Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

FINAL YEAR THESIS

An Artificial Intelligence framework for Air Navigation based on Deep Reinforcement Learning

Author: **Luis Navarro García**

Tutor: **Juan Antonio Vila Carbó**

Bachelor's Degree in Aerospace Engineering

Higher Technical School of Design Engineering

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

July 2021

A mis padres, por el continuo apoyo en las decisiones que he tomado en mi vida; al resto de mi familia, por ser los responsables de hacerme crecer feliz y ser los cimientos de lo que soy hoy; y a mis amigos, por ayudarme a salir de malos momentos con una sonrisa y sacarme de mi zona de confort siempre haciendome una persona un poco mejor.

Agradecimientos

Me gustaría agradecer a mi tutor, Juan Antonio Vila Carbó, la confianza depositada desde el comienzo en mi capacidad de poder llevar a cabo un trabajo sobre temas que no necesariamente sabía que tenía el conocimiento necesario para hacer. Además, su función de tutor ha sido vital en darle un sentido y rumbo adecuado al proyecto desde el primer momento. A su vez, he de agradecer al grupo de Sistemas de Navegación Aérea la acogida y su contínuo apoyo, sobre todo en manos de Pablo Morcillo y Joaquín Vico, que han aportado su granito de arena de forma contínua y una opinión más que necesaria para encaminar el trabajo hacia buen puerto. Agradecer también a Juan Vicente Balbestre Tejedor la confianza depositada junto a Joan a la hora de comenzar el trabajo.

La Universitat Politènica de València ha ayudado a potenciar mi espíritu emprendedor, no solo a nivel empresarial, sino como actitud diaria, apoyando las ideas fuera de lo habitual y las aspiraciones de todo aquel que tenga un sueño y ganas de perseguirlo. Por lo tanto, agradezco a la universidad estos cuatro años juntos y empiezo una segunda etapa aquí con la confianza de estar muy bien rodeado.

También me gustaría agradecer, aunque quizá no sea lo habitual, a la ciudad de Valencia. Porque me siento agradecido de tener este escenario para desarrollarme como persona. Sus lugares, gente de todos sitios, cultura, y aspiraciones sirven de guía para crecer tanto en lo personal como en lo profesional.

Abstract

Air traffic has been continuously rising. The expected exponential growth of unmanned traffic, like for delivery services, can result in an uncontrollable airspace that might rely on finding solutions that depend on more automation. Adding Artificial Intelligence to the question might result in a good step into the future of Air Navigation given its recent advancements. This project covers some theory of Artificial Intelligence in order to understand a basic series of concepts prior to the explanation of Deep Reinforcement Learning as the choice of Artificial Intelligence to be used in the project. This type of Artificial Intelligence is then implemented into a workflow that makes it possible to test the application of Deep reinforcement Learning at the end before discussing the results.

Keywords: **Air Navigation, Artificial Intelligence, Neural Networks, Deep Reinforcement Learning, Proximal Policy Optimization, RLlib, BlueSky**

Resumen

El trafico aéreo no ha parado de aumentar. Se espera, además, un aumento exponencial del número de aeronaves no tripuladas, para servicios como el *delivery*, que pueden acarrear que el espacio aéreo se convierta en incontrolable y donde soluciones como aumentar el nivel de automatización podrían ser clave. Añadir Inteligencia Artificial a la cuestión podría resultar en un buen paso hacia delante en el futuro de la Navegación Aérea dado sus avances en los últimos años. El trabajo cubre algo de teoría sobre Inteligencia Artificial para poder comprender una serie de conceptos previos a la explicación del Aprendizaje Reforzado Profundo como elección de tipo de Inteligencia Artificial a desarrollar en el proyecto. Este tipo de Inteligencia Artificial es luego implementado en un entorno que hace posible la aplicación de Aprendizaje Reforzado Profundo a la Navegación Aérea para posteriormente realizar una serie de experimentos y discutir sus resultados.

Palabras Clave: **Navegación Aérea, Inteligencia Artificial, Redes Neuronales, Aprendizaje Reforzado Profundo, RLlib, BlueSky**

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

AI Artificial Intelligence

ANN Artificial Neural Networks

DL Deep Learning

DNN Deep Neural Networks

DP Dynamic Programming

GPI Generalized Policy Iteration

IDE Integrated Development Environment

LSTM Long Short-term Memory

MC Monte Carlo Methods

MDP Markov Decision Process

ML Machine Learning

PPO Proximal Policy Optimization

RLlib Reinforcement Learning Library

RL Reinforcement Learning

TD Temporal Difference

TRPO Trust Region Policy Optimization

UAV Unmanned Aerial Vehicle

UI User Interface

UPV Universitat Politècnica de València

Chapter 1

Project definition

Thoughts and motivation behind the creation of the project and a description of it.

1.1 Introduction

Drones have been flying for several years now, but it is in the recent past that they have become a much more affordable and, thus, used product, to the point that it is difficult not to see any fly on a regular basis. Due to this, we are already used to hearing news of drones putting people in danger, whether it is by getting too close to manned aircraft, or just by getting too close to people. On top of that, it is difficult to have regulations followed by individuals, often due to the vast amount of users that there already is and how hard it makes it to know what each one of them is doing. The great variety of drones, many even being homemade ones, explains the lack of standards and safety measures on these kind of aircraft, specially smaller ones used by hobbyists, small film makers, or the ones treated as toys.

Drones are starting to be used for applications such as the delivery of packages in urban settings and could soon result in traffic densities way higher than for manned civil aviation, that as well is expected to grow. From this, and what was mentioned before, we can extract that the situation could easily turn into a much harder problem that might actually be impossible to control if we just take the measures as we know them today.

In an scenario where many unmanned aircraft are airborne the necessity for a service that organizes and helps avoid conflicts is a key problem that has to be dealt with in a near future and, thus, the addition of AI to the equation could lead to a solution that is both affordable and effective.

AI has not stopped gaining maturity and has recently meant a radical change in how many problems are approached. This subject is shown all the time around the news as well, and it seems like most people still find it difficult and confusing to know what is behind this kind of technology and how it can be approached in order to solve modern day problems and proposing great new never-before-seen ideas that could revolutionize certain aspects of life and industry.

1.2 Past work

The topic has been researched in several occasions. Most of the work that seems to exist, or at least the work that is publicly available to read, is done using a simulator that will later be introduced in the document and is called BlueSky [12].

The efforts have been mainly in the field of deep multi-agent reinforcement learning and most authors state the difficulty of finding an optimal solution to most of the problems due to the instability that this kind of algorithms can face due to several reasons that will be discussed in future sections. Another recurrent topic found in previous work is the high amount of computing power and time needed to solve the complex problems proposed in the papers or thesis.

The paper by the authors Marc Brittain and Peng Wei seems to find optimal solutions to the case studies that are proposed, mainly related to conflict resolution in air routes, by using different kinds of algorithms and a framework created by themselves. This is at the expense of a really good computing platform and around 5 days each time training has to be done [13].

The job done by Marta Ribeiro is focused more around drones in urban areas, the problem she also finds is concerning the convergence of the model, making it hard to reach an optimal solution [14].

Bart Vonk's thesis tries to use reinforcement learning for autonomous sequencing and spacing of aircraft. His results also show the concern of learning stability and an optimal solution is only found for a simple scenario involving one aircraft [15].

Dennis van der Hoff found in his thesis that, for lower density traffic, his method could provide collision avoidance and approach to the correct runway. However, he also found that more complex scenarios were difficult to get solved [16].

The fact that some authors did not find the results they wanted shows the high level of experimentation in which this field is at, the problems that were mentioned as not solved were mainly to the high degree of freedom and the great amount of different states and possible actions that were involved in those problems, making the agent harder to train.

1.3 Motivation

The interest in the subject of AI came around the second year of the Bachelor's Degree in Aerospace Engineering. The way AI could solve such complex problems seemed somewhat magical and the curiosity of knowing what was behind those algorithms lead to a lot of research and learning of the basics of AI.

Being part of Horus UPV team at university, dedicated to the design and construction of an autonomous UAV, I discovered that the knowledge acquired during the previous summer was actually enough to understand how to solve a computer vision task related to the project and got the responsibility of leading the computer vision team inside Horus UPV for the next year.

It was during quarantine and the following summer that a lot of time was spent obtaining several certificates, specially in the field of Deep Learning, and got a better quality education related to the field. This led to a better global understanding of the subject and the potential of it as a solution to many problems. One of those problems was Air Navigation, which lead to a marriage of the self-taught skills related to AI with what was being taught at university.

The last year of the degree was the moment the project was born. Talking with who would later become the tutor of the project about the field of AI and its potential uses for Air Navigation, he thought AI could be useful in a near future to some of the research projects carried out at the university and helped establish boundaries to what could be done, narrowing the task to a more specific subject and application. This is the moment I was included in the *Bubbles* project by *SESAR Joint Undertaking* as an apprentice. One of the partners, as well as the coordinators, is UPV and the group of *Sistemas de Navegación Aérea* at the university is, thus, in charge of a good portion of the research. In Bubbles, I was asked to start the research in the field in order to solve the problem of tactical separation of drones for the work package 4. My participation was conceived as an apprenticeship and as a way to develop my final year thesis with the work done with them.

This project contains a lot of personal ambition and started as a challenge, something different that could, maybe, in a near future have served as the seed of something bigger and hopefully useful. As well, it serves as a way of achieving a lot of personal growth, having to work on something that is probably out of the comfort which means a lot of new knowledge and development of problem solving skills, specially having the change to participate in the Bubbles project with a professional environment around.

1.4 Description and objective of the project

The project consists in the research of a potential application of AI for air navigation purposes and, for this, the basics of AI and the different types of it have to be understood. Later a workflow is created using several tools, that will be introduced, and ideas in order to be able to solve problems regarding the use of AI in the field. Finally, there is experimentation in order to test if the different software components can work well together and the potential application of this in the field of air navigation. It has to be highlighted that this project is conceived as a research oriented project.

The main objective of the project, thus, will be understanding AI and the different kinds of it, its potential for solving air navigation problems and the explanation, set-up, and test of the different software in order to create a workflow that helps solving these problems with AI.

1.5 Justification

As mentioned previously, there is a need to increase the level of automation in air navigation due to the increase in traffic density, and specially when talking about drones, the creation of a centralized management and control service how we know now seems unrealistic.

Increasing the level of automation is a task that has been worked on and is still being developed today but usually means having to deal with unknown levels of safety that, specially in manned aviation, mean that its implementation into production is usually difficult. However, drones are not like manned aviation in terms of safety as, in most cases, damage to a drone usually does not mean endangering anyone. This

is an opportunity to use this kind of aviation as a test platform for new techniques that could eventually even make their way to manned navigation, but even if that is not the case, it would still hugely benefit the way we deal will the increase in traffic that supposes the addition of all unmanned traffic to come in a near future.

The paragraph above serves as the reason why studying how to apply, and the potential use cases of AI in the field, seems to be an attractive idea to take into account and explore.

1.6 Structure

The project consists of several sections in which the aim is to introduce the field of AI in a general way, later go into more specific details, and afterwards propose a workflow with several software that will be introduced and tested before closing up by reflecting on what has been done and achieved.

First it is necessary to have a quick look into what AI is, the different kinds of AI, and understand what is the most attractive kind in order to solve the proposed problem. It will be then key understanding how that exact type of AI works as it will be the tool that will help solve the problem, and thus, knowing the potential of your tools is vital for extracting the most out of them.

After the more theoretical explanation, it is necessary to take a look into how we can make use of all that by using different software and, for that, the library that will help with the AI part, *RLLib*, will be explained. As an experiment with real aircraft cannot be performed due to obvious reasons, a simulator will be key in order to approach this problem and, thus, a brief explanation of the *BlueSky* simulator will be carried out in order to understand how it can help as a platform to quickly and easily obtain a simulation of different aircraft in premeditated scenarios taking into account many complex factors that realistically replicate a real world behavior. A quick look into what virtual environments are, the Integrated Development Environment used and another coding platform will close up the talk about software.

It is essential to take a look into how the different software integrates and works together, also to test how the proposed techniques behave when dealing with air navigation problems. Having said that, a little experimentation will be carried out to test all of this.

The results obtained have to be analysed to conclude whether we can take this as a valid solution to the problem. In this section the capabilities of AI will be assessed for this kind of problems and it will be possible to extract a more precise conclusion regarding its usage as a way to solve different air navigation problems proposed.

Finally, its is key to reflect on what things regarding the project could be changed in the future and what ideas could be implemented in order to help give it a better shape. Also, it is good to mention the potential applications of the solution presented in the document or, at least, when the technology is better developed and matured.

Chapter 2

Theoretical Study

In this chapter, the necessary theory to understand how to apply AI to Air Navigation is discussed..

2.1 What is AI?

Less than ten years after cracking Enigma, the Nazi encryption machine, Alan Turing asked a simple question. "Can machines think?". In his paper "Computing Machinery and Intelligence" [17], published in 1950 and the Turing Test, which is an exam that evaluates the capability of a machine to act in a humanly intelligent way, Alan Turing established the basis and goals of AI.

AI can be thought of as the branch of computer science that tries to replicate or simulate human intelligence and behavior in machines, however the exact definition of AI that is universally accepted is yet to come. A major question that can come to mind after the definition of AI above is that it does not really explain what AI is, as no answer to what makes a machine intelligent is offered.

In the book *Artificial Intelligence: A Modern Approach* [18], Stuart Russel and Peter Norvig say that AI is "the study of agents that receive percepts from the environment and perform actions" and they explore four traditional approaches to the definition of AI: thinking humanly and rationally, and acting humanly and rationally. The first two cover the thought process and reasoning and the next two deal with the behavior.

Another definition comes from Patrick Winston, the former Ford professor of

artificial intelligence and computer science at MIT, he defines AI as "algorithms enabled by constraints, exposed by representations that support models targeted at loops that tie thinking, perception and action together"

Both definitions mentioned above may seem pretty abstract to the average person, but they help establish the field as an area of computer science.

2.1.1 How AI is used

AI can be divided into two categories that cover a huge amount of topics: Artificial General Intelligence and Narrow AI.

Artificial General Intelligence can be said to be focused on the AI that can be seen in science fiction movies. This kind of AI tries to achieve, as its name says, a general intelligence that, just like a human, can be employed by a machine to solve any kind of problem. This topic seems to be the ultimate goal for many researchers but it has proven to be a really difficult task and it seems like it will not be a reality anytime soon.

Most AI based applications that can be seen nowadays are in the field of Narrow AI and many examples can be found, like *Siri*, or the recommendation algorithms from *Youtube* or *Netflix*. Narrow AI usually operates within a limited context and tries to simulate human intelligence. It is usually created to focus on a single task extremely well and although they might seem intelligent, they usually operate under a lot of constraints and limitations that not even the most basic human intelligence compares.

As it can be deduced from the examples in the previous paragraph, Narrow AI is clearly the most successful implementation of AI up until now, and it has experienced a massive growth in the last decade.

Machine Learning breakthroughs and specially one of its types, Deep Learning, power much of the the Narrow AI that is used today. Those usually are the names that first come to mind when thinking about AI and, sometimes, it can be difficult to differentiate the concepts of AI, Machine Learning and Deep Learning. Essentially, Deep Learning is a kind of Machine Learning as expressed in the beginning of the paragraph, and Machine Learning is one of the techniques used in AI. [19]

The project is built using techniques of Machine Learning and for that it is key to understand what Machine Learning is and what are the different types of it.

