---
```

Figure 5.3: Levene test on DQN and PPO [Own creation]



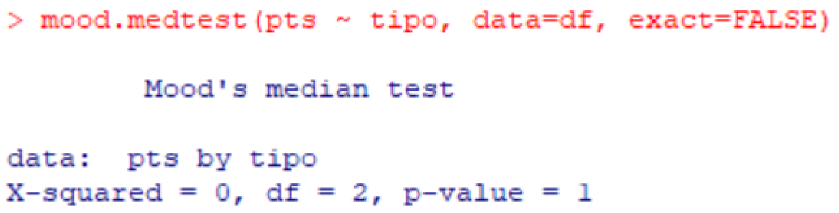
```
> leveneTest(pts ~ tipo, data=df)
Levene's Test for Homogeneity of Variance (center = median)
       Df F value    Pr(>F)
group   1  66.495 1.046e-15 ***
      998
```

Figure 5.4: Levene test on PPO and perfect DQN [Own creation]



```
> leveneTest(pts ~ tipo, df)
Levene's Test for Homogeneity of Variance (center = median)
       Df F value Pr(>F)
group   1   4.312 0.0381 *
      998
---
```

Figure 5.5: Levene test on DQN and perfect DQN [Own creation]

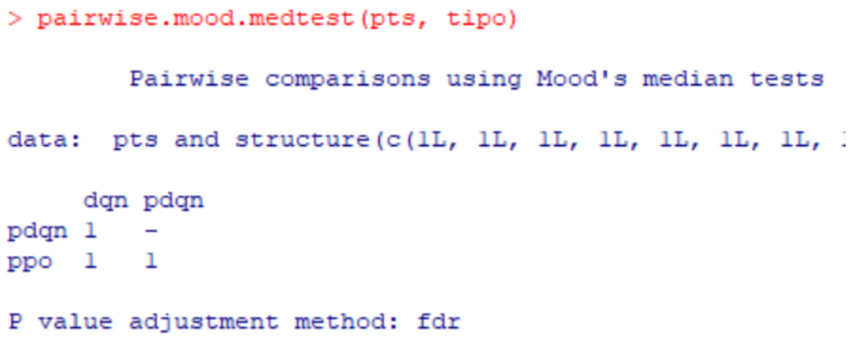


```
> mood.medtest(pts ~ tipo, data=df, exact=FALSE)

         Mood's median test

data:  pts by tipo
X-squared = 0, df = 2, p-value = 1
```

Figure 5.6: Median test applied to the 3 agents [Own creation]



```
> pairwise.mood.medtest(pts, tipo)

        Pairwise comparisons using Mood's median tests

data:  pts and structure(c(1L, 1L, 1L, 1L, 1L, 1L, 1L,

     dqn pdqn
pdqn 1   -
ppo  1   1

P value adjustment method: fdr
```

Figure 5.7: Pairwise Median test applied to the 3 agents [Own creation]

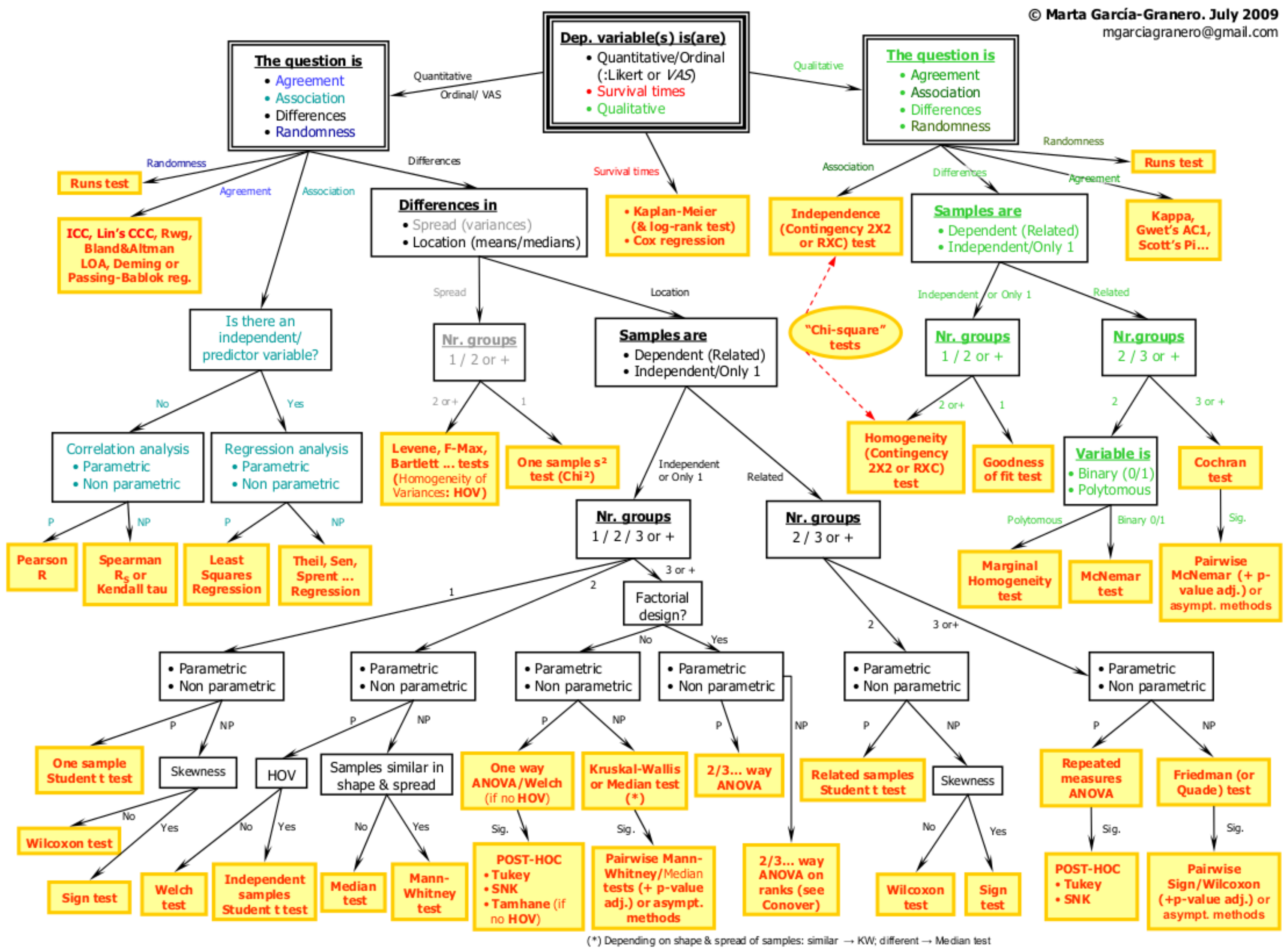


Figure 5.8: A flow chart to select which statistical test to apply [19]

5.2 Comparing Explanations between Agents

In section 5.1, we have seen that the agents have a similar performance in terms of score. That could lead to similar explanations between agents, but it does not necessarily imply it.

The next step is to compare quantitatively how the MDP obtained from each agent, which is the base of the explanations, reacts to randomly generated states. As it was proved in [5], the explanations obtained with this method in this environment are equivalent to the actions chosen by the agents. So if the MDPs of each agent choose the same actions in the same states, we can conclude that the explanations are equivalent.

There are seven different binary predicates to describe the state in the discretization proposed, so there are a total of $2^7 = 128$ states. With this in mind, we generated $128 \cdot 30 = 3840$ random states, to have a better probability of repeating states.

After generating 3840 states, we asked the MDPs of each agent which is the action with more probability of happening. If the state generated does not exist in the MDPs, we select a nearby state that exists and does the same process.

