UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

AutoTrading with Reinforcement Learning

Autor: Johnny Núñez Cano

Director:	**Dr. Eloi Puertas i Prats**
Realitzat a:	**Departament de Matemàtiques i Informàtica**
Barcelona,	**19 de juny de 2021**

Abstract

Trading is the act of studying any financial market and making money with it through buying and selling of assets. In this project, I will try to automate the actions performed by a trader without having a thorough knowledge of the financial market or trading techniques. I will use algorithms based on reinforcement learning techniques used in other fields such as robotics without human interaction in the algorithm's execution. The main objective of this project is to investigate the feasibility of using these techniques adapted to Deep Learning and the ability to cope with the volatility of cryptocurrency. Furthermore, to show the results of these algorithms, the cryptocurrency Bitcoin and ADA will be used as a market study by obtaining the historical and making market analysis.

Resum

El trading és l'acte d'estudiar qualsevol mercat financer i guanyar diners a través de la compravenda d'actius. En aquest projecte tractaré d'automatitzar les accions que realitza un trader sense tenir un coneixement exhaustiu del mercat financer o de les tècniques de trading. Utilitzaré algorismes basats en tècniques d'aprenentatge per reforç utilitzades en altres camps com la robòtica sense tenir interacció humana en l'execució d'aquest algorisme. L'objectiu principal d'aquest projecte és investigar la viabilitat de l'ús d'aquestes tècniques adaptades al Deep Learning i la capacitat de fer front a la volatilitat de les criptodivises. Per a mostrar els resultats d'aquests algorismes, s'utilitzarà la criptomoneda Bitcoin i ADA com a estudi de mercat obtenint l'històric i fent anàlisi de mercat.

Resumen

El trading es el acto de estudiar cualquier mercado financiero y ganar dinero con ello a través de la compraventa de activos. En este proyecto trataré de automatizar las acciones que realiza un trader sin tener un conocimiento exhaustivo del mercado financiero o de las técnicas de trading. Utilizaré algoritmos basados en técnicas de aprendizaje por refuerzo utilizadas en otros campos como la robótica sin tener interacción humana en la ejecución de dicho algoritmo. El objetivo principal de este proyecto es investigar la viabilidad del uso de estas técnicas adaptadas al Deep Learning y la capacidad de hacer frente a la volatilidad de las criptomonedas. Para mostrar los resultados de estos algoritmos, se utilizará la criptomoneda Bitcoin y ADA como estudio de mercado obteniendo el histórico y su análisis de mercado.

Acknowledgements

First of all, I would like to express my gratitude to Dr. Eloi Puertas. He has been a great help throughout the university career, both as a Director of studies and with this project. We have shared our passion for technology in conversations about distributed systems, cryptocurrencies, or robotics. Thanks to his excellent listening skills, we have shared ideas to improve this project and the academic field with his constant opinion and feedback about any technological and personal area.

I also thank Dr. Sergio Escalera and Cristina Palmero for allowing me to get an insight into the excellent work of their research team involved in performing a challenge on computer vision.

appreciate the help of of the professors at the University of Barcelona who have provided me with the training. I can't thank Dr Juan Gabriel Gomila (Professor of Mathematics of the Balearic Islands University) enough for introducing me to the world of Artificial Intelligence. Together with him, I now help students from around the world to acquire this branch of computer science.

Finally, I want to thank my family, mainly my parents, for their effort, love, and patience for allowing me to recover my health and study for a university career which means the world to me.

Contents

Introduction

I want to start this paper by saying that, in my personal life, I consider myself a technology enthusiast. I am constantly reading, informing myself, and contrasting all kinds of information related to the world of computing. I am writing this paper because I am one of those people who have been following Bitcoin since its inception when Bitcoin was worth pennies 12 years ago and when I was just 12 at a time. I am one of those people who did not get rich and lost money because I did not understand the potential of this technology and the new money 2.0. I really learned what the Blockchain was, the problems of today's capital, and being underaged as well as the access to these cryptocurrencies around 2014. They are just a representation of the great potential that the Blockchain has. However, I have never learned to trade at a professional level. I consider it as being a computer scientist, a philosophy of life that you have to dedicate to day by day in order to improve. Although I still do not know all about trading, I do have basic knowledge of chart analysis techniques, and I have studied Blockchain technology more thoroughly. So I came to a conclusion to make it the main goal of this project. Hence, with a more solid foundation of computer science and the world of programming I ask myself the following question: Is it possible to automate transactions without actually spending years of money and time? I will try to answer it in this project. In addition, a topic that is not often considered is gambling. I will try to prove that making money is not easy, we have to be responsible for the actions taken by the entire market and control our fears, so that we can see the risks the robots take when making decisions.