2.2 Machine Learning (ML)

ML is focused on the creation of algorithms. Algorithms can be easily and briefly defined as sequences of instructions that generate a solution to a problem.

An algorithm can be created by a programmer who establishes the rules that describe the behavior of an algorithm. For example, if a program is created that prints to the console the values of a linear function for the first one hundred positive integers given a certain slope and intercept introduced previously by the user in the terminal, the program will consist of a section of code that handles the acquisition of the two constants and its storage in a variable, a section that calculates the one hundred values and another section that prints all those values in the terminal.

In ML, however, the rules are established by the computer and the computer is provided with several tools that allow it to learn from data without the necessity of programming each step of the process, and thus, the program defined in the last paragraph can be solved from the point of view of ML as an example. Let's suppose that the first one hundred integers were saved and tagged under the name "inputs" and the output of the terminal was saved under the name "output" after running the previously explained algorithm. In this case we know what we want to achieve but we don't know how was that output created. A solution using ML would be the creation of a model that gets fed the input and output data and learns the way to get to produce an output similar to that data by modifying its parameters.

The problem described above only serves as an example to understand the different approach between what could be called "traditional" programming and ML. However, the solution of that problem is as simple as a regression and the field of ML can get way more exciting. Present solutions to problems that were unable to be tackled before like Object Detection in Computer Vision have been radically influenced by ML as this is a task humans can do very easily, but it is difficult to tell a computer how to do it by "traditional" programming.

A successful ML algorithm can do many different things, in a recent research brief about AI and the future of work [20] it is said that "the function of a machine learning system can be *descriptive*, meaning that the system uses the data to explain what happened; *predictive*, meaning that the system uses the data to predict what will happen; or *prescriptive*, meaning that the system will use the data to make suggestions about what action to take".

There are three categories for ML:

Supervised ML models learn from labeled data, this helps the models to learn and grow to be more accurate over time. An example of supervised ML can be the problem described at the beginning of the section, or a model trained with pictures of dogs that have been labeled by people and that can later identify dogs by itself.

Unsupervised ML programs look for patterns in unlabeled data. This type of ML is useful for finding patters that people aren't necessarily looking for or that they cannot really perceive on their own. A typical example for an unsupervised ML program is the product recommendation system after buying or adding to cart another product in sites like *Amazon*. This program recognizes that people that buy an specific product are likely to buy any of the recommended products.

Reinforcement ML learns through trial and error to take the best action by establishing a reward system. It is commonly used to train models on how to play games, some even better than humans; or to make self-driving vehicles possible by telling the car when it made right and wring decisions helping it learn over time. The field of RL can be of deep interest for its application in Air Navigation, as well as it is being used for self-driving cars. This is because it can learn complex policies in high dimensional environments and work over real life or high accuracy simulations where data might be difficult to get or process. The topic of RL will be covered more in-depth in a future chapter.

One of the reasons there are great ML models nowadays and the exponential growth it has had comes from the advancements in computing power. This has also meant having better processors available to the general public, contributing to a big growth of the AI community that subsequently has meant a lot of open-source research and an event better development of the subject.

As explained in previous paragraphs ML models learn from examples, which is basically data, which leads the other main reason this kind of technology has grown so much: the huge amount of data available. In a world where the internet is present in every aspect of daily routine, data collection means having the ability to know unlimited information about consumers and many other aspects. ML helps make use of that vast amount of data, for commercial purposes and for research and it is a pillar on which ML stands.

An example of how such great amout of data has been used is, for example, *Google Translate*. That was trained on huge amouts of information available on the web and in different languages in order to offer effective translations.

ML models have not only meant doing some tasks in a more effective or automated way than humans, but also some perform actions that humans could never really achieve, like *Google Search*. The amount of information that can be browsed in no time and the understanding of the information that might be the most relevant to the user is just something a human cannot do, so it is not an example of computers putting people aside like it is a lot of times feared, but actually an example of computers doing things that could have never been feasible if done by humans. [21][20]

ML derives into many kinds of algorithms and approaches, which many are not part of the scope of the project nor really help in its development. There are many different kinds of fields of study and applications like Natural Language Processing, that aims to understand natural language as spoken and written by humans, helping create coherent text, translations and even conversations, being great examples *Siri* or *Alexa*.

Uses of ML that are worth mentioning are: recommendation algorithms, image analysis and object detection, fraud detection, chatbots, self-driving cars and even medical diagnostics and imaging.

2.3 Deep Learning (DL)

Deep Learning is one specific kind of ML that has exponentially grown in recent years. This type of ML is based on how a human brain works in a way that we have neurons, or cells, that individually compute a result to their input and pass their result to the next series of neurons. These are also know as perceptrons. The following discussion will be centered in concepts that lead to the understanding of neural networks in a basic way as the implementation of these algorithms is done using high level approaches provided by different ML programming frameworks.

2.3.1 Neurons and Artificial Neural Networks (ANN)

If a single neuron is taken, the calculation that takes place inside if it is a linear combination of its inputs that is then passed to an activation function before becoming the output of the neuron. It is interesting to observe that a neuron that has no activation function is just a linear regression that in the case of having multiple inputs would become a multiple linear regression. In the figure 2.1 the basic structure of a neuron can be observed. From the figure we see that a neuron can have any number of inputs and each one of them will be part of a weighted sum to which a bias term is also added, the result of all this function is then passed to the activation function and the output of the activation layer is then considered as the output of the neuron. Activation functions will be covered later in the document. A neuron "learns" by modifying its weights and bias in order to fit an input to a desired output as also mentioned earlier in the chapter.

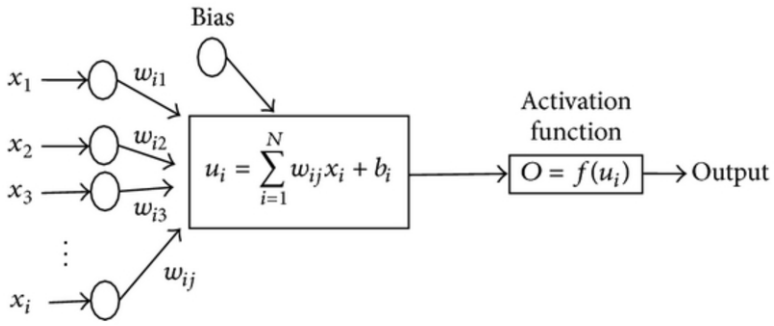

Figure 2.1. The basic structure of an artificial neuron [1].

From this explanation a basic understanding of what a neuron is can be obtained, so guessing from its name and ANN is a set of neurons that connect with each other. A NN typically consists of an input layer, a number of hidden layers being the ones that are intermediate, and an output layer. As explained before this structure is inspired by the human brain and a basic diagram can be seen in the figure 2.2.

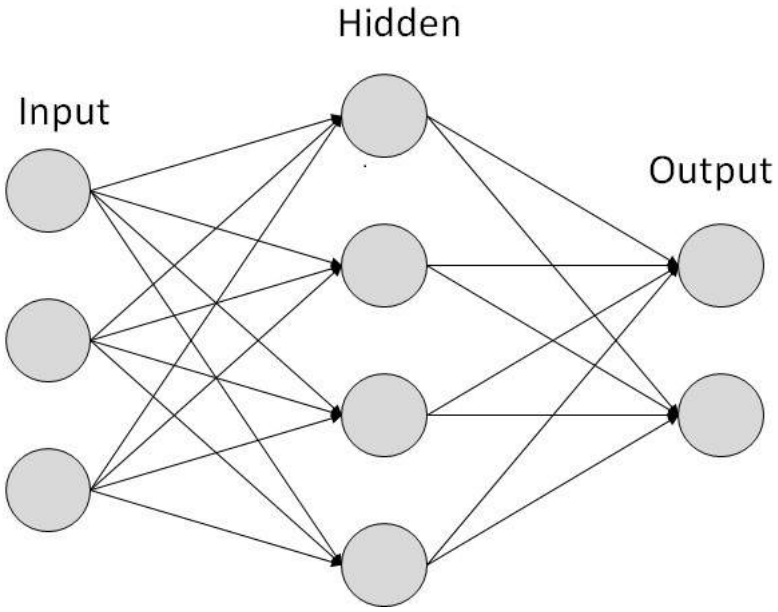

Figure 2.2. Simple ANN [2].

Deep learning comes from the concept of Deep Neural Networks which are NNs that have more hidden layers. An example can be seen in the figure 2.3.

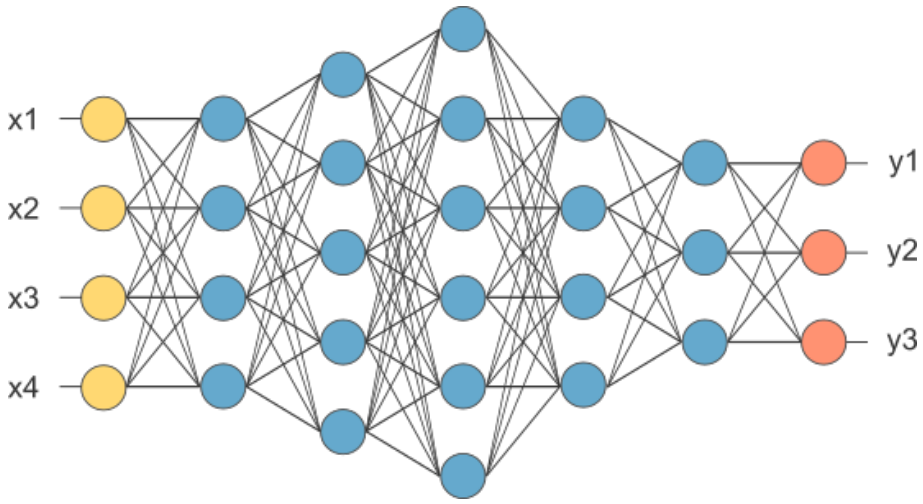

Figure 2.3. Simple DNN [3]

The advantage of using DL is the benefit of not really having to know much about the input as it can learn representations and features directly from it. Also, a good reason why deep learning is so used currently is that it allows for better generalization and more complex models that learn from much bigger datasets, which are really common to find know. A problem with traditional ML, also NNs, is that generalization is sometimes not easy to achieve due to the problem of overfitting, which is essentially adjusting your model too much to your training data resulting in a poor performance when receiving inputs that has not previously seen.

Now that an introduction to NNs has helped understand the structure and basic principles behind them, a brief look into activation functions and the training process should offer a decent general knowledge behind how these algorithms work.

2.3.2 Activation functions

The functionality of the activation function inside a neuron is to determine its output in terms of its combination. It is also the function that the NN represents and some of the most typical are: the *hyperbolic tangent* activation function, the *logistic* activation function and the *rectified linear* activation function [22].

The hyperbolic tangent activation function, represented in equation (1), is a sigmoid function that makes the output of the neuron vary from -1 to +1. Its output can be seen in the figure 2.4.

$$activation = tanh(combination) \tag{1}$$

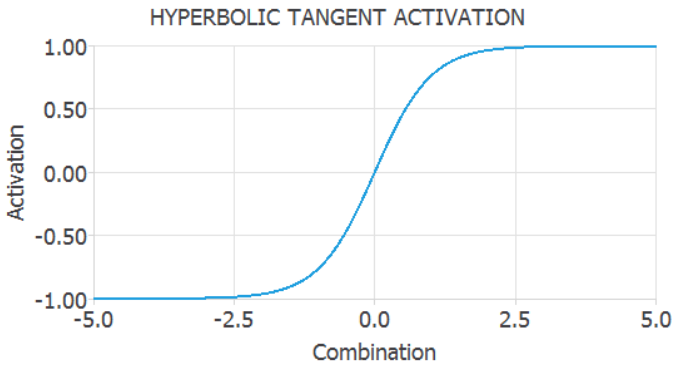

Figure 2.4. Hyperbolic Tangent [4]

The logistic activation function, seen in equation (2), is also a sigmoid function but it varies from 0 to 1. It is represented in the figure 2.5.

$$activation = \frac{1}{1 + e^{-combination}} \tag{2}$$

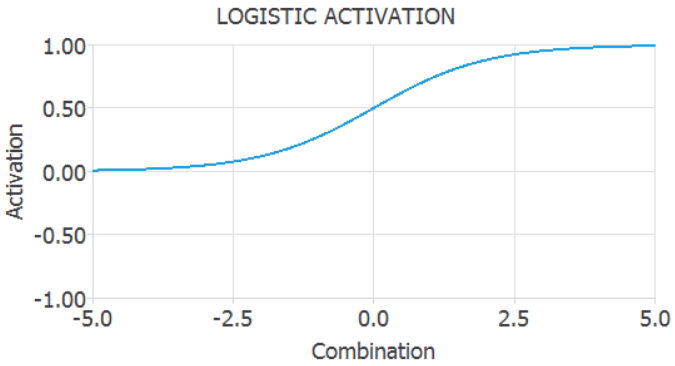

Figure 2.5. Logistic Activation [5]

The rectified linear activation function, also known as ReLU is one of the most, if not the most, used activation functions. It is 0 when the combination is negative and equal to the combination when it is positive, as seen in equation (3). ReLU helps train models faster as it helps with the vanishing gradient problem while training. Vanishing gradient is when the gradient gradually starts becoming smaller after each iteration to a point that is makes impossible updating the parameters in an effective way, the optimization of the parameters will be discussed in the next section. The figure 2.6 shows the output of the function.

$$activation = \begin{cases} 0 & combination < 0 \\ combination & combination \geq 0 \end{cases} \tag{3}$$

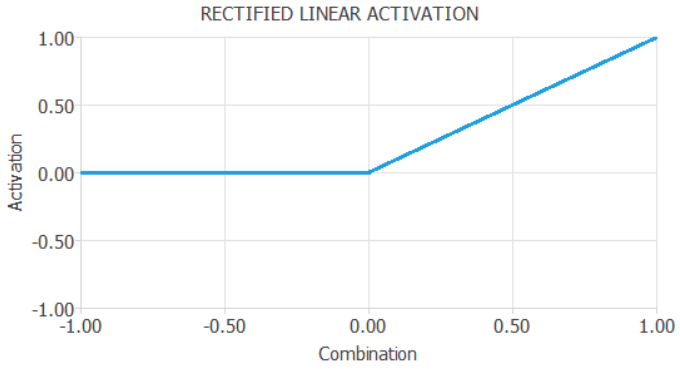

Figure 2.6. Rectified Linear Activation [6]

2.3.3 Training

The *loss index* has a vital role in the use of NNs. It provides with a measure of the quality of the representation required to learn and defines the tast to be done. The loss index has two terms: the error term and the regularization term, as it can be sen in equation (4) [23].

$$loss\_index = error\_term + regularization\_term \tag{4}$$

The *error* is the key term in the expression and measures how well the NN is fitting the data. Some of the most important error functions used in the field are: the *mean squared error*, the *normalized squared error*, or the *cross entropy error* amongst many others.

The mean squared error is the average squared error between the outputs of the NN and the real output coming from the data that we want to fit, its expression is seen in the equation (5).

$$mean\_squared\_error = \frac{\Sigma(outputs - targets)^2}{instances\_number} \tag{5}$$

The normalized squared error gets the squared error between the output of the NN and the target output from the where training from data divided by a normalization coefficient. If the error is close to 1 is predicting the data "on the mean", if the value of the error is 0 then the prediction is perfect. The equation (6) shows the expression for this error.

$$normalized\_squared\_error = \frac{\Sigma(outputs - targets)^2}{normalization\_coefficient} \tag{6}$$

The cross-entropy error is used for problems of binary classification. This means that the output that we expect from the NN is either 0 or 1. The advantage of using this error is that it penalizes with high error is a target is label mistakenly. The equation (7) shows this type of error.

$$cross\_entropy\_error = -\Sigma(target * log(output) + (1 - target) * log(output)) \tag{7}$$

A regular solution is one that for small changes in inputs, small changes in the output are obtained. For non-regular problems a solution is to control the effective complexity of the NN and this can be done by the usage of a regularization term in the loss index.