Once we got the actions selected for each state, we applied a χ^2 test to see if the actions of the agents are independent. The null hypothesis of the χ^2 test is that the actions are dependent. In Figure 5.9 we can see that we obtain for each agent a p-value greater than 0.05; therefore, with a significance level of 95%, we can not reject the null hypothesis, and the actions chosen by each agent are dependent. That makes sense because they perform an action for the same state, so its decision is related to the state.

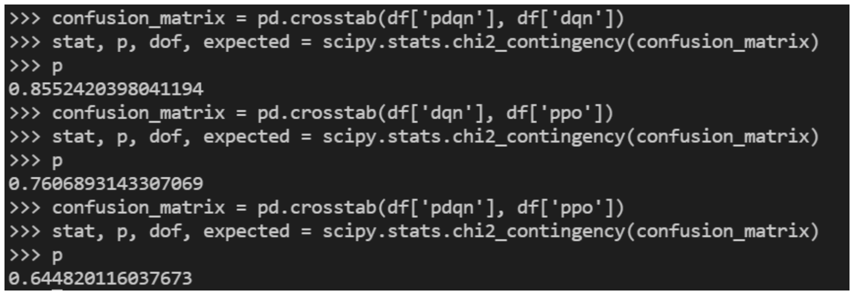
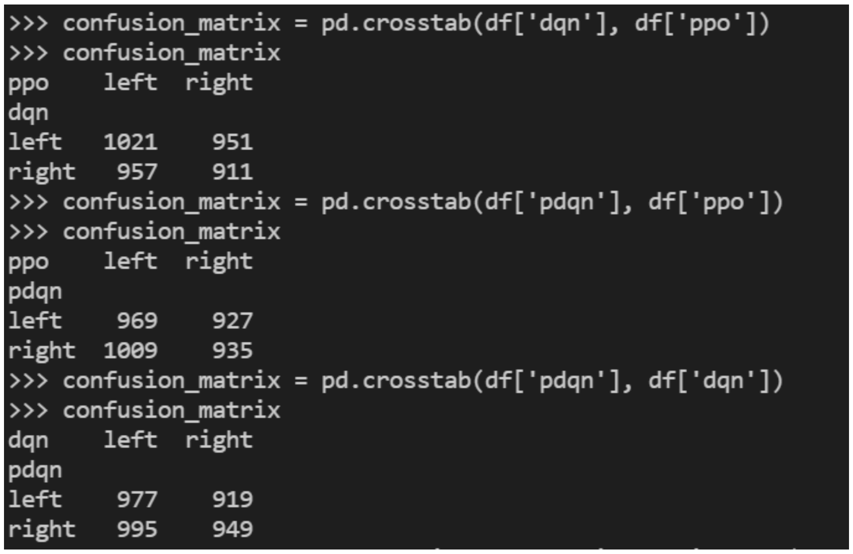

```
>>> confusion_matrix = pd.crosstab(df['pdqn'], df['dqn'])
>>> stat, p, dof, expected = scipy.stats.chi2_contingency(confusion_matrix)
>>> p
0.8552420398041194
>>> confusion_matrix = pd.crosstab(df['dqn'], df['ppo'])
>>> stat, p, dof, expected = scipy.stats.chi2_contingency(confusion_matrix)
>>> p
0.7606893143307069
>>> confusion_matrix = pd.crosstab(df['pdqn'], df['ppo'])
>>> stat, p, dof, expected = scipy.stats.chi2_contingency(confusion_matrix)
>>> p
0.644820116037673
```

Figure 5.9: Pairwise χ^2 test applied to the 3 agents [Own creation]

Finally, knowing that the samples are dependent, we applied a pairwise McNemar test to see if the explanations are somehow correlated. The null hypothesis H_0 is that for each pair of actions taken from two different agents at the same state of the world, $P(action_{agent1}) = P(action_{agent2})$. That means that the null hypothesis is that the actions that the two agents decide after checking their policy graph (their MDP) are equivalent, and therefore the explanations for both agents are comparable. In Figure 5.10 we can see the confusion matrix of each pair of agents, and respectively, in Figure 5.11 we see the McNemar test to each of these pairs. We can see that we have a p-value greater of 0.05 for all the pairs, so we can not reject the null hypothesis with a significance level of 95%. That means that the explanations are statistically comparable.