The project's primary goal is to introduce artificial intelligence technology, especially the reinforcement learning branch used in other fields such as robotics or video games. Nevertheless, here we will present it in the financial area to create an automatic robot, called a trading robot, which knows how to process the buying and selling of not needed cryptocurrencies in an intelligent and automated way.

A trader carries out a study of the market, the product or technology, and his own survey of graphical analysis, trying to find patterns or trends that allow him to see and predict the market's direction. Each trader has his own tactics or policy regarding decision-making, which means that the trader's judgment, experience, and psychology play a significant role. A trader requires many years of experience, training, and above all losses to become an expert and make a living from trading. This process, which is done manually, requires a substantial investment of time, dedication, and effort that might need years. However, the solution I am looking for in this project would allow bots to learn in

a matter of hours and make decisions quickly and intelligently without an essential factor that tends to cause us humans make mistakes, with our feelings and emotions.

Bots have always been present in many areas of computing, for example, Alexa or Siri in language recognition, simple bots created by decision trees on websites or events, or even NPCs in video games, but relatively few bots in finance have been seen publicly. It is well known though that even Wall Street has automated its processes because nowadays everything is handled by computers. Reinforcement learning and neural networks can help to create a bot that can find a policy for different trading situations, reducing both training time and the possibility of reacting quickly to a change in market trends without prior knowledge.

My knowledge acquired throughout my university career, the Treball de Fi de Grau (TFG), emphasizes what was taught in the subjects of "Machine Learning," where the study of data analysis and the main architectures of Deep Learning are applied together with the subject of "Intel·ligència Artificial" and "Robòtica" where the philosophy of reinforcement learning is taught and studied.

1.1 Motivation and Objectives

Artificial intelligence currently emphasizes the use of Deep Learning, where it has been seen the excellent projection at the hardware and software level in recent years in computer vision, natural language processing and robotics. Therefore, Reinforcement Learning is considered state of the art in many branches of research.

This project aims to investigate the consequences obtained by applying Deep learning in such a volatile market as the world of cryptocurrencies, studying the different algorithms and implementations of neural networks existing in other fields of computer science, and being able to perform a technical market analysis. There is a particular interest in learning the technology behind a cryptocurrency, especially in the bitcoin, where it is a revolutionary technology, thanks to mathematics and computer science which make a specific interest in a future digitized world.

The applied algorithm should be faithful to the trend line and relatively fast. It should be able to make a decision in a matter of milliseconds as soon as the algorithm's input is formed. However, all the implementations seen so far have been carried out in regular markets such as the stock market, so the algorithm will have to learn in an even more volatile market. Therefore, a study of indicators will be carried out to help the neural network to create a more significant number of states in order to be able to make better decisions.

All the results obtained by the algorithms will be compared and analyzed. This will allow us to see the strengths and weaknesses of the algorithm itself. The final objective will be to see the best algorithm executed in real-time and see its results and also to see if it is possible to trade.

1.2 Memory Organization

The structure implemented in this project aims to increase the complexity of the concepts. The explanation will be based on related works, helping to complement the description of the concepts in a visual way, then the structure and design of the implementations and the different tests carried out will be explained. It is organized as follows:

- **State of the art:** All work requires previous research. This section will show the story of how reinforcement learning came about, how the idea of Deep Learning came about, and the most recent related work on such techniques that encourages research to find the most significant benefit to the problem of this project.
- **Data Analysis:** Knowing how to get the data, what each piece of data means so that we can understand and implement the different solutions throughout the project.
- **Methods:** This point one of the most important ones. It will be explained theoretically what each method does and the advantages and disadvantages of taking that methodology at a theoretical level.
- **Design of Experiments:** The different methods described in the previous section will be designed to allow us to choose the best model to solve the problem. Then, the implemented architectures will be tested, and we will try to improve the implemented architectures.
- **Analysis of Results:** The results obtained will be discussed. The different architectures implemented and the reason why such a result can be obtained will be compared, concluding the possible deficits that that architecture may receive.
- **Conclusions:** The analysis whether the proposed objectives have been met or not will be presented. There will also be some discussions on how to improve current work and some future ideas.

State of the Art

This section summarizes some existing and just released methods that also address the trading bot problem. They will be mentioned and briefly explained to put this project into perspective and better contextualize it.

2.1 Background

In this section, before state of the art related to trading bots, I will explain reinforcement learning concepts and the different parts that make up a neural network.

2.1.1 Reinforcement Learning

Reinforcement learning[1] is a branch of Artificial Intelligence. The goal of an agent is to maximize its benefits and reduce its losses in the most effective way possible by observing its environment and performing specific actions resulting from rewards and punishments.