Usually the regularization term measures the values of the parameters of a NN. By adding that term to the error the size of the weights and biases are reduced and the response is forced to be smoother. The most used types are L1, equation (8), and L2, equation (9), regularization.

$$l1\_regularization = regularization\_weight * \Sigma\, |parameters| \tag{8}$$

$$l2\_regularization = regularization\_weight * \Sigma\, parameters^2 \tag{9}$$

The regularization weight has to be tuned to fit the desired solution. If the solution is too smooth that it has to be decreased, and the opposite if the solution oscillated too much.

The loss index depends on the function that the NN represents. It can be visualized as a surface with the parameters as coordinates as seen in the figure 2.7. The

learning process for a neural network consists in tuning the parameters of the NN in order to find the minimum value of the loss index.

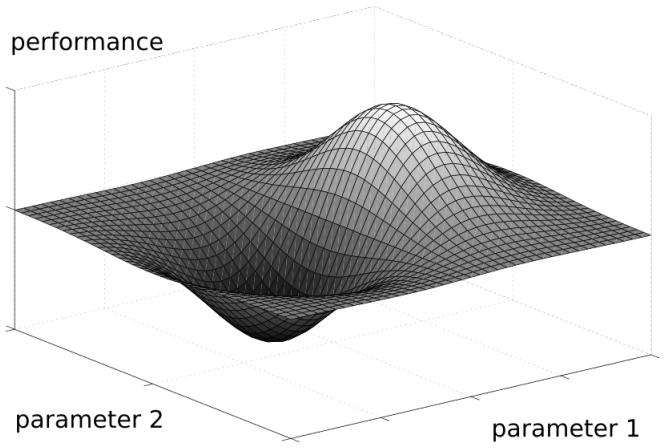

Figure 2.7. Performance Function [7]

As already mentioned, the training of NN consists in searching for set of NN parameters that minimize the loss index. For this we know that if the NN is at the minimum of the loss index, the gradient will be 0.

The loss is generally a non-linear function of the parameters of the NN. Finding closed optimization algorithms for the minima is not possible and the solution is carried out through the parameter space by a succession of steps or epochs. At each epoch the loss will decrease by optimizing the value of the NN parameters according to a certain algorithm.

So, to train a NN, we start from a set of parameters that are usually initialized as random numbers and a new set of parameters is generated at each iteration in order to reduce the loss index.

Usually the optimization stops when a given criteria is reached, some common are:

- Loss has reached a goal value
- A maximum number of epochs is reached
- The loss improvement in one epoch is less that a set value

The optimization algorithm is what determines how the adjustment of the parameters of the NN is carried out. There are many different types of optimization algorithms that vary in computation and storage requirements and no one is actually suited for every single case. Some examples of the most common are the *gradient descent*, the *stochastic gradient descent*, or the *adaptive linear momentum* algorithm.

Gradient descent, equation (10), is the simplest optimization algorithm. This algorithm updates the parameters at each epoch in the direction of the negative gradient of the loss index respect to the parameter it is trying to optimize. There is a constant called *learning rate* that specifies how much the gradient is affecting the new parameters and it is usually set to a certain value chosen before optimizing or can be adjusted at each epoch using line minimization.

$$new\_parameters = parameters - loss\_gradient * learning\_rate \tag{10}$$

Stochastic gradient descent, equation (11), updates at every epoch the parameters many times using batches of data.

$$new\_parameters = parameters - batch\_gradient * learning\_rate \tag{11}$$

Stochastic gradient descent has many advantages as it makes it easier to fit in the memory as only a single training example is being processed by the network at a single time. Also, for the same reason, it is computationally fast. And, for larger datasets, it can converge faster as it updates the parameters more frequently.

The *Adaptive linear momentum*, or ADAM, optimizer is very similar to gradient descent, but it uses a more sophisticated method to calculate the training direction and helps produce a faster convergence. This algorithm also helps to avoid getting stuck at the local minima of the loss. The exact method used by the algorithm is out of the scope of the project as it usually already implemented in ML frameworks.

2.4 Reinforcement Learning (RL)

Reinforcement Learning is a really popular subject in the field of ML and its popularity does not stop growing. The basic definition of it was given in one previous section but the goal of this section will be to understand at least the basics of it.

As a reminder, RL is one type of ML technique that allows an agent to learn from an interactive environment. That is, instead of providing previously stored data, we allow the agent to interact with an environment in order to, by trial and error and feedback from its experiences, learn how to behave in a proper way. Both supervised learning and RL use mapping between input and output, but in supervised learning the feedback provided to the agent is a correct set of actions for a given task and, in RL, punishes and rewards are used as signals for positive or negative behavior that guide the agent towards learning.

Comparing unsupervised learning and RL, unsupervised learning tries to find similarities in data, however in RL the goal is to find an action model that maximises the total cumulative reward collected by an agent, in the figure 2.8 we see the generic feedback loop for RL [24].

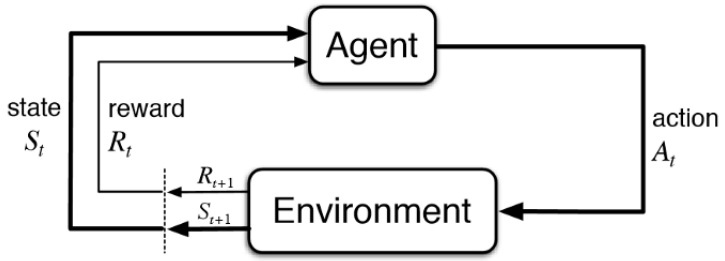

Figure 2.8. Generic Reinforcement Learning Loop [8]

The basic components of RL are:

1. **Environment**: "world" in which agent acts
2. **State**: current data about the agent like, for example, position
3. **Reward**: feedback in the form of points
4. **Policy**: the method that describes what action to take given a certain state
5. **Value**: future reward expected to receive given a certain action in a given state

A really helpful way to understand a RL problem can be through games. For example in the game Breakout you interact with a ball with a small paddle to make it bounce into the bricks at the top and win points. However the player gets a penalty if it misses the ball. In this case, the game screen would be the environment in which the agent acts by moving the paddle to the left, right, or keeping it in place. The agent will receive reward by hitting the bricks and a penalty for missing the ball. The states would be the location of the agent in the world and the total cumulative reward is the agent beating the game. A screenshot of how the game Breakout looks is given at figure 2.9.

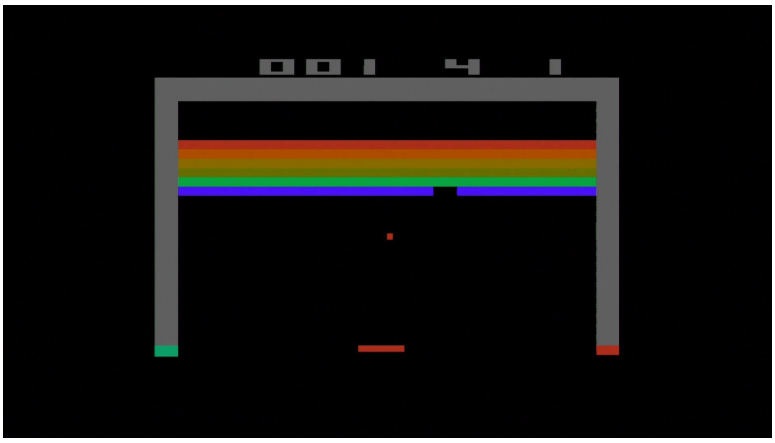

Figure 2.9. Atari Breakout (2600) Screenshot [9]

To build an optimal policy that behaves the way it is needed the agent faces the dilemma of exploring new states but, as well, optimizing the overall reward. This is called the *Exploration vs Exploitation* trade-off. To be able to balance both, the agent may want to involve in short term sacrifices and that way the agent can collect more information to make the policy better and provide the best overall decision in the future. This topic will be mentioned and better explained in future sections.

2.4.1 Markov Decision Process (MDP)

The key abstraction in Reinforcement Learning is the Markov Decision Process. The MDP is the classically formalized process of sequential decision making, where actions taken not only influence immediate rewards, but also subsequent states and thus future rewards. In each time step the process is in a given state, the agent chooses an action to make dependent on that given state which in turn receives a reward and leads the agent into the next state. It is key to know that for each action there is

always a probability that certain action leads to a certain state, which is the state transition probability. [25]

An important assumption is the *Markov Property*. A state is a Markov state if it has the Markov Property, which is when the current state contains the information that is needed in order to determine the state transition and it does not depend on previous states.

With the approach given to the problem, it will be seen that the Markov Property is fulfilled because when a flight is simulated the changes given an action can be computed with the current information.

So, an MDP is defined as a tuple of size 4 (S, A, P_a, R) where S is the finite set of different states, A is the different possible actions, P_a is the state transition matrix and R the reward function.

The agent's goal is to maximize the cumulative reward it receives over the whole sequence of actions that it takes. The return is the sequence of reward received over the trajectory, as seen in the equation (12) [16].

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{12}$$

G_t is the return.γ is the discount factor that balances the importance that the future reward will have over immediate reward. T is the time step at the end of the whole sequence of actions. The return gives the total future discounted reward from the current time step, that means that is an accumulation of reward of the future and does not look backward at already received reward.

Another important group of functions that is usually found in RL algorithms are *value functions*. This function estimate if the current state is good or bad to stay at. This is determined by the *expected return*, that is expected cumulative reward in the future. Rewards depend on the action taken by the agent and the behavior that determined which action to take is called *policy*. A policy maps the probability of performing each possible action given a state and is defined as $\pi(a|s)$. The value function can be defined as in equation (13).

$$v_\pi = \mathbb{E}[G_t | S_t = s] \tag{13}$$

$v_\pi(s)$ is the value function of the policy π, G_t is the return and s is the current state. Concurrent with equation (13), formally known as *state-value function* for policy π, the *action-value function* gets defined. The action-value is different from the state-value function in relying in both state and action, seen at equation (14).

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a] \tag{14}$$

In the equation (14), the action-value function, $q_\pi(s, a)$ is the return expected starting from state s, taking the action a and following the policy $\pi(a|s)$.

2.4.2 Bellman Equations

There is still the question of how to maximize the cumulative reward. For that, a policy $\pi(a, s)$ has to be found so that the cumulative reward $v_\pi(s)$ is maximized. As the previous definitions of value functions cannot be solved in a numerical way, to be able to find the optimal policy, a recursive relation is necessary for the state-value and the action-value functions. The recursive relations to be presented use the property of the recursiveness of the return as baseline, the equation equation (15) shows the return expressed as recursive expression with itself.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{15}$$

Considering a case in which we have a set of discretized actions, to determine the current value of this state under the policy $\pi(a|s)$, the value function $v_\pi(s)$ has to be used as seen in the equation (16). The total value of that specific state will be the summation of all the action-value functions weighted by the probability that that given action will have given the policy.

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a) \tag{16}$$

It is key to know, though, that just as $v_\pi(s)$ is dependent on $q_\pi(s,a)$, $q_\pi(s,a)$ is also dependent on $v_\pi(s)$. When taking an action, the action-value function provides with a value for this action and there is also a probability associate with the outcome of the action. The probability of taking a certain action to end up in a certain state is the transition probability, $P^a_{ss'}$. Knowing this, the equation (17) is now the equation for the state-action value function.

$$q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s') \tag{17}$$

The equation (16) and equation (17) can be combined in order to obtain a recursive relation between the initial state and the transition one. If it done for both $v_\pi(s)$, and $q_\pi(s,a)$, we obtain the equation (18) and the equation (19).

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \Big(R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s') \Big) \tag{18}$$

$$q_\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a \in A} \pi(a'|s') q_\pi(s',a') \tag{19}$$

These equations are named the *Bellman Equations* and they express the relationship that exists between the value of the current state and the successor one.

Now it is time to find a method to find the optimal policy, which is the one that maximizes the accumulated reward over time. The policy defines how the agent behaves and has a great influence on the value of both value functions. It can be said that a policy π is better or equal than a policy π' when the expected return is greater or equal to the one of π' in all states.

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in S$$

There is always going to be a policy that complies with the statement above and will be called an optimal policy. Multiple optimal policies can exist but they will all have the same value function. The optimal state-value function and the state-action value function can be defined as $v_*(s) = \max \pi v_\pi(s)$ and $q_*(s,a) = \max \pi q_\pi$ and finding these, but especially q_*, help finding the optimal policy.

The Bellman equations mentioned before can be rewritten for the case of an optimal policy resulting in the *Bellman Optimality Equations* that can be seen in the equation (20) and equation (21).

$$v_*(s) = \max_a \Big(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \Big) \tag{20}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a') \tag{21}$$

The absence of $\pi(a|s)$ comes due to the fact that, under an optimal policy, simply taking the action that results in the maximum expected return is an optimal policy. When the selection of the action is done as to always obtain the maximum possible value it is called a *greedy* policy. Once $v_*(s)$ or $q_*(s,a)$ are found, solving the MDP is quite simple; acting greedily with respect to the value functions will be the optimal policy.

For computing the optimal policy a series of algorithms called *Dynamic Programming (DP)* are needed. However, DP algorithms are not usually practical as they require knowing the perfect model of the environment as well as its computational expense although all other methods that try to find the optimal policy can be seen as an approximation to DP with less computational effort and without perfect knowledge of the environment's model. The idea behind DP is to make use of the value functions in order to structure the search for an optimal policy.

The first step in DP is being able to compute the state-value function v_π for a policy $\pi(a|s)$. Determining the state-value function for a given policy is called *policy evaluation*. Converting the equation (18), which is one of the Bellman equations into an iterative update rule gives the equation (22) given that the policy is stationary.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \Big(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \Big) \tag{22}$$

As with all iterative processes, initial conditions are needed in order to being able to calculate. The condition can depend on the problem but, usually, $v_0(s) = 0$ for all $s \in S$. In general $v_k \to v_\pi$ for $k \to \infty$.

In policy evaluation there is a method for measuring the quality of a certain policy using iterative solving of the state-value function. This method also helps

to find better policies. If the state-value function v_π is fully determined under a random policy π, for each state s and assessment is made to see if an action a that is $a \neq \pi(s)$ and results in a better behavior can be found. It just happens to be that the action-value function $q_\pi(s, a)$ does exactly that, meaning that if there is an action to be selected that provides with more value and follows policy π, it can be said that selecting this action in this state will always provide more value for the whole trajectory. This is $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, where $\pi'(s)$ is the new policy that will provide with a better behavior, so a new and better policy has been found. This is named *policy improvement*. Having said that, selecting the action that maximizes the value through $q_\pi(s, a)$ can be written as seen in the equation (23). This form of achieving maximization is called greedy optimization policy.