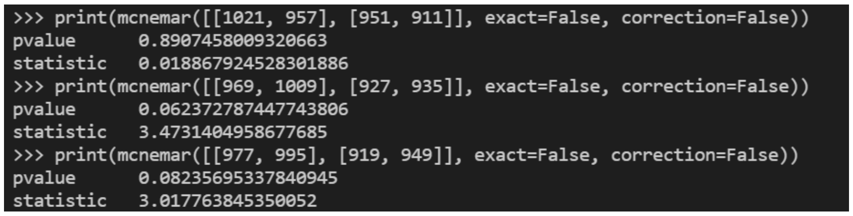
This is an interesting result and it is the most important insight of this project. As we have seen, we have three agents trained with different methods that achieve **comparable performances**, and the MDP graphs we obtain after discretizing the state representation are **not different enough to infer that the policies based on these graphs produce non-comparable explanations**. Further research should be done to compare other metrics rather than success, such as velocity, pole angle, cart speed.

It is also relevant to note that, even though the null hypothesis can not be rejected in any case, if we analyze the statistics and the p-values we can conclude that DQN and PPO have more similar explanations than the other pairs, whereas PPO and perfect DQN are the least similar. This matches our empirical analysis of watching the agents running and analysing the environment traces, where PPO and perfect-DQN very frequently seem to decide opposite actions on similar situations.



```
>>> confusion_matrix = pd.crosstab(df['dqn'], df['ppo'])
>>> confusion_matrix
ppo    left  right
dqn
left   1021    951
right   957    911
>>> confusion_matrix = pd.crosstab(df['pdqn'], df['ppo'])
>>> confusion_matrix
ppo    left  right
pdqn
left    969    927
right  1009    935
>>> confusion_matrix = pd.crosstab(df['pdqn'], df['dqn'])
>>> confusion_matrix
dqn    left  right
pdqn
left    977    919
right   995    949
```

Figure 5.10: Confusion matrix of each pair of agents [Own creation]



```
>>> print(mcnemar([[1021, 957], [951, 911]], exact=False, correction=False))
pvalue      0.8907458009320663
statistic   0.018867924528301886
>>> print(mcnemar([[969, 1009], [927, 935]], exact=False, correction=False))
pvalue      0.062372787447743806
statistic   3.4731404958677685
>>> print(mcnemar([[977, 995], [919, 949]], exact=False, correction=False))
pvalue      0.08235695337840945
statistic   3.017763845350052
```

Figure 5.11: Pairwise McNemar test applied to the 3 agents [Own creation]

For the qualitative analysis, as we implicitly asked the agent what will it do in some state in the previous experiment, we will ask the three agents when will it decide to go to the left or to the right with the major probability. Also, we will ask each agent why it did not perform some actions in a given state.

The main problem that we found designing this experiment is that we did not have an expert on the CartPole game that could validate these results. We focused in the implementation and the quantitative metrics, leaving this as future work.

Chapter 6

Conclusions

AI is a field with a considerable amount of momentum, but humans and complex algorithms do not usually share a common language. In many cases, AI algorithms are black boxes that are hard to understand and interpret. The sub-field of explainability (also called eXplainable AI or XAI) aims at casting some light on these black boxes in order to provide comprehensible feedback related to their behaviour.

In this project, we have followed upon exploration made by previous students on the topic of explainability of reinforcement learning (RL). We have taken as a basis a explainability method based on creating an MDP from agents trained in RL environments with the objective was of testing if, given a fixed explainability method and a fixed environment, we are able to obtain equivalent explanations training agents using different learning algorithms.