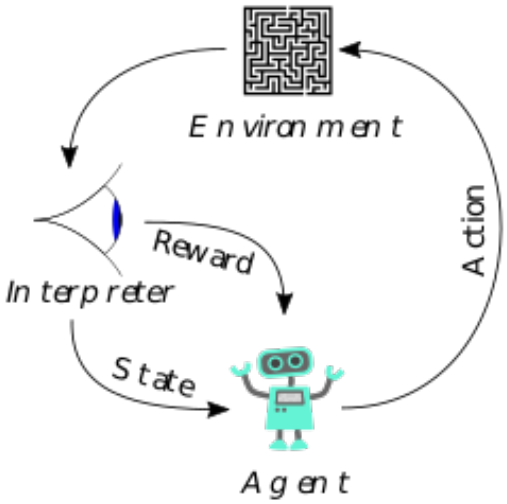

Figure 2.1: Reinforcement Learning Diagram. Image by Wikipedia

The following definitions are important throughout the paper:

- Agent: the program that is trained for performing a pre-specified task.
- Environment: the ecosystem in which the agent performs the tasks, whether real or virtual.
- Action: movements made by the agent, which have an impact on the ecosystem.
- States: The environment offers possible situations where the agent can take action, and this changes each time the agent takes action.
- Reward: the evaluation of an action, which can be positive or negative.
- Discount: Solves the problem that the agent does not know which path to take because it has a dichotomy in decision-making equality.
- Penalty: is a value used so that our agent remains indefinitely without taking action; when he has comparable actions, it is a way of forcing him to solve the problem.

The agent should not choose a plan to solve his problem, because the definition of a plan is to predict something without changing its behavior. Then it changes in the environment may occur that cause the agent to have to change the action along its sequence of actions. action. Hence, in reinforcement learning we talk about policy, the agent should choose a policy that allows him to solve the problem posed so that the agent can solve the problem with some randomness as well as if the environment changes.

The action that the agent chooses at each moment should not only depend on the reward that it is going to receive in the short term, but it should also select the actions that in the long term will bring it the maximum possible gain in the whole episode (all the states that are between an initial state and a terminal state). To solve all these hypotheses, the Bellman-Ford equation was born.

Bellman-Ford Equation

$$V(s) = \max_a(R(s,a) + \gamma V(s'))$$

where:

$V =$ Value
$R =$ Reward
$s =$ state
$s' =$ future state
$a =$ action
$\gamma =$ discount

The Bellman equation came to solve one of the paradigms of programming, such as dynamic programming[2]. This equation allows us to obtain the value of the state maintained by the agent. Therefore, it produces an estimate of the reward that the agent will bring until the end of the plot, starting from the state s, which means that all acting agents are seeking what improves the value or situation regarding the current state.

The gamma is to solve the problem when the agent does not know which path to take because he has a dichotomy in decision-making equality. Thus, he would have the same value in all his actions.

The problem with this equation is that you get a plan, so it is a deterministic solution, it would always take the same actions.

To solve this problem, probabilities come into play. The deterministic search ensures that the same action is always executed 100% of the time. With probabilities, we obtain a non-deterministic search; for example, if an agent must take Left, Right, Up and Down if we add a certain probability for taking each of these actions, we would add randomness. Example: Left: 15%, Right:15%, Up:20%, Down:50%. To improve Bellman's equation, Markov processes appear.

Markov Processes

It is a time-dependent random/stochastic phenomenon that enforces the Markov property. The Markov property[3] refers to the property that specific random or stochastic processes are memory-less, which means that the probability distribution of the future value of the random variable only depends on the value in the present and is independent of everything that has happened in the past.

So all processes that fulfill this property are called Markov processes, i.e., a Markov process is a process where at each time step, the conditional probability distribution of the past times is independent.

This means that a previous action will not condition the taking of the current condition, so it is independent. So the decision we make is random or partially under the control of the agent's decision-maker to avoid the agent having 100% randomness.

The V becomes the weight of the possible decisions that the agent can make.

$$V(s) = \max_a(R(s,a) + \gamma \cdot \sum_{s'} P(s,a,s') \cdot V(s'))$$

where:

V = Value
R = Reward
s = state
s' = future state
a = action
γ = discount
P = Probability

With this equation, we solve the deterministic search problem, and now we will obtain a policy to solve the problem.

One of the most famous policies is the ϵ-greedy policy, where the agent will almost always take the best possible action given the information it has. Still, occasionally, with a ϵ, the agent will take a completely random action. This ϵ value is decided at the outset before executing the agent and will be the way we balance the exploration and exploitation problem. Exploiting allows us to maximize the rewards with the set of actions we already

know. In contrast, exploration will enable us to investigate all possible actions to see if we can get a better group of actions to maximize the gain further.

Into this new equation comes the penalty factor. It is a factor where the agent is forced to make decisions and not stay indefinitely in a state, so in this case, we dedicate negative intermediate rewards