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{23}$$

As now a way of both evaluating the policy and improving it have been found, they can get used in alternating manner to evaluate the policy, then improve it, and so on. This is called *policy iteration* and can be seen in the equation (24), where the E stands for evaluation, and I for improvement.

$$\pi_0 \xrightarrow{\text{E}} \pi_{\pi 0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} \pi_{\pi 1} \xrightarrow{\text{E}} \pi_2 \xrightarrow{\text{E}} ... \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{\pi *} \tag{24}$$

Generally, though, it is not necessary to achieve full convergence in each step as it usually takes several steps when evaluating a certain policy to converge to the true value function. For most application the absolute converged value is not really necessary, only the ordering and the tendency of this values really is. This way, this results in a lower computational cost. A special case for reducing the iterations that it takes policy evaluation is called *value iteration*. Value iteration only does one evaluation iteration and then combined with the policy improvement step it reduces the multiple policy evaluations and subsequent policy improvement to one single update. The equation for value iteration is the equation (25).

$$v_{k+1}(s) = \max_a \sum_{s'} P^a_{s,s'}[R_a(s, s') + \gamma v_k(s')] \tag{25}$$

A relation can be observed between value iteration and the Bellman Optimality equation. Value iteration merges the greedy policy improvement with the update of

the value function under the new policy.

The iterative method described before for the interaction between evaluation and improvements called *generalized policy iteration (GPI)* and it is usually seen in Reinforcement Learning methods. When the updates start to stabilize, the Bellman optimality equations are approximated in a correct manner and the optimal policies and value function are found. DP gives a method of finding in a consistent manner the optimal policies and value functions, but for bigger state representations it requires a lot of complete sweeps and thus using a lot of computational resources. As well, and to really be highlighted, they require the full model of the environment. Having mentioned this, the next section will introduce model free methods which rely on exploring the environment and trying to estimate the value functions and policy based on just the feedback provided by the environment. Model free methods that are built on experience are called *Reinforcement Learning* methods.

2.4.3 Model free methods

In model free methods of RL there is a lack of knowledge about the transition probability of the environment, so the learning is done based purely on the interaction with the environment. Model free methods consist of two categories which are value based methods or policy gradient methods. In value function methods an extension is done to the dynamic programming methods that were discussed in the previous section. In policy gradient methods, which are a more recent development, the policy space is parameterized and estimated in a direct manner.

2.4.3.1 Value based methods

This part of the document will cover a more traditional approach that originates from dynamic programming to tackle model free problems. *Monte Carlo methods* will be explained before and later the concepts that were explained about DL in a past section will be combined with value based solutions to cover one of the state-of-the-art reinforcement learning methods, *Deep Q-learning*. Value based methods go way beyond what will be covered, but it serves as a way to understand the way they work.

2.4.3.1.1 Monte Carlo methods

These methods rely on the experience collected over a complete episode, so each problem that wants to be solved using this kind of methods needs to have a finite length. The difference with DP methods is that where in DP methods the value-functions are directly calculated for all the states, in MC methods the value-functions a learned from the returns on a trajectory in the environment.

Compared to DP methods, in MC not all information is available so the rewards a responsible of estimating the value function $V(s)$. This is done by knowing all the gained reward-state pairs collected during an episode and then calculating the return $G(s)$ for each state during the episode. Once the returns are calculated the value function $V(s)$ is estimated with an average of the returns for each one of the states that have been returned in an episode $V(s) = average(G(s))$. For each episode a single state can be visited multiple times, so there is the choice of only evaluating the value function the first time a state is visited, or taking into account all the times that is has been visited. Usually the first visit approach is preferred and is called *First-visit MC Prediction*. Given that the calculation of the return requires knowing the estimated returns of the states before it, this method requires having to complete a whole episode before any calculations are made. At this point the calculations are made going backwards from the end of the episode to the beginning.

Calculating the action-value function is of more interest than the state-value function as for control problems is of more interest knowing the value that a certain action contributes to the return. The amount of parameters to be estimated for the action value function increases a lot depending of how many actions are available, making it a setback to overcome. Another problem is that when the policy is deterministic for each state only one action-value is estimated. This is clearly part of the exploration problem, which will be talked about later. In order to apply First-visit MC prediction to action-value functions a policy that makes sure of having a non-zero probability for all actions in each state has to be used.

The method for evaluating and improving the policy is as seen in the DP methods, GPI, and can been seen in equation (26)

$$\pi_0 \xrightarrow{\text{E}} q_{\pi 0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q\pi 1 \xrightarrow{\text{E}} \pi_2 \xrightarrow{\text{E}} ... \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_{\pi *} \tag{26}$$

The evaluation step is the same as discussed before, the iteration step is different

as while in DP methods the method is known and, thus, the state-value functions, in MC that is not known as it is a model free method. When using action-value functions though, the policy is reduced easily by taking the action that for each step will maximize reward, but without waiting for full convergence, allowing to truncated iterations. Value iteration in DP was similar to this, which took a single evaluation step and then did an improvement on the policy. In MC, after each episode the returns will be used for evaluating the policy for each state for then improving the policy for each of them.

The problem of exploration has not yet been dealt with and a little knowledge of it is necessary to understand certain approaches taken by the methods that are used in RL. In RL, this problem is key as the way of making an agent behave in an optimal way is usually making it learn action values conditional on subsequent optimal behavior, however to obtain optimal behavior there is a need to explore all of the states, which is sub-optimal behavior. A distinction can be made between two methods of evaluation, *on-policy* and *off-policy*. On-policy methods evaluate or improve the existing policy that is used to make decisions. Off-policy methods use a different policy to get data to the one that is trying to evaluate or improve. An on-policy method that is improved for better exploration is called ϵ-*greedy*. This policy method behaves greedy with a probability of $1 - \epsilon$, and chooses an action at random with probability ϵ. With this method some actions a forced to be sub-optimal and, thus, improving the exploration.

Off-policy methods consist of two different policies, one that is trying to get to be an optimal policy, and a different one to make sure there is exploratory behavior. The policy that is learning to be optimal is the *target policy*, the one that ensure exploration is called *behavior policy* and it is the one used to gather data from the environment. This method requires extra care as the data gathered is not from the target policy and this induces greater variance and an slower convergence.

As it has been discussed before, and being coherent, the target policy is called π and the behavior one b. For off-policy methods the term *coverage* refers to the fact that both policies have to ensure the visitation of the same state-action values. Before moving to discuss off-policy MC control, MC evaluation is considered. The difference between the target and behavior policy require making compensations to correct the different during learning. This is achieved by *importance sampling* which tries to determine the values given by a distribution by using samples of another distribution, in this case the two policies. This method is used as a way of weighting

the returns according to the relative probability of their succession of states occurring under each of both policies. This weighting is called importance-sampling ratio, seen in the equation (27).

$$p_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{27}$$

An extension to this approach is *weighted importance sampling*. This method is different as it uses a weighted average instead of the absolute one. This can be seen in the equation (28) to calculate $V(s)$ with the returns scaled by the weighted importance sampling ratio. The sum is for the first-visited stated over all episodes.

$$V(s) = \frac{\Sigma p_{t:T-1} G_t}{\Sigma p_{t:T-1}} \tag{28}$$

This latter approach is the one that is almost exclusively used in practice.

Like most methods there is an incremental update rule for the weighted importance sampling ratio, seen in the equation (29)

$$V_n = \frac{\Sigma_{k=1}^{n-1} W_k G_t}{\Sigma_{k=1}^{n-1}} \tag{29}$$

Each update requires knowing the previously applied weights W_k. This is done keeping the previous weights in C_n. The rule for updating V_{n+1} and C_{n+1} are in the equation (30) and equation (31).

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n] \tag{30}$$

$$C_{n+1} = C_n + W_{n+1} \tag{31}$$

Knowing all this, the pseudo code for the Off-policy Monte Carlo control in given, combining everything that has been mentioned. This is shown in algorithm 1.

Algorithm 1: Off-policy MC control [16].

Initialize, for all $\in S, a \in A(s)$:

$Q(s,a) \leftarrow$arbitrary

$C(s,a) \leftarrow 0$

$\pi(s) \arg\max_a Q(S_t, a)$

while *True* **do**

  $b \leftarrow$any policy with coverage of π

  Generate an episode using b:

    $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T, S_T$

  $G \leftarrow 0$

  $W \leftarrow 1$

  for $t = T-1, T-2, ...,$ *down to* 0 **do**

    $G \leftarrow \gamma G + R_{t+1}$

    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$

    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$

    $\pi(S_t) \arg\max_a Q(S_t, a)$

    if $A_t \neq \pi(S_t)$ **then**

      exit For loop

    end

    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

  end

end

2.4.3.1.2 Temporal Difference methods

These methods are really similar to DP or MC. Compared to MC, it uses experiences also, and it does not need system dynamics. TD, however, does not wait for a full episode to be completed, but it learns based on the learned estimates.

The policy evaluation in TD is similar to MC. MC waits till the end of the episode to calculate return, but TD does not wait and updates the value function on each step. In the equation (32) the prediction step for MC is shown right above the prediction step for TD. α is the step-size, that is constant.

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_t - V(S_t)] \\ V(S_t) &\leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \end{aligned} \tag{32}$$

Updating based on existing estimates is called the *bootstrapping* method.. The TD equation above is a *one-step TD* as it only bootstraps one step ahead, which is $V(S_{t+1})$. The term $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the error between the old value and the estimate of the new value, it is called *TD error* δ and is term that is used many times in RL.

TD on-policy follows the pattern of GPI like DP or MC did but using the TD method of evaluation. As in MC methods, instead of doing the evaluation with the state-value function, the action-value function is used. This is shown in the equation (33).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{33}$$

The sequence of elements S_t,A_t,R_{t+1},S_{t+1},A_{t+1} that can be seen at equation (33) gives name to this on-policy TD method: *SARSA*.

The off-policy method for TD also exists and it is one of the most used methods in RL. It is known as *Q-learning*. As SARSA does, Q-learning also bootstraps but its off-policy behavior is mostly affected by the $\max_a$ operation. The update rule for Q-learning is shown in the equation equation (34).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{34}$$

The behavioral policy in off-policy TD, Q-learning, is often ϵ-greedy for selection the action A_t, the target policy is greedy. Q-values are stores in a tabular manner.

Q-learning serves as the basis for most value-based DL methods. This is done by using DL as a function approximation.

2.4.3.1.3 Deep Q-Learning

This method relies in applying DL algorithms to Q-value estimation. However performing this task is not straight forward and it would produce poor results unless some modifications are done. RL, as it has been discussed, depends on a reward signal that can often be sparse, noisy or even delayed. That delay between the action and the reward can be really problematic for DL as it was explained in previous sections that most DL methods depend on directly gained feedback. Another problem is the high correlation in data for RL given usually the state returned by the environment highly depends on the previous state. Also, the change in policies means that the data distribution changes and this is something that for DL methods is not wanted, as they assume a fixed data distribution.

The update rule for Deep Q-learning is similar to the one of the Q-learning method, equation (34), but the notation changes slightly. This can be seen in equation (35).

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s'a') - Q(s,a)] \tag{35}$$

For being able to apply DL as a function approximator, the action-value function $Q(s,a)$ has to be parameterized: $Q(s,a;\theta) \approx Q(s,a)$. The weights θ are used to parameterize $Q(s,a;\theta)$. Updating the weights is done by the use of a loss function where a gradient descent method or similar can be used. This loss function is defined in the equation (36).

$$L_i(\theta_i) = (y_i - Q(s,a;\theta))^2 \tag{36}$$

The i in L_i and y_i denotes the iteration, where L_i will be the loss and y_i the target for each iteration. In this case, the target is the left side of the TD error, $y_i = r + \gamma \max_{a'} Q(s'a';\theta_{i-1})$. The target has to be noted that it is always kept fixed with respect to older parameters in order to stabilize the network updates and

diminishing the issues with non-stationarity and correlation that were discussed at the beginning of the Deep Q-learning section. In order to differenciate the loss function with respect to the network weights θ_i so that they can be updated, as explained in the DL section, the equation (37) can be seen.

$$\nabla_{\theta_i} L_i(\theta_i) = (r + \gamma \max_{a'} Q(s'a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a, \theta_i) \tag{37}$$

Once the discussed solution is in place, the utilization of Deep Q-learning still is not easy as the problems mentioned at the beginning of the section regarding the application of DL to Q-learning are still there. However, this method has returned many state-of-the-art results, even outplaying humans in several Atari games.

A key to solving the Atari problems were to use a *replay memory* that stored actions and transition states from the past so the learning could happen by sampling from this database. This method is called *experience replay* and permitted randomly selecting sampling from the database and, thus, breaking the correlation and in turn helping with the non-stationarity. Also, a generalization between games was achieved by using a *Convolutional Neural Network CNN* as the function approximator so as to use the screen as an input and obtain an interpretation of it. These kind of solutions are individually crafted in order to implement Q-learning to each kind of problem if trying to correctly approach the limitations and the objective that it trying to be solved. No generalization can always be implemented and, thus, the programmer has to be fluent in these kind of methods in order to understand its setbacks and figure their way around it.

2.4.3.2 Policy Gradient methods

The discussion until now has presented only methods that work around the estimation of the value function, meaning that the policy takes an action in the direction of highest return. The methods to be presented now optimize the policy directly: $\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$. In which θ are the wights of the policy. In order to optimize the policy, equation (38) is given. α is the step size and $J(\theta_t)$ is an arbitrary performance measure. In order to be able to optimize using the gradient, the underlying function approximation has to be differentiable.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{38}$$

Usually, the arbitrary performance measure is a value function. The combination of the use of a value function with a policy estimation method results in a series of methods called *actor-critic*, in which the value function is considered the critic dictating how good or bad the state is to be in, and the actor is the policy that determines which action to take.

Parameterizing the policy provides with some advantages over greedy policies, usually found in value based methods. One of the advantages is that the policy space can be approximated by stochastic function approximation. Depending o the problem, stochastic function approximation enables the policy to converge to a stochastic optimal policy. Another one of the advantages is that sometimes estimating the policy itself is easier than estimating the complete value function for the whole problem. A final advantage is that when optimizing the policy, the probabilities of the actions change smoothly between updates when compared to value based methods. In value based methods, when a certain action-value surpasses another one, under a greedy policy, this new action is chosen instantly and can result in an erratic behavior.

To provide with weight updates the performance measure has to be defined, an example of which can be the value of the starting state s_0, where $v_{\pi\theta}(s_0)$ is the true value function for the policy π_θ. This can be seen in the equation (39).

$$J(\theta) \doteq v_{\pi\theta}(s_0) \tag{39}$$

The performance measure will be different for different problems and can be constructed as seen. The policy has to have an effect on this performance measure, specifically, the gradient of the performance measure with respect to the parameters. From the *Policy Gradient Theorem* that establishes a proportional relationship between the gradient of the performance measure and the gradient of the policy, a definition can be given and it is shown in equation (40).

$$\nabla J(\theta) = \mathbb{E}_\pi[G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}] \tag{40}$$

In equation (40), the S_t and A_t are the samples actions and visited states. G_t is the episode return. If equation (38) and equation (40) are combined, it results in a classic policy gradient method called *REINFORCE* and it is shown in equation (41).

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} = \alpha G_t \nabla_\theta \ln \pi(A_t|S_t,\theta) \tag{41}$$

The parameters in equation (41) move in the direction of space that increases or decreases the probability of repeating that action, and it is weighted by the received return. That means that high returns for certain actions are updated more heavily. The division by the actual probability of selection the action results in balancing out the updates depending on the probability of selection. If not, actions that have already been visited frequently would be updates in an unfairly manner due to their higher change on taking that action.

The method *REINFORCE* uses the whole return at each time step, similar to MC, so updates are only done at the end of each episode.