Our results showed us that in the CartPole environment using the MDP based explainability method, given different agents trained with different algorithms with a equivalent performance in score, it is possible to obtain similar explanations.

6.1 Main highlights

The novel contribution made in this thesis is a **methodology for comparing the explanations of three different agents and we have applied it in a specific RL scenario**. As a use case, and in order to make a fair comparison, we have trained all the agents in the same environment (CartPole) and we have applied the same explainability method (by building MDPs from the observation of the trained agents). To achieve this contribution, we have fulfilled the objectives presented in §1.3.1.

In the elaboration of this thesis we studied the state-of-the-art and identified the current methods in Reinforcement Learning in §2.3 (objective **O1**). We did not research too deeply in more advanced architectures of Deep Learning as Transformers or Convolutional Neural Networks because it was outside the scope of this project and creating an optimal agent policy was not an objective.

The selection and discretization of the environment was simplified, since we followed upon the previous work from [5] as discussed in §4.3 (objective **O3**). Also, in order to make a direct comparison with this previous work, we chose the same environment (CartPole) introduced in §2.2.1.1 (objective **O2**).

We explored in §2.4 some of the current techniques of explainability methods in general Machine Learning and also some alternatives to Reinforcement Learning (objective **O4**).

Although our initial plan was to implement agents trained with two different learning methods, we ended up training three different agents as described in §4.2. This proved beneficial for the analysis developed in Chapter 5 (objective **O6**).

Some research about the metrics to compare the explanations was made in Sections 3.2.3 and 4.4, especially researching about quantitative metrics where some statistical tests were discovered, and we learned how to interpret them (objective **O5**). We could not finish the comparison in the explanations based on the Frobenius norm between the matrix representation of the MDPs. We also could not do a more exhaustive quantitative analysis based on metrics of the environment such as mean velocity or distance traveled, and we leave this for future work.

The metrics, the explainability method, and the agents were implemented successfully.

6.2 Future work

During the realization of this project, we thought about some ideas that could be interesting to explore but we did not have enough time to develop them. They could be a future expansion of this work.

- Using Frobenius norm between the matrix representation of the MDPs to compare the explanations obtained.

- Try the comparison in other environments or with other explainability methods.

- Use new architectures of Deep Learning such as Transformers, CNNs, or RNNs to implement the Reinforcement Learning agents.

- Doing a more profound quantitative analysis based on the metrics of the environment.

- Contact with experts in the CartPole environment to do a qualitative analysis.

- And finally, do a more thorough hyperparameter exploration, including testing other optimizers.

6.3 Experience used in the project

To do this project we used the experience acquired in some subjects of the degree. We programmed in python for the vast majority of the project, so subjects such as LP, APA, and CAIM helped us to be more familiar with this programming language beforehand. The little experience acquired with the Neural Network framework Keras in one of the practices made in APA, helped us to be more accustomed to using it. Finally, our little knowledge in PE guided us on how to interpret statistic tests and how to argue that the premises are satisfied.

6.4 Experience gained from the project

Personally, I have acquired a lot of understanding making this project about Reinforcement Learning, which was one of the fields of Machine Learning that I knew the less. Also, I learned a lot of statistical tests and how to use them and apply them, increasing the number of statistical tools that I have to analyze and contrast hypotheses. My know-how of the Tensorflow library has increased as well, being more familiarized with it.

6.5 Technical skills

About the technical skills that I had to adquire during the realization of this project, I had four requirements to fulfill. The first one was CCO1.1, which consists in analyzing the computational complexity of a problem and proposing and implementing the best performance. This was accomplished while implementing the experiments, being careful with the efficiency of executing these ones.

The second one was CCO2.1, which consists in demonstrating the fundaments, paradigms, and techniques of intelligent systems. This was accomplished in section §2.

The third one was CCO2.2, which consists of gathering, acquiring, and formalizing the human knowledge in a computer system. This was accomplished changing the rewards of the agents based on the visualization of traces of some episodes and its analysis.

Finally, the fourth one was CCO2.4, which consists of demonstrating knowledge and developing techniques of computational learning. This was highly accomplished when we implemented two Reinforcement Learning agents that learned by themselves a proper way of succeeding in an environment.

Chapter 7

References

[1] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. "Vizdoom competitions: Playing doom from Pixels". In: *arXiv.org* (Sept. 2018). URL: `https://arxiv.org/abs/1809.03470` (cit. on p. 6).

[2] Joseph Suarez et al. "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents". In: *arXiv.org* (Mar. 2019). URL: `https://arxiv.org/abs/1903.00784` (cit. on p. 6).

[3] Cinjon Resnick et al. "Pommerman: A multi-agent playground". In: *arXiv.org* (Apr. 2022). URL: `https://arxiv.org/abs/1809.07124` (cit. on p. 6).

[4] Mikayel Samvelyan et al. "The starcraft multi-agent challenge". In: *arXiv.org* (Dec. 2019). URL: `https://arxiv.org/abs/1902.04043` (cit. on p. 6).

[5] Antoni Climent Muñoz. "An application of explainability methods in reinforcement learning". In: (2020). URL: `https://upcommons.upc.edu/bitstream/handle/2117/335594/154840.pdf` (visited on 03/01/2022) (cit. on pp. 6–9, 21, 24, 25, 33, 35, 36, 44, 45, 52, 56).

[6] Bartomeu Perello Comas. "Creation of an agent in reinforcement learning using explainability methods in a complex environment". In: (2021) (cit. on p. 7).

[7] Thomas M Mitchell. *Machine learning.* Mcgraw-Hill, 1997 (cit. on p. 11).

[8] URL: `https://www.kdnuggets.com/2018/10/mitchell-paradigm-concise-explanation-learning-algorithms.html` (cit. on p. 12).

[9] Andrew G Barto and Richard S Sutton. *Reinforcement Learning: An Introduction (Adaptive computation and machine learning).* Mit Press, May 1992 (cit. on pp. 11, 13).

[10] URL: `https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html` (cit. on p. 13).

[11] URL: `https://ieor8100.github.io/mab/Lecture%20MDP.pdf` (cit. on p. 14).

[12] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: `10.1016/j.neunet.2014.09.003` (cit. on p. 15).

[13] URL: `https://databricks.com/glossary/neural-network` (cit. on p. 16).

[14] URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html (cit. on p. 18).

[15] Volodymyr Mnih et al. “Playing Atari with deep reinforcement learning”. In: *arXiv.org* (Dec. 2013). URL: https://arxiv.org/abs/1312.5602 (cit. on pp. 18, 38).

[16] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv.org* (Aug. 2017). URL: https://arxiv.org/abs/1707.06347 (cit. on p. 19).

[17] Andrew Y. Ng and Stuart Russell. “Algorithms for Inverse Reinforcement Learning”. In: (2000). URL: https://ai.stanford.edu/~ang/papers/icml00-irl.pdf (cit. on p. 19).

[18] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *arXiv.org* (Apr. 2016). URL: https://arxiv.org/abs/1511.06581 (cit. on p. 38).

[19] Marta Garcia-Granero. *Apuntes asignatura Biostatistics del Máster en Investigación Biomédica de la Universidad de Navarra*. 2009 (cit. on p. 51).

[20] URL: https://www.glassdoor.es/Sueldos/barcelona-junior-developer-sueldo-SRCH_IL.0,9_IM1015_KO10,26.htm (cit. on p. 65).

[21] URL: https://www.glassdoor.es/Sueldos/barcelona-project-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,25.htm (cit. on p. 65).

[22] URL: https://www.glassdoor.es/Sueldos/barcelona-project-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,25.htm (cit. on p. 65).

[23] *EUR-lex - 52021PC0206 - en - EUR-lex - europa*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206 (cit. on p. 71).

[24] URL: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf (cit. on p. 71).