The *REINFORCE* method can be extended using a *baseline* with which the action value can be compared to. This later will lead to the introduction of *actor-critic* methods. When using action value for the updating of the parameters, the interest is not the absolute value, but the action that is relatively better compared to the other ones. If absolute values are used, the variance between states can be high. By introducing the baseline, the updates are normalized reducing the variance at each update step. The equation (42) shows the updates version of *REINFORCE with baseline* in which $b(S_t)$ is a arbitrary baseline value depending on the current state.

$$\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{42}$$

The baseline can be taken as a random value, but it should have the property that when in a state where the actions are high value, the baseline should be also high value, and vice-versa. For this, the use of an estimates state value $\hat{v}(S_t, w)$ is a good approach. The w are the weights for the state value estimation. As the *REINFORCE* algorithm is a MC method, the value function can be estimated using the same approach as MC. The use of an approximated state value function as the baseline is related in a closely manner to actor-critic methods that will be discussed next.

2.4.3.2.1 Actor-Critic

The *REINFORCE* with baseline is not considered an actor-critic method even though is very similar due to the fact that the value function is used as a baseline with respect to the return but not as an estimate. As well, a method like MC that depends on complete episodes has a really slow convergence and are not suitable for online estimation. So, by replacing the return with a value function estimate, results in a one step actor-critic method. The value function is updates with bootstrapping, like in TD, so it has to be recalled that $\mathbb{E}(G_t) = V(S_t) = \hat{v}(S_t, w)$. The one step actor critic method is shown in equation (43). δ_t is the TD error that was discussed in past sections.

$$\theta_{t+1} = \theta_t + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{43}$$

This is an online process that in every step can be updated. The term $(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))$ is the critic in this case, and is the esstimation of the value function. The actor is the policy estimation $\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$. This method is the basis for actor-critic.

The actor and the critic are usually estimated by the means of a NN, so by using DL, and the basics of policy gradient methods are explained. In order to apply DL to policy gradient methods there are a few challenges to overcome, like the introduction of more steps to provide stability for gradient updates. This will be covered in the next section, plus more on the application of DL to policy gradient.

2.4.3.2.2 Policy optimization

This section cover two of the most popular policy optimization methods: *Trust Region Policy Optimization (TRPO)*, and *ProximalProximal Policy Optimization (PPO)*.

TRPO provides a method that help guarantee policy improvement for non linear approximation methods with thousands of parameters. For this, TRPO's goal is to find a way to regularize the step size, something required for complex non-linear policies. If a constant step size was to be used, the result would probably not converge or result in local optima, and especially if the policies are non-linear. Also, if a step size that is to small was to be used, it would probably never converge.

To the previously defined variables, a series of additions are going to be made.

The first one is going to be the *advantage function*, which is an often used substitution for the state-value function or the action-value function. The goal of the advantage function is to normalize the value function's value and, for this, it subtracts the state-value function from the action-values like the equation (44) shows.

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{44}$$

The advantage function is A_π, then $Q_\pi(s,a)$ is the action value function and $V_\pi(s)$ is the state-value function. It is important to know that the value functions are discounted.

Another term to introduce is the *policy performance* η, which is defined as the discounted expected reward, seen in equation (45).

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,\dots}\left[\sum_{t=0}^{\infty}\gamma r(s_t)\right] \tag{45}$$

In the original paper [26], the authors used a a really useful identity that expresses the performance of one policy in terms of another policy and the advantage function. This is shown in the equation (46), where π and $\tilde{\pi}$ are policies, and A_π is the advantage function determined by π.

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,\dots\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right] \tag{46}$$

This identity is then expanded to resolve around states instead of steps, which is more of the nature of RL. The equation (47) is the result of this, where $\rho_{\tilde{\pi}}(s)$ is the discounted visitation frequency.

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s)\sum_a \tilde{\pi}(a|s)A_\pi(s,a) \tag{47}$$

What the equation (47) means is that when the expected advantage at every state s is not negative, the policy performance of $\tilde{\pi}$ will be at least equal or higher than the policy performance of π.

It has to be noted that on the equation (47) the new policy has a dependency on the state visitation frequency. When using information from the old policy this is difficult to obtain, so there is a need for an approximation for the state visitation

frequency. For this, the state visitation frequency that is used is from the old policy $\rho(\pi)$ instead of $\rho(\tilde{\pi})$ to work as an approximation. There is proof that for a sufficient small policy improvement step, the derived identity will still hold, but that means that there is a need to determine the size of this step. In the equation (48), the final local approximate identity is given depending on the mentioned constraints.

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{48}$$

Last paragraph stated the need to now know how to determine the step size that is sufficiently small to not break the policy improvement guarantee, but still assures that the policy improves. The authors of the paper *Trust Region Policy Optimization* [26] utilize KL divergence between the old and new policy, giving equation (49). KL divergence measures the difference between two probability distributions and by using it, it allowed the theorem to be of practical use for most policies.

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C\dot{D}_{KL}^{max}(\pi, \tilde{\pi}) \tag{49}$$

$D_{KL}^{max}(\pi, \tilde{\pi})$ is the KL divergence between the two policies, $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and $\epsilon = \max_{s,a} |A_\pi(s,a)|$. With the combination of the KL divergence and the variable C, the lower bound approximation of the performance function is defined. Before the usage of this equation is discussed, another term has to introduced. If $M_i(\pi) = L_{\pi_i}(\pi) - C\dot{D}_{KL}^{max}(\pi, \tilde{\pi})$, combining it with equation (49), results inequation (50).

$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M(\pi_i) \tag{50}$$

The equation (50) shows that the actual policy performance will always be higher or equal to that of M_i. So, if M_i is maximized, this will result in a policy performance improvement. This algorithm is called *minorization-maximization (MM).*

As parameterized policies are learned, the previous notations concerning π have to be substituted by the policy parameters θ. The objective function, then, will result in the maximization of $M_i(\theta)$, parameterized by the policy parameters θ and is shown in equation (51).

$$\underset{\theta}{\text{maximize}} \left[L_{\theta_{old}}(\theta) - C\dot{D}_{KL}^{max}(\theta_{old}, \theta) \right] \tag{51}$$

The authors also noted that if using C to penalize the function, the updates would result to be too small to allow for convergence. Determining a constant C that would give good performance resulted to be pretty difficult due to the dependency on the KL divergence. So, instead of penalizing the Kl divergence, a constraint is set on it: $D_{KL}^{avg}(\theta_{old}, \theta) \leq \delta$. It has to be pointed the usage of average KL divergence as the max KL divergence is unstable. Also, there is no upper bound for the KL divergence, threatening with instability. This result in the final constraints of Trust Region Policy Optimization, shown on equation (52).

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, L_{\theta_{old}}(\theta) \\ &\text{subject to}\, \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{52}$$

Now that the theory is explained, a practical objective function can be defined so as to optimize it subjected to the theorem provided by equation (52). From equation (48), if the objective function is rewritten in a more convenient way, the result is shown in equation (53).

$$\begin{aligned} \Delta\eta &= \sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{old}}}(s,a) \\ &= \sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_{\theta_{old}}(a|s) \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right] \end{aligned} \tag{53}$$

The first part, $\sum_s \rho_{\pi_{\theta_{old}}}(s) \sum_a \pi_{\theta_{old}}(a|s)$, gets determined by the episode returns of the older policy. The part to be optimized is inside the brackets, $\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right]$. If equation (52) and equation (53) are combined and converting it to a sampled based approach like usually in RL, the equation (54) is obtained.

$$\begin{aligned} &\underset{\theta}{\text{maximize}}\, \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right] \\ &\text{subject to}\, \hat{\mathbb{E}}_t \bar{D}_{KL}(\theta_{old}, \theta) \leq \delta \end{aligned} \tag{54}$$

The method that TRPO uses to optimize this objective function is to use MC roll outs to determine state visitations, Q-values, and KL divergence estimate. To optimize the parameters a conjugate gradient algorithm is uses, requiring a local approximation of both the objective function and the constraint of the KL divergence.

For continuous state and action problems TRPO has shown great results, also for the Atari games mentioned in the Deep Q-learning section.

Computational requirements are really heavy for TRPO, and it is also incompatible with some of the methods used in DL. However, the Trust Region approach has proven to be worthy to the point that Trust Regions as an idea is used extensively in RL.

This is where **PPO** is introduced. PPO is less computational expensive method than TRPO is, but is also easy to implement and is built on the foundations of TRPO.

PPO is a simplification of the TRPO method that allows to use standard gradient descent methods compared to TRPO that needs more extensive methods like conjugate gradients. Two general ideas are behind PPO. TRPO applies a constraint on the update step with the KL divergence between two policies. PPO, however, instead of applying a constraint, it limits the update step depending on the probability ratios between the old and the new policy. From equation (54), and with the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, a new function is made that instead of using the constraint, limits the update step due to the constraints invoked on $r_t(\theta)$. This can be seen in equation (55)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta)), 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right] \tag{55}$$

The parameter ϵ is a tunable hyper parameter. The min operator takes either the clipped or unclipped probability ratio. The figure 2.10 shows the effect that the modified loss function has. When a certain action, under a new policy, has a high chance of being executed with respect to the old policy and the associated advantage function is positive, it can result in a large update step when it is not unbounded. With this update rule, the side of the step is limited by the clip boundaries that in this case are $[1-\epsilon, 1+\epsilon]$. This way more conservative steps are applied. When there is a negative advantage function and a probability ratio is well under 1, the same thing happens.

The paper in which PPO was presented [27], showed that there were significant performance gains compared to other policy gradient approaches. If we combine that with the easy implementation and the ability to use more common gradient optimization methods makes PPO an state-the-art policy optimization method that is widely used in RL.

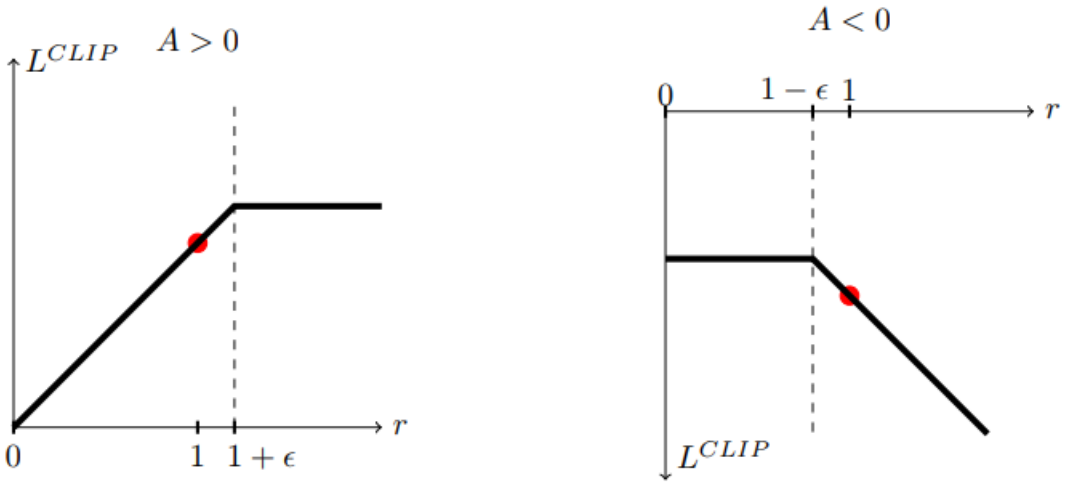

Figure 2.10. Clip parameter illustration [10]

2.4.4 Problems of RL

RL is unstable and has difficulty converging towards an optimal solution. In a scenario where there's not one single agent but several of them, it is called multi-agent, and the problem gets worse. When implementing RL there are a few obstacles that have to be taken into account and they will be discussed now.

The *Curse of dimensionality* comes from the fact that the state and action space grow a lot when the problem complexity increases. When the state and action spaces are continuous these increases even further. Generalization by function approximations line NNs is possible, but it is easy to imagine that complex problems quickly ask for a lot of computational power to be solved, as well as better and more efficient function approximation. This usually comes at the cost of increased variance or other issues with stability.

The exponential increase of the space ties closely to another problem: *explore or exploit* dilemma. This problem has been stated before in past sections. To build a good representation of the environment a lot of states have to be visited, but this means that as the algorithm trains and a better and better policy is obtained, there is a lot of chances that many of the states stay unexplored and, thus, potential better policies have not been found. This means that the solution found was stuck in local optima. Of course these is tied with how many states there are as, the more, the worse. This is where the dilemma comes from as it is not easy to judge when to explore or to exploit the current knowledge in search of the optimal solution.

Another problem is *credit assignment*. This problem comes from the fact that is not easy to evaluate which action caused a certain reward. This is something that can be partially dealt with by decreasing the sparseness of the reward, but this means customizing a reward too much and thus giving the agent too much information, this can lead to the agent being constricted and learning from you input instead of finding more exotic solutions that might come if not provided with that much information. This, in turn, devalues the usage of RL, as one of the main attributives is essentially this.

A problem that comes when dealing with multi-agent problems that the environment is non-stationary. When having several agents interacting the agent not only receive the impact of their own actions, but also the of all the other agents. This, in turn, makes the learning difficult as the action performed by an agent might mistakenly be attributed to more agents than himself only. Also, the agents update their policy each time step, increasing unpredictability from the point of view of a single agent. This non-stationarity of the environment means that the Markov Property is violated, and for that reason convergence to the optimal policy is not assured.

Another issue that arises from multi-agent problems is the way conflicts are resolved. In multi-agent the distinction can be done between cooperation and competition, which leads to having either a total reward being more prevalent, or the individual one.

Another factor is the amount of information that is available between the agents. In a cooperative multi-agent approach, information about the other agents is key to obtain a fully cooperative behavior, for example, when trying to avoid a crash, if the knowledge about the action the other agent is taken is available, then the action to take will be according to the one observed from the other.

Also regarding the amount of information is the observability of the system. It is not realistic the the agent has full observability of the system and, thus, most likely the system will be only partially observable. One of the reasons why this is the case is that having full observability would make the state space huge. And the problem with this was discussed earlier in the section.

Chapter 3

How to work with RL

This chapter shows how to implement a solution using the ideas and software presented in the previous chapter.

3.1 *RLLib*

RLlib is part Ray, which is a framework that "provides a simple, universal API for building distributed applications" [28]. Ray provides with simple primitives for building and running distributed applications, and this way it enables users to parallelize single machine code without many or no changes to the code. Ray includes a large ecosystem of applications, libraries and tools on top of the core Ray, one of which is RLlib, that enable complex applications. The framework has a Python, Java and a C# experimental API. This project will make use of the Python API.

As most, if not all, the code required to run the experiments is done with RLlib only, no further explanation of Ray is necessary. However, there is great documentation that is kept updated and that can be found online from one of the references [28].

"RLlib is an open-source library for RL that offers both high scalability and a unified API for a variety of applications" [28]. This library is a really popular library and we can see examples of it being used in companies like JPMorgan, which utilizes it to power trading models [29].

RLlib supports a variety of ML frameworks like TensorFlow or PyTorch, but the internals are mostly non-dependant on any of these frameworks.

The RLlib API is quite straightforward and simple thanks to great abstraction, especially for simple scenarios. However, it is also possible to use the library in a much more low-level way, providing with a platform for experimentation and development.

3.1.1 Environments

RLlib is compatible with OpenAI Gym, which is "a toolkit for developing and comparing RL algorithms" [30]. Gym offers a simple API to build your own custom environments or test some existing ones. This is done by inheriting a Python class with a series of functions, that are expected to be recognized and accessed later by the training API, and that will have to be overwritten to return a given output that is also expected by the training API and that will be information about the environment like the reward necessary to find a solution to the problem proposed. This method serves as a way for abstract implementations of RL algorithms to run the environments is a simple manner. The main functions of this class are the *init* function, the *reset* function, and the *step* function.