List of Figures

List of Tables

Appendix A

Cost Analysis

In this annex we will discuss the cost or budget that would be required for the realization of the project. Costs can be splitted into several sections: staff costs, amortization and contingencies.

A.1 Staff Costs

First of all we must present the different roles that will be necessary for this project. The project managers are responsible for the planning of the project, also a developer will be required because the core of the project is evaluate the effect of using different agent learning algorithms in the explanations obtained, so programming this algorithms and train them will be essential. Finally, the last role would be a tester so it can verify the correctness of the different agents. Due to the pandemic and the adaptation that it implies, the use of a work space won't be necessary. Weekly meetings would be done to control the flow of the project, so all the work can be done from home.

Role	**Salary**	**SS**	**Cost**
Project Manager	20.26	6.08	26.34
Developer	12.2	3.66	15.86
Tester	16.1	4.84	20.94

Table A.1: Table that contains a summary of the staff salaries. salary from [20][21][22]

ID	**Task**	**Time**	**Proj. Manager**	**Developer**	**Tester**	**Cost**
T1	Project Management	162 h	162 h	-	-	**4267.08**
T1.1	Context and Scope	20 h	20 h	-	-	526.80
T1.2	Planning	12 h	12 h	-	-	316.00
T1.3	Budget and Sustainability	10 h	10 h	-	-	263.40
T1.4	Meeting	10 h	6 h	-	-	263.40
T1.5	Document Writing	90 h	90 h	-	-	2370.60
T1.6	Defence of the Project	20 h	20 h	-	-	526.8
T2	State of the art Analysis	40 h	5 h	35 h	-	**686.80**
T2.1	Agent Learning Algorithm	10 h	1.25 h	8.75 h	-	171.70
T2.2	Explainability methods	10 h	1.25 h	8.75 h	-	171.70
T2.3	Reinforcement Learning Environments	10 h	1.25 h	8.75 h	-	171.70
T2.4	Metrics to compare explanations	10 h	1.25 h	8.75 h	-	171.70
T3	Implementing an Agent	40 h	4 h	30 h	16 h	**916.20**
T3.1	Coding the Agent	15 h	1 h	14 h	-	248.38
T3.2	Training the Agent	10 h	1 h	9 h	-	169.08
T3.3	Debbugging the Agent	15 h	1 h	7 h	7 h	283.94
T3.4	Analysing and Testing the Agent	10 h	1 h	-	9 h	214.8
T4	Replicate explainability Method	40 h	2 h	28 h	10 h	**706.16**
T5	Implementing metrics to compare explanations	40 h	2 h	28.5 h	9.5 h	**703.62**
T5.1	Coding the metrics	20 h	1 h	19 h	-	327.68
T5.2	Debugging the metrics	20 h	1 h	9.5 h	9.5 h	375.94
T6	Compare Results	30 h	2 h	28 h	-	**496.76**
T6.1	Applying metrics to each Agent	10 h	-	10 h	-	158.60
T6.2	Comparing and Analysing the Results	20 h	2 h	18 h	-	338.16
	Total Cost		177 h	149 h	35.5 h	**7776.62**

Table A.2: Table that contains a summary of the staff costs [Self Made]

A.2 Hardware Costs

As we specified in the previous section we have been using a decent laptop in order to code, train and write the report of the project, the cost of this laptop was 1200€. As hardware cost we will have to calculate the amortization of the laptop.

A.3 Software Costs

All the software used for this project such as Visual Studio, Python and its libraries used, GitLab, Gmail, Meet and Overleaf are free. So it will not be considered calculating the amortization.

A.4 Amortization

In our case the amortization comes from the hardware used, as we are renting a service in order to train the only amortization is the laptop used for programming and writing, also as the software we are using is open source there is no amortization. So the formula for calculating the amortization is the following.