The *init* function, as its name says, is run when a new instance of the environment is created, and just for one time. In here, it is typical to declare constants and it is mandatory to define the state space and the action space as well. There is no return expected from this function.

The *reset* function serves as a way to restart an episode of the environment. In this function all of you variables have to be set back to their initial states and it the function has to return a series of initial observations that have to match the state space definition in the initialization function.

The *step* function is the most complex one. This function is responsible for advancing a step in time of the environment. That means that the next series of states have to be calculated in this function. The logic behind rewards also has to be programmed inside this function so that the rewards corresponding to the new states are calculated. The *step* has to return both the new states and the reward, but also a flag called *done* that is a boolean that indicates when an episode has finalized and that has to be set to true according to the logic behind the environment that is being created. Also, like for the *reset* function, the states return have to match the expected

format declared in the state space at the the *init* function. The step function will be accessed by the training API to send actions for the agents to execute, so the *step* function also expects the input of the action, and that has to be taken into account.

As the environment is defined as a Python class, the use of helper functions is completely available. That means that the tasks do not necessarily have to be programmed inside the previously mentioned functions, but the calls to the helper functions must happen inside those in order to return what is necessary from the environment.

The great thing about this approach is that anything that can be written inside those functions can be a valid environment and, thus, it can be used to play games and make different calls to external applications, making accessible to use RL solutions for a huge amount of problems.

RLlib has their own implementation of Gym environments, but for multi-agent RL. This type of environments can be created by inheriting from a Python class called *MultiAgentEnv* and the usage of it is really similar to the standard Gym environments but with the necessary changes to support the addition of several different agents. The changes are mainly due to the fact the the several agents are interacting on the same environment and so, for example, the *step* function has to be able to return more than one array of states, one for each agent. This extends to all the other functions and tasks and the process of doing this is pretty intuitive and well explained by the RLlib documentation.

The environments in RLLib go into a much deeper level with even more kinds that support many different things like specific types of multi-agent problems. However, it is not necessary to go into that much detail, and a further more in-depth explanation is always available in the documentation.

3.1.2 Algorithms

RLlib supports many different kinds of algorithms that are suitable for many different tasks, from very simple "hello world" applications, to state-of-the-art algorithms.

Among the many different algorithms we of course find some of the different ones discussed in previous sections. From value based models to policy gradient methods. One example being the popular PPO algorithm.

The different ways in which to customize an algorithm are endless. RLlib offers the

possibility of tweaking the different hyper parameters that are available for different algorithms. In the case of PPO, for example, we can specify how we want the model to be. This models are usually DL based, so NNs are used to approximate the policy function or the value function. These models are also fully customizable as it was mentioned before, the library is fully integrated with ML frameworks like TensorFlow that allow for this.

Specific algorithms are used by instantiating their trainer class and that will be discussed in the next section.

3.1.3 Training

RLlib provide a great training API that is simple and easy to use, but at the same time gives the possibility to access more low-level functionalities.

At a high level, RLlib offers a trainer class that is responsible for holding the policy for environment interaction. Through the trainer interface, the policy can be trained, saved, or an action can be computed.

For the sake of understanding a little bit better the way the trainer works, a brief explanation of it will be given. As always, the information available is much larger and more specific, but it is not necessary to know how the trainer operates as a user.

To start using the trainer, RLlib expects a series of configuration parameters and the environment that we want to train. The list of configuration parameters is pretty large and might be overwhelming at the beginning, however there are now well known parameters that can be found like the γ of the MDP, or the selection of the ML framework that is going to be used, like TensorFlow. These parameters will be better discussed when analysing the results of different experiments in a future chapter.

Once an instance of the trainer class is created, some of the functions that the trainer is capable of doing are *train*, *save*, or *compute_action*. The first one as its name suggest serves as a way to find the optimal policy, while *save* is for saving the parameters of the policy in order to restore it later. The *compute_action* function takes an observation of the environment as the input and returns an action to be sent to the *step* function of the environment in order according to the latest policy.

By using the *save* functionality, the checkpoint of the model is stored in a folder which also contains a file that can be read by TensorBoard. TensorBoard is a tool developed by the people from TensorFlow that provides with a visual platform to observe or debug the training and that is usually handy when dealing with ML algorithms. The figure 3.1 shows how the TensorBoard Application looks.

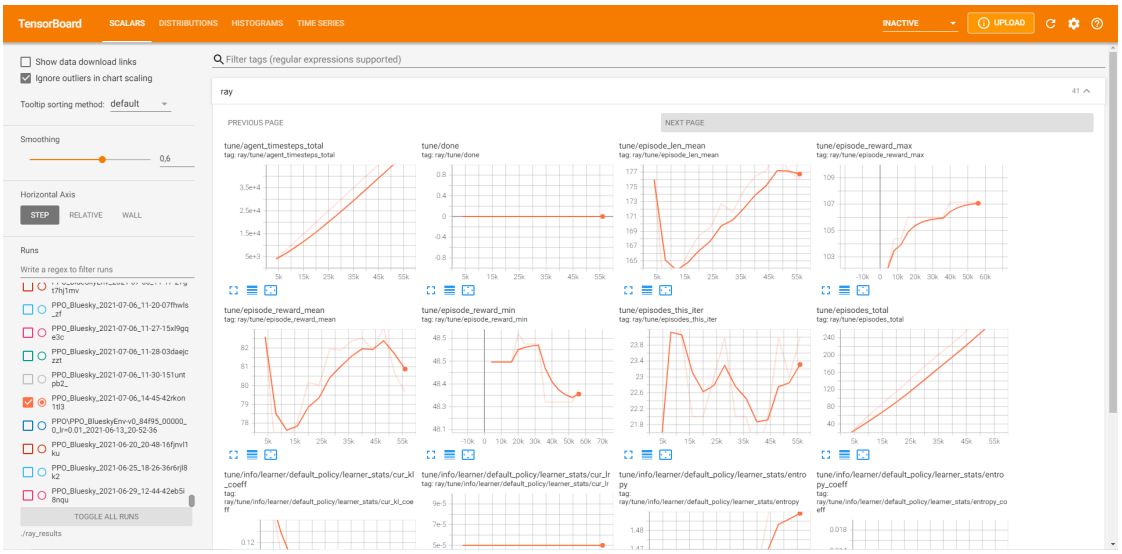

Figure 3.1. TensorBoard User Interface [11]

3.2 BlueSky simulator

BlueSky is an open-source air traffic simulator developed at TU Delft. "The goal of BlueSky is to provide everybody who wants to visualize, analyze or simulate air traffic with a tool to do so without any restrictions, licenses or limitations. It can be copied, modified, cited, etc. without any limitations" [12].

The simulator is written in Python, so it makes a great deal when trying to integrate it into the RL pipeline that is being built. It also contains data on navaids, performance data of aircraft or of airports. In addition to this, the simulator can perform simulations of aircraft performance, autopilot and even conflict detection and is compatible with the BADA 3.x files [31].

The characteristics presented above make of the simulator a great potential candidate for the application of RL, as the performance of the aircraft in the simulation will be close to the performance of a real one and thus provide with more realistic results.

BlueSky can run in many different manners. The most simple approach is to run it by default. By doing this the simulator opens a screen with a UI that can be use to visualize traffic and to input commands to create or modify many aspects that might be interesting like aircraft routes, sectors, waypoints, etc. This way of using BlueSky is really user friendly and for many applications might result as enough. However, BlueSky can also run as a server, without the UI, and can be controlled with Python scripts. A look at the UI is given in figure 3.2.

Figure 3.2. Bluesky User Interface [12]

If BlueSky is able to be controlled with Python scripts, the usage of Gym environments that can be in turn get used with RLlib, opens up the possibility of solving RL problems with this tool.

By using Python scripts, BlueSky has the option of controlling every aspect of the simulation. It allows for traffic creation, advancing the simulation step by step, defining routes, etc. All of this might be really overwhelming at the beginning because the source code for the simulator is pretty big. However, with enough dedication the philosophy behind its code can be understood and this way finding how to implement it to custom solutions becomes much easier.

It is worth mentioning that the simulator is well supported and the Github community [32], specially the developers from TU Delft, make a great effort responding doubts and fixing any kind of problem or bug that may arise.

3.3 Anaconda

Anyone that works with code, specially open-source like Python, might find than when building a program on top of different libraries that might be supported by different people, and those libraries might as well have dependencies, when an update that changes any of the libraries brings an incompatibility, the issue is carried down and has a cascade effect that can break a lot of code.

It is good practice using virtual environments. Virtual environments isolate a copy of a Python interpreter and its libraries making it possible to have multiple versions of the same libraries, and even of Python, on the same machine with the security that no version conflicts will be produced.

A great tool for the creating of virtual environments is Anaconda. "Anaconda is a package manager, an environment manager, a Python/R data science distribution and a collection of over 7500 open-source packages" [33]. Environment in this context is not to be confused with a RL environment, but a virtual environment.

Anaconda has a UI and a command prompt, both easy to use, and they allow activating or deactivating the environments or installing packages in a really simple way. For this project, given the high number of dependencies, not using a virtual environment is almost not a choice, and the usage of Anaconda will be of great utility.

3.4 Writing code

Code can be written in many different kinds of environments. For the project two different philosophies of writing code are used that benefit different kinds of the process.

3.4.1 Jupyter Notebooks

Notebooks execute code written in different "cells" that all run with the same Python Kernel. The use of a notebook permits a lot of experimentation with code as different functions or sets of code can be tested separately in order to tune it, specially when configuring the training API for the RL algorithms. The notebook environment used for the project is Jupyter Notebooks, which is open-source and offers easy installation and UI, and powerful environment, making it widely popular in the data science field.

3.4.2 Visual Studio Code

Integrated Development Environments (IDE) are solutions for creating code. They usually include a source code editor, an interpreter or build tools, and a debugger. The use of this kind of environment allows for the creation of a more definitive code that is not run cell after cell but in a command prompt. The IDE used for the project is Visual Studio Code, which is free and open-source and uses the python interpreter that is active in an Anaconda environment. The code created with this kind of environments is used in the project to run training in a command prompt, with a script that deals with everything in just one run.

Chapter 4

Experimentation

In the following pages, several problems are tried to be solved using RL techniques. All of them make use of fixed-winged Boeing 737s as the aircraft of choice. This is because the implementation of drones in BlueSky is not mature enough and, as the purpose of this experiments is just to test the different approaches, the fixed wing aircraft are also well suited.

4.1 Single agent

As an introduction to the field of RL, the use of complex environments is not recommended given all that has been discussed in previous chapters. When dealing with RL, in which many cases converging into a good policy can be really hard to achieve, having a complex environment that can add more difficulty for the algorithm to converge is a recipe for frustration. So for the sake of simplicity and better understanding the behaviour of the training process, a simpler environment is created that focuses on testing the pipeline created and not so much on the use cases of the algorithm.

The problem consists of a single aircraft that has to navigate to a random waypoint 15 nautical miles away from it. For this, a Gym environment has to be created that interacts with the BlueSky simulator.

4.1.1 Environment

For creating the environment it is helpful to first analyse the problem and how it can be implemented in a Gym environment. Specially two things have to be taken into account, which is the action space, and the state space.

The actions that are performed in the simulation are only heading changes. This is done to reduce both the observation and action space, subtracting complexity. It is decided that the actions should be in a continuous form, that means that the agent will be able to select for each step of the simulation, the new heading that the aircraft will want to follow from 0 to 360 degrees. This is in contrast to a set of discrete actions that could, for example, be turning left, right, or maintaining the heading.

The state space has to offer enough information for the agent to take decisions on what to do. For this problem the heading of the aircraft, the heading at which the waypoint is at, and the distance to the waypoint are the given states.

With the problem clear, the environment has to be constructed as a Python class that will inherit from the *env.gym* class and will have the methods or functions discussed before: *init*, *reset* and *step*.

4.1.1.1 Init function

In the *init* function, as previously discussed, the most important thing is to define the state space and the action space. Also, in this function, all things that have to be executed only one time when the simulation is created should be added to this function. In this environment the definition of the action and state space is the most concerning given the scope of the project, this means that the definition of constants other code related issues do not necessarily have to be mentioned.

The action space is defined as mentioned previously and, thus, the agent will be able to change the heading of the aircraft from 0 to 360 degrees. However, this wide range of values is not ideal for the convergence of the RL algorithm. To solve this issue, the action space is defined as action that can go from -1 to 1 and, then, this action will be converted to the values expected by the simulator before sending it the command to change the heading of the aircraft.

The state space is defined in a similar way, all the possible states will have to be included and bounded. This means that each state will have a minimum and

maximum possible value which will be have to be thought of in advance in order to not have the observations returned by the simulator be out of the state space. The heading is established as the first element of the action space, the desireed heading to reach the waypoint is the second element, the distance is the third element and it goes from 0 to a maximum range of about 40. This higher value is done to be always sure that even if the policy does not know how to reach the waypoint, the distance observation will not be higher that the one defined in the state space, which would cause an error. Also, like for the actions, all the states are normalized between the values of 0 and 1, which is easily done given the boundaries of the states are known.

4.1.1.2 Reset function

The *reset* function is called after every episode is terminated and it serves as the start of the next one. If it can be recalled, in the *reset* function the objective is to set the simulation in this case to its default values in order to have a starting point for the next episode. Also, the first set of observations has to be returned by this function.

In the *reset* function, the first thing that is done is check if any BlueSky server is active and then a connection is established if not already done so. Then, the aircraft and the waypoint have to be generated. The aircraft is spawned in a random location given by the boundaries established for the state space, over the center of the eastern coast of Spain. The coordinates of the waypoint that it has to follow are then calculated given the location of the aircraft. The waypoint is created 15 nautical miles away from the aircraft in any direction between the heading minus 50 degrees and the heading plus 50 degrees. This is done so that the environment does not always show the same situation which could lead to little generalization. Once the location of the aircraft and the waypoint are clear, both are created within the simulation. The aircraft is created with a flight speed of 250 knots and an altitude of 5000 meters.

After the simulator has received the command to create both the aircraft and the waypoint, another command is sent that returns the observations in the format that is given by the simulator. All these observations have to be converted in order to comply with the state space that was defined in the *init* function. For this the distance between the aircraft and the waypoint has to be calculated and then an array with all the observations has to be defined with each observation normalized between 0 and 1. The final step is to return the array of observations.

4.1.1.3 Step function

The *step* function is the most complex one as several tasks have to be performed in it, the first one being to receive the action and process it. The input of the step function is the action that has to be taken by the aircraft for that given step and this is given by the policy. If it can be recalled, the action space was defined a a range between -1 and 1, so the action that is received will be a number inside that range. This action than has to be converted to a value between 0 and 360 and it is sent to the simulator as a command.

The next task is to perform a step in the simulation, which is done by sending a certain command to the BlueSky simulator that takes care of all the necessary calculations.

Once the next step of the simulation has been calculated, the observations have to be obtained the same way as it was done in the *reset* function.

A really important part of the function is the calculation of the reward which will be calculated using the observations. In this environment, the reward is given by the distance between the aircraft and the waypoint. The reward will be maximum at the waypoint and will take the value of 1, decreasing linearly to 0 which is at the distance of 20 nautical miles. Another reward is received is correctly arriving to the destination waypoint, and has a value of 50. The sum of the reward of that step is then multiplied by a factor that goes from 0 to 1, being 1 when the heading is the right one to reach the waypoint, and 0 when it is completely the opposite. The final step to calculate the reward is to scale it so that it it not too high, this way it does not go over 1 and the algorithm can perform better, similar to what was done with the actions and states.