$$Amortization = productPrice * \frac{monthsUsed}{expectedLifeTime}$$

So amortization will be $1200 * \frac{5}{40} = 150$ €

A.5 Contingencies

Throughout a project unexpected events may appear and those events can have an impact in the cost, having a contingency budget is necessary to cover them in case of need. A common value used in software projects is a 15% of the total cost. This is the value we calculated at the start of the project with an estimated cost of 10803.51€.

Contingency cost = 7926.62 * 0.15 = 1188.99€

A.6 Total Cost

The next Table sums up all costs calculated previously. We will also have to add the price of the electricity, internet and heating that will be consumed during the elaboration of this project. This will suppose about a 6% of the total cost

A.7 Management control

Once the budget has been calculated we must have a form of controlling the budget and amortization deviations, in order to do so after a project stage has been completed we will calculate both deviations.

Source	**Cost**
Staff Cost	7776.62
Hardware Cost	150.00
Contingencies	1188.99
Electricity, Internet and Heating	546,94
Total	**9662.55**

Table A.3: Table that contains a summary of the costs. [Self Made]

In order to calculate the deviations we just need to subtract the real cost of a task from the estimated, if we get a positive result we are below the estimated meaning we have spare budget that can be assigned to troubleshooting but in case the result is negative we will have to take funds from the contingency budget.

Appendix B

Sustainability Analysis

Before doing the survey I already knew the existence of the three different dimensions that make up sustainability, environment, social and economic, but my knowledge was not as high as expected regarding the social dimension.

While the project was being organized one of the goals was to optimize the resources that could be used which has relation with the environment and the economic dimensions.

B.1 Environmental Dimension

In regards to the environmental dimension, the computer we are using for the writing and the coding is one we already had, so there is no more potential electronic waste.

The main potential source of environmental damage that this project could produce is the waste of energy related to the computational power needed to train the agents, because it requires huge amounts of energy. For that reason, our concern is that we invest more time coding to avoid spending extra time training the agents.

B.2 Social Dimension

Since the beginning of the degree I knew the existence of artificial intelligence which was a topic that had my attention.

With every passing day, AI is getting even more relevance and doing more crucial tasks as those related with medicine. That is the reason why European Union and other political entities have hardened their stand in what tasks or decisions could be automated by a AI algorithm due to his ethical and social consequences, demanding that those tasks completely automated by a AI algorithm have a explanation of their decisions that can be understood by humans.

So this project will explore the field of the explanations of the AI algorithms and therefore, contributing to the social side of the AI.

B.3 Economic Dimension

In my opinion the calculation of the budget has been as realistic as possible, the only problem we may face are the training hours. I think that I underestimated them a bit, but nothing else from there.

As this is a research project and an extension of a PhD project it is quite hard to estimate the economic impact it may have. One benefit of knowing the behaviour of an agent or network could be that once a similar task has to be faced there is no need to invest resources into solving the problem from scratch.

The major problem regarding this type of project is that is very hard to improve the economic issues because improving the training speed which would involve less economic resources implies using better hardware which is more expensive and less energy efficient

Appendix C

Regulatory Analysis

Explainability is a hot topic right now because the laws and regulations are tightening up to algorithms that affect directly the life or rights of humans or animals. These laws state that algorithms that act as a black box have to explain their decisions and thus it applies to all the algorithms of the Machine Learning family.

According to the Regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (artificial intelligence act) [23], in 5.2.2, the agents implmented in this project are not considered a prohibited AI practice, because they not violate any fundamental rights, do not have any potential to manipulate people nor cause psychological or physical harm. Also, the section 5.2.3 does not apply to this AI system, because is not intended to be used as safety component of products that are subject to third party ex-ante conformity assessment nor other stand-alone AI systems with mainly fundamental rights implications that are explicitly listed in Annex III. In seciton 5.2.4, says that every AI system that interact with humans this must be notified to the users, and that applies to this project.

During the realization of this project, no personal data is used and any person is being profiled, so the "Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales" [24] does not apply to this project.