From last chapter, is has to be recalled there was a flag called "done" that also had to be returned by the environment and means the episode is over. For this environment this flag is set to true when the aircraft reaches its destination.

After performing all of these tasks, the normalized observation array, the reward, and the done flag are returned as these are the values that the training API will be expecting to collect, as explained in the previous chapter.

4.1.2 Training

For the training of the policy the algorithm PPO is chosen given the ease of implementation and how friendly it is for new learners of RL, as explained in previous sections. It also has the possibility of parallelizing several tasks of the training process which accelerate the training.

The training is done with RLlib using the *trainer* class discussed in the previous chapter and that takes as the input the environment and a set of configuration hyperparameters.

The selection of the configuration hyperparameters is sometimes hard to decide. No real instructions are given on how to perform this selection and the selection usually involves trying with the parameters used in a configuration from a problem that is similar and then tweak them until the desired solution is obtained. The selection of these hyperparameters is key to obtain the solution and the amount of trial and error needed to obtain them can be quite high and time consuming, as each training session can take several hours, or even days. Considering the amount of iteration in order to obtain these parameters, and that they are vital to obtain a valid solution, they are included in the results section.

4.2 Multi-agent

As the airspace is composed of more than one aircraft and, as it was mentioned at the beginning of the project, the final objective of using this technology is to be able to add a layer of automation to the airspace, the most attractive approach is to be able to use multi-agent RL to expand on the task presented in the last section and add a level of complexity. This is done by not only making the aircraft reach a specific waypoint, but also doing so while avoiding other traffic that they may encounter.

The approach to the problem has to take into account several factors including the way policies are going to be assigned to each agent, or the way rewards are going to be shaped. As before, these details are discussed in the following sections.

4.2.1 Environment

The environment that is used to solve this problem has to take into account the existence of other aircraft, which involves having to deal with more sets of actions, observations, rewards and "dones". The problem is considered as multi-agent because an attractive solution is to have the aircraft cooperate in order to avoid collisions. As it was explained in last chapter, RLlib offers a way to construct multi-agent environments in a similar way that Gym environments work. For this, the Python class from which the environment will have to inherit is named *MultiAgentEnv*.

For this problem, the environment, instead of spawning one single aircraft, will spawn two of them in a position where having a conflict can be likely. Also each individual aircraft will have a waypoint individually assigned that they will have to reach. Having said this, the way the environment is constructed goes as follows.

4.2.1.1 Init function

The *init* function is close to the one for the single agent, however the state space has to take into account the distance between the two aircraft as well. Apart from this, some constants may be added to ease the programming part, but they do not mean anything to the way the problem is approached.

4.2.1.2 Reset function

For the *reset* function, the objective will be to generate two aircraft, which will behave as two agents. The first one is created with the same approach as for the single agent, randomizing a location within a given boundary that is approximately over the center of the east coast of Spain. The way the second aircraft is added is by computing a location that is 10 nautical miles ahead in the direction of the heading but with some noise added in order to not always be perfectly aligned with the first aircraft. Before sending the command to the simulator to add both aircraft to the simulation, their heading are slightly adjusted randomly also to avoid having always a repetitive situation, this means that some trajectories will be non-collision paths and others will.

For the creation of the destination waypoints, a similar approach is used. The waypoint is determined by calculating the coordinates given a distance of 15 nautical miles in the direction of the heading and with added randomness, this has the same objective as before, providing a more diverse training environment. Each aircraft will have its waypoint calculated separately given their respective location.

Like for the other case, observations are then obtained from the simulator that will be adjusted to the shape that the state space has. This means that the distance between each aircraft and their respective waypoints will have to be calculated but also the distance between both of them. The result has to be two arrays of observations, one for each agent, where the coordinates of the agent and its waypoint are included as well as the heading, the distance to the waypoint, and the distance to the other agent. All of this will have to be normalized between values of 0 and 1 and arranged into a Python dictionary, in which the keys are the name of the agents, and each key contains the respective array of observations. This dictionary than is returned by the function like for the single agent case.

The dictionary is the way RLlib understands that a multi-agent environment is being used and is one of the additions that come from inheriting from the *MultiAgentEnv* class that RLlib provides.

4.2.1.3 Step function

The *step*, function has to calculate the next set of observations given the actions that are introduced as arguments into the function. The way actions are introduced are

by a dictionary just like the way observation were returned by the *reset* function. It has to be taken into account that the action space defined inside the *init* function will determine which input will be received in the dictionary, as the policy will follow that action space, the same way it happened for the single agent case.

As the actions in the action space go from -1 to 1, just like for the single agent environment, the action will have to be converted to the range of 0 to 360 that can be used to send the command to the simulator to change heading to that direction.

After having performed a step in the simulation, following the actions that came from the input of the function, the new observations have to be computed, which is done the same way as for the *reset function*.

Another important procedure is calculating the rewards and "dones". Which will follow the same structure that for observations and actions, being contained inside a dictionary in which the keys are the names of the agents. The "dones" dictionary is now a group of flags instead of a single one like for the single agent problem. Apart from a key for each agent, the "dones" dictionary will have to get added a new key that RLlib expects which is called "__all__". This done flag is set to true when the whole episode is done and it is mandatory to have it included.

The reward for this problem also gives a reward of 50 for reaching the waypoint and gives a value from 0 to 1 the closer the aircraft gets to the waypoint, similarly to what was done for the single agent. However, if crashes want to be avoided, the reward should reflect this. The way this is done is by penalizing the agents if they collide by a value of -50. Also, the done "__all__" key is set to true in order to stop the simulation and start a new episode.

Once the observations have been collected and processed, the rewards have been calculated, and the "dones" dictionary has been filled, these three dictionaries are returned by the *step* function just like for the single agent problem.

4.2.2 Training

This problem is also tackled using PPO, for the same benefits expressed before and the added one that also supports multi-agent problems in its RLlib implementation.

Having two agents leads to the question if it is necessary to have more than one policy. This might be the case if the agents are not of the same type. If, for a example, the simulation was a jet fighter trying to intercept another aircraft, there would be

a need to have two policies and train them in a different way in order to compete. As they are competing, the aircraft trying to not get caught will be rewarded by opposite motivations with respect to the jet fighter. However, the problem that the created environment tries to solve, as mentioned before, is cooperative and, as well, both aircraft are from the same type, meaning their performance should be the same. As it can be appreciated from this approach, the assumption can be made that the same policy is valid for both agents for the reason that they are wanted to behave the same way.

The training for this environment is, thus, performed the same way as for the single agent case as how the algorithm treats the received observations and rewards will not change depending on the agent they are coming from. The environment is introduced into the trainer class of RLlib that uses the PPO algorithm as well as several hyperparameters that have to be chosen and tweaked after each training in order to improve the results of the algorithm and that, as for the single agent case, will be discussed later.

Chapter 5

Results and conclusion

This chapter will discuss the results obtained in the previous experiments.

5.1 Single Agent

The single agent experiment offered some promising results as in most cases the aircraft reached the destination waypoint without a problem. Having said this, the amount of work to getting to a decent solution was really big and over 200 hours were spent training different approaches and hyperparameter configurations until the result provided with a good enough performance.

For this problem, the standard configuration given by RLlib for the PPO trainer was loaded and a couple of parameters had to be tuned. One of the parameters that was tuned has to do with the amount of parallel simulations that are run, and for the computing power that was available, 10 of them seemed appropriate and helps accelerate the learning process. Another one of the parameters is called *horizon* and it is the maximum number of steps carried out by an episode before forcing it to stop. The horizon parameter is vital as when the policy is not performing good enough to reach the waypoint the simulation would run for too long, until the *done* flag is activated, slowing the learning process. The final parameter was the *entropy*. The entropy is controlled the amount of actions that are carried out in a random way without following the policy. The goal of this parameter is pretty much controlling the rate of decay of the entropy, for this problem a value of 0.007 was used. The maximum amount that this parameters usually takes is 0.01 so the value was pretty

high. However, the use of this value helped the entropy decay more slowly and better exploration was obtained throughout the training.

Having said this, a picture of a trajectory being followed by the aircraft can be seen in figure 5.1. The big red X is the destination waypoint and the white triangle is the start of the trajectory. The representation over the map, containing visual references to different cities in the area, helps with obtaining a better perception of the trajectory.

Figure 5.1. Representation of the trajectory using Google Earth.

From the TensorBoard interface, the evolution of the mean reward and the entropy can be visualized as discussed in a previous chapter. The mean reward is the main performance parameter that has to be taken into account as, in the end, the objective of the training process is maximizing it. In the figure 5.2 the reward steadily rises until it stabilizes, which means the algorithm has found an optima point. The entropy in this case, seen in figure 5.3, is shown in order to visualize what was explained at the beginning o the section regarding the hyperparameter selection, and as it can be seen it tends to gradually decrease.

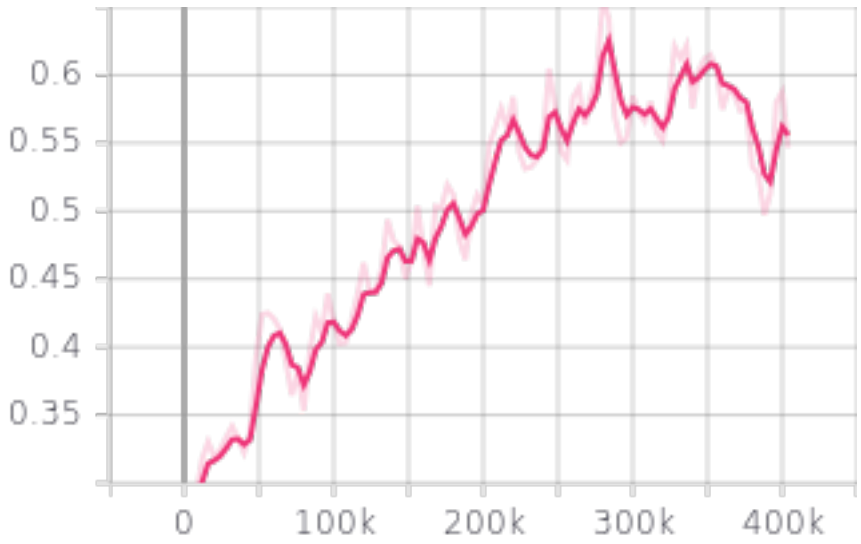

Figure 5.2. Single agent training mean reward evolution.

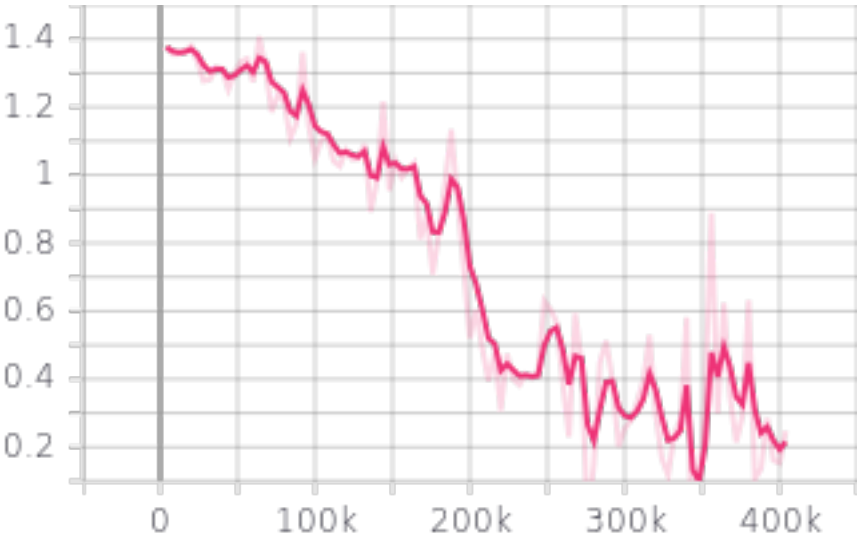

Figure 5.3. Single agent training entropy evolution.

5.2 Multi-agent

In order to analyse results, just like for the single agent case, some graphs from TensorBoard will be given to be able to observe the behavior of the training process prior to the discussion. The figure 5.4, figure 5.5 and figure 5.6 show the evolution of the mean reward, the maximum reward and the entropy of the training process.

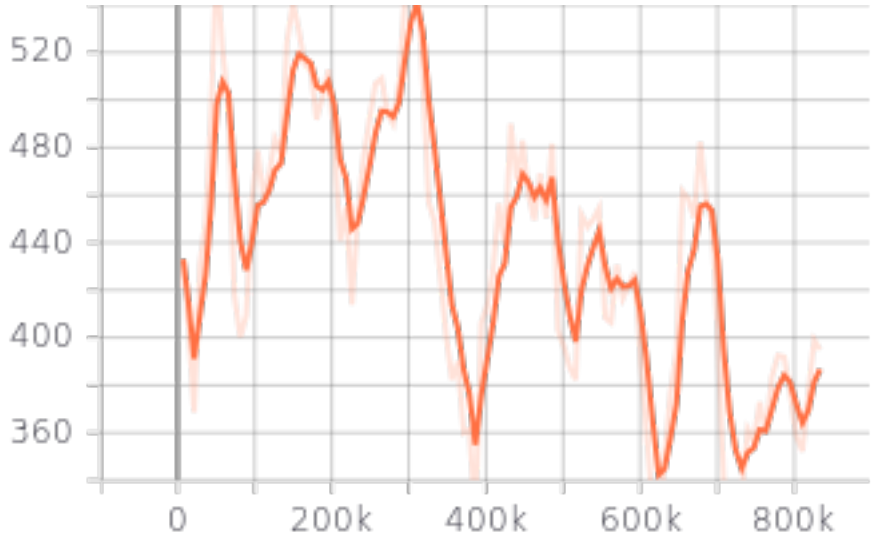

Figure 5.4. Single agent training mean reward evolution.

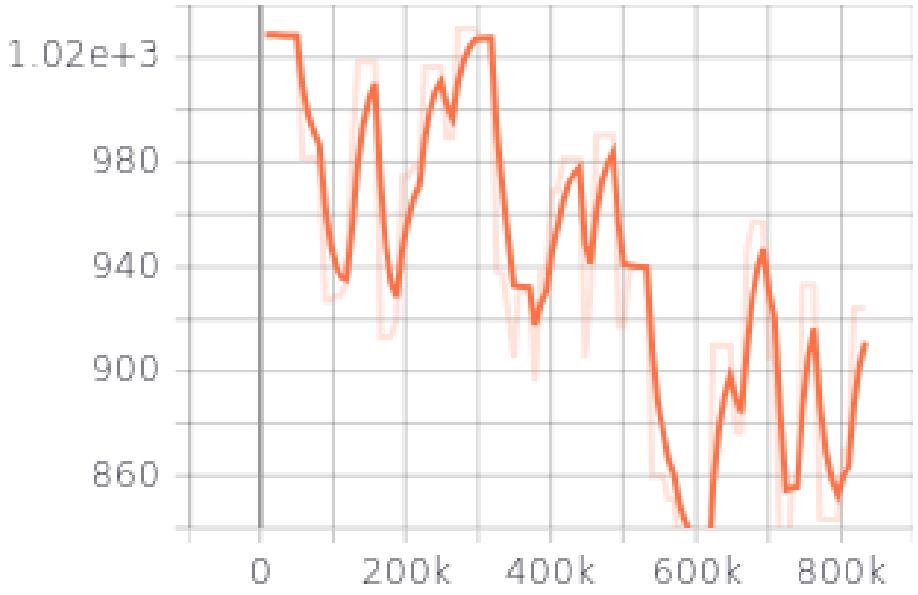

Figure 5.5. Single agent training maximum reward evolution.

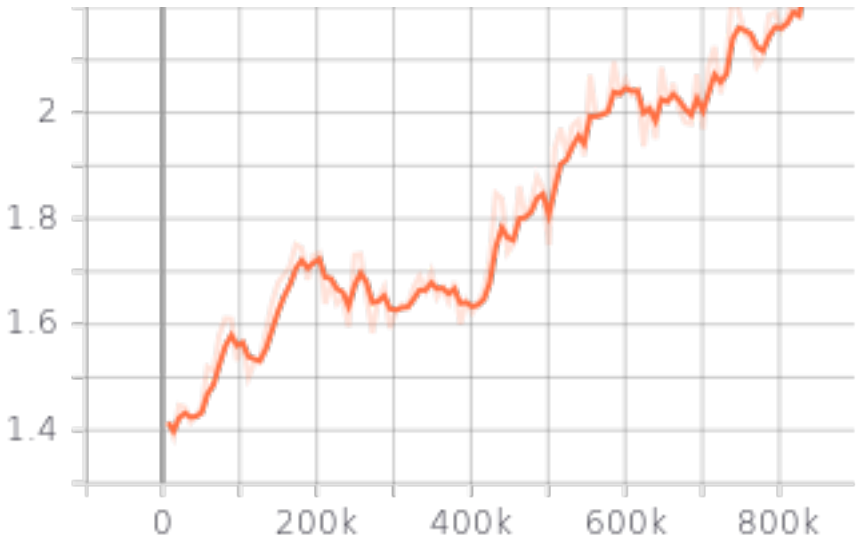

Figure 5.6. Multi-agent training entropy evolution.

We can observe from the graphs that the reward is not evolving, even it seems like the tendency for it is to go down. The entropy is also continuously rising. The results from the multi-agent environment could not be improved and a total of around 500 hours were used experimenting with different training approaches to no good result. For this reason, the above graphs only serve as an example of what it looks like when the solution is not arriving to the desired result.

Given the bad results, there is not a final choice of hyperparameters, as many of them were tried to no good outcome.

From the ideas and issues mentioned in past chapters, the fact that the multi-agent approach does not work should not come as a surprise given the difficulty of obtaining convergence. This does not mean, however, that the task cannot be done as in the thesis by Dennis van der Hoff's [16], for example, some of the multi-agent solutions seem to work properly.

The problem of convergence is not great news, but as well as some success was obtained in Dennis van der Hoff's thesis [16], many other tries discussed in the previous work section suffered the same fate.

5.3 Conclusion

The idea behind this project was learning and testing the methods discussed throughout the document with the ambition of obtaining results that would lead to a deeper interest and better implementation ideas for the technology. The results obtained, however, show the difficulty that resides on the construction of RL models, even for the simplest approaches, and all the limitations that they have been presenting up to date. In a field like Air Navigation, where safety is always the priority, the implementation of this ideas still seems like a long way into the future given the difficulty that would mean solving for an algorithm capable of dealing with much more states and actions than the ones used in the project, being that the case if a real scenario was trying to be approached. This, though, does not mean that the ideas exposed do not have an opportunity to be implemented as only the surface was scratched throughout the project.

This realization of this project has provided with a great insight into the field of RL. On top of it, a lot of new more in-depth knowledge can be built about new and better performing algorithms and methods that can be applied in order to solve more complicated problems in the Air Navigation field with more effectiveness. Some of those approaches could mean having more complex multi-agent architectures or working on better performing models, custom built for better adaptation to a specific problem. From this, even if the idea seems to still be far into the future, the possibility still remains to construct implementations that use these solutions and that could help dealing with the great variety of issues that are arising in the field of Air Navigation, like the exponential increase in drone traffic.

Another idea that can be extracted from the project is that the approach, if further research into the topic is done, has to be extremely experimental, which will result in failing many times in the process and dealing with issues that might not have an answer or have not been tried yet. The field of RL is constantly improving and expanding so, working in parallel to it developing solutions related to Air Navigation, could accelerate the introduction of this technology once its maturity makes it possible.

Chapter 6

Future Work

As is has been mentioned throughout the document, the objective of this project was to cover the basics of the implementation of AI, RL more specifically, to solve problems related to air navigation. That being said, the maturity of this technology is not yet developed and experimentation and research constantly lead to new great state-of-the-art algorithms that improve performance in an exponential way year after year. If working in parallel, research on this kind of applications could contribute to the development of these kind of algorithms and thus enter a feedback loop in which the world of RL and air navigation push each other to improve. This chapter introduces several interesting future ideas that could be attractive to work on.

6.1 Continuing With "The Basics"

Although this project tries to cover the basics, even these cannot be fully covered in such a small stretch of time and text. It is really important that steps are not skipped as it usually means having to come back to the basics all the time because the foundations were not correctly established in the first place.

The focus of doing more research could be related to understanding more different types of algorithms and working on implementing them with a more low-level approach that truly helps understanding the concepts and features of them. Not only on the RL side, but also experimenting with more kinds of NNs.

An approach that could be interesting regarding NNs is the use of *Long Short-Term Memory (LSTM)* neurons. What these kind of neurons do, with a high level

approach, is to store their inputs so that they can understand sequences of data rather than just an isolated input, which in the case of aircraft, given they are flying trajectories, seems like an interesting concept to introduce and might be useful as the function approximator for the policy or value-function.

Regarding RL algorithms, the use of methods like *Deep Deterministic Policy Gradients (DDPG)* which were not covered in this project could also be of interest and offer a different performance, also *Soft Actor-Critic (SAC)* could be interesting and the list just keeps going on.

6.2 Implementing Drones

The BlueSky simulator can simulate the performance of several different types of drones. However, the implementation of drones is still in very early stages. This will soon change as the team behind BlueSky is working on the development of a better suited environment for the simulation of this types of aircraft, in which city obstacles like buildings will be able to be implemented offering a much more interesting level of possibilities in the research field. This extension will mean that urban delivery traffic could be simulated in a much more realistic way. Also, a great use case for this extension is the study of the surroundings of airports, in which many conflicts can arise and specially if drone traffic is expected to continue rising.

As mentioned in the introduction, one of the attractive ideas surrounding the development of this kind of AI techniques is the ability to use them in the world of unmanned aircraft, so it is naturally something that has yet to be implemented and worked on in the future, and the use of the BlueSky simulator might still be a great way of simulating precisely all the needed scenarios.

6.3 The Problem of High Computing Requirements

It has been mentioned several times that one of the main obstacles RL has to overcome is the necessity for high computing resources. This would be drastically amplified if in the discussion the topic of drones is added, meaning that if the problem was still complex, this would only make things worse. In the field of manned aviation, this would be different, but as it was discussed at the introduction of the report and at least for now, this technology is only appealing to the world of unmanned aircraft

given the high security and conservative standards imposed by manned aviation.

What is clear is that the idea of having on-board systems that process the algorithms is something unreal on small unmanned vehicles like delivery drones, which is the vast majority of them. It is unreal in all possible ways due to the fact that it would be ridiculously expensive and heavy, and yet not powerful enough. But, what is a possible workaround that can be made to overcome this situation?. A possible promising solution will be discussed in the next section, as this problem is the most concerning for the deployment of the technology.

6.4 Deployment

It might be too soon to star the discussion on this topic as many other different things have to be done before considering the deployment of the technology. However, no technology is useful if the deployment of it does not bring it down in a useful way to the user. As mentioned before, the deployment on big manned or unmanned aircraft is not such a big problem, but the fact that most big aircraft are manned and as discussed, for now, it cannot be considered to used this technology in that field, the discussion will focus on the deployment on drones.

The previous section mentions the problem that has to be faced in order to achieve the introduction of this technology into UAVs, specially the smaller ones. A solution that might not yet be possible to be implemented, but that might soon be a reality, is the use of 5G, or at least on the urban space. The use of this technology is ideal in the urban environment as the low latency given by 5G can allow for the usage of edge computing as a centralized service. This way, the drones do not necessarily have to compute their own actions on-board, but they could request the realization of the computation to a bigger server that can process the states and provide actions much quicker and, thus, the overall time would be greatly reduced. However this approach, of course, also can bring a lot of problems, specially if the network is of low quality, as that would mean that the service would go out of order. This would probably be the case in more remote areas, but more workarounds might be able to be found, as these areas will specially have a very low density traffic.

The topics introduced in this chapter allow for deep analyses, discussions and are all worthy of a a lot of research. However, and sadly, not all topics can be covered in this project. It is sure that the future of air navigation promises great advancements

and interesting approaches that will bring safe, efficient, and cheap solutions to all kinds of flights, making of this field a great one to work on.

Chapter 7

Budget

In the chapter the budget required for the realization of the project is discussed. It will also include a final price broke down in different components.

7.1 Salaries

The first thing to determine is the cost per hour worked. The student was under an apprenticeship and the salary is 6.75 €/h. The estimated salary for the tutor is around 30 €/h.

The project had a duration of around 5 months and the total amount of hours worked is estimated at an average of 10 hours per week for the first 3 months, and 6 hours per day for the last 2. This makes the amount of hours dedicated by the student to be:

$$\begin{aligned} 10\ \frac{hours}{week}\ x\ 4\ \frac{weeks}{month}\ x\ 3\ months &= 80\ h \\ 6\ \frac{hours}{day}\ x\ 5\ \frac{days}{week}\ x\ 4\ \frac{weeks}{month}\ x\ 3\ months &= 360\ h \\ Total\ hours &= 440\ h \end{aligned} \tag{56}$$

The hours dedicated by the tutor are estimated as 20 and, thus, the salary costs result in what is shown at table 7.1.

Concept	Quantity (h)	Price per unit (€)	Total price (€)
Salary expense of tutor	20	30	600
Salary expense of student	440	6.45	3574
Total salaries			3574

Table 7.1. Salaries

7.2 Equipment and Software

The cost of equipment is going to take into account the amortization of the equipment used and the software licenses.

The cost of the equipment used taking into the account the amortization can be seen in table 7.2.

Equipment	Amortization period	Total cost (€)	Utilization period	Cost in the project (€)
ASUS ROG Strix G712LV	24 months	1499	5 months	313

Table 7.2. Equipment cost

The software used in thee project is all open-source, making the cost 0. The software has been mentioned in a previous chapter.

7.3 Indirect costs

Indirect costs are the ones that do not depend directly on the project, but the fact that the project is done originates this costs. Indirect costs are the ones related to the job done by the administration of the school, power consumption and internet. These cost are hard to calculate due to the complexity and variety of them so it is approximated to 5% of the total budget of the project. Indirect costs are shown in table 7.3

Concept	Total	Percentage (€)	Total price (€)
Indirect Cost	3887	5%	195

Table 7.3. Indirect costs

The total expenses originated by the project divided by category and the total cost is shown in table 7.4.

Concept	Total (€)
Salary	3574
Equipment	313
Indirect	195
Final	4082

Table 7.4. Final budget

Bibliography

[1] M. Oduncuoglu and H. I. Kurt, *The basic structure of an artificial neuron*, May 2015. [Online]. Available: https://www.researchgate.net/figure/The-basic-structure-of-an-artificial-neuron-43_fig3_282849515

[2] Totorialspoint. A typical ann. [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/images/atypical_ann.jpg

[3] OpenNN. Deep neural network. [Online]. Available: https://www.neuraldesigner.com/images/deep_neural_network_big.png

[4] ——. Hyperbolic tangent. [Online]. Available: https://www.neuraldesigner.com/images/hyperbolic_tangent.png

[5] ——. Logistic activation. [Online]. Available: https://www.neuraldesigner.com/images/logistic.png

[6] ——. rectified linear activation. [Online]. Available: https://www.neuraldesigner.com/images/rectified-linear-activation.png

[7] ——. Performace function. [Online]. Available: https://www.neuraldesigner.com/images/performance_function.svg

[8] S. Bhatt. Generic reinforcement learning loop. [Online]. Available: https://miro.medium.com/max/700/1*7cuAqjQ97x1H_sBIeAVVZg.png

[9] Atari. Atari breakout (2600) screenshot. [Online]. Available: https://www.mobygames.com/images/shots/l/617748-atari-80-classic-games-in-one-xbox-screenshot-breakout-2600.jpg

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Clip parameter ilustration. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[12] J. Hoekstra and J. Ellerbroek, "BlueSky ATC Simulator Project: an Open Data and Open Source Approach," June 2016. [Online]. Available: https://www.researchgate.net/publication/304490055_BlueSky_ATC_Simulator_Project_an_Open_Data_and_Open_Source_Approach

[13] M. Brittain and P. Wei, "Autonomous air traffic controller: A deep multi-agent reinforcement learning approach," *CoRR*, vol. abs/1905.01303, 2019. [Online]. Available: http://arxiv.org/abs/1905.01303

[14] M. Ribeiro, J. Hoekstra, and J. Ellerbroek, "Determining Optimal Conflict Avoidance Manoeuvres At High Densities With Reinforcement Learning," December 2020. [Online]. Available: https://www.researchgate.net/publication/346647401_Determining_Optimal_Conflict_Avoidance_Manoeuvres_At_High_Densities_With_Reinforcement_Learning

[15] B. Vonk, "Exploring reinforcement learning methods for autonomous sequencing and spacing of aircraft," April 2019. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:2e776b60-cd4e-4268-93e3-3fcc81cd794f?collection=education

[16] D. van der Hoff, "A Multi-Agent Reinforcement Learning Approach to Air Traffic Control," June 2020. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:4d02e51a-0187-404d-b465-7ae01feba8e8?collection=education

[17] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. October, pp. 433–60, 1950.

[18] S. J. Russel and P. Norvig, *Artificial intelligence: a modern approach*. Upper Saddle River, New Jersey: Prentice Hall, 2010.

[19] What is Artificial Intelligence? How Does AI Work? [Online]. Available: https://builtin.com/artificial-intelligence

[20] T. W. Malone, D. Rus, and R. Laubacher, "Artificial Intelligence and The Future of Work," December 2020. [Online]. Available: https://workofthefuture.mit.edu/wp-content/uploads/2020/12/2020-Research-Brief-Malone-Rus-Laubacher2.pdf

[21] S. Brown. Machine Learning, explained. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[22] OpenNN. Neural network tutorial. [Online]. Available: https://www.neuraldesigner.com/learning/tutorials/neural-network

[23] ——. Training strategy tutorial. [Online]. Available: https://www.neuraldesigner.com/learning/tutorials/training-strategy

[24] S. Bhatt, "Reinforcement learning 101," Mar. 19, 2018. [Online]. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[25] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Complete Draft)*. Cambridge, Massachusetts London, England: The MIT Press, 2017.

[26] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[28] A. Inc. Ray docuementation. [Online]. Available: https://docs.ray.io/

[29] ——. Understanding the ray ecosystem and community. [Online]. Available: https://www.anyscale.com/blog/understanding-the-ray-ecosystem-and-community

[30] O. AI. Gym documentatio. [Online]. Available: https://gym.openai.com/

[31] A. Nuic, D. Poles, and V. Mouillet, "BADA: An advanced aircraft performance model for present and future ATM systems," March 2010. [Online]. Available: https://www.eurocontrol.int/sites/default/files/2019-03/overview-bada-apm.pdf

[32] TUDelft-CNS-ATM. Bluesky github. [Online]. Available: https://github.com/TUDelft-CNS-ATM/bluesky

[33] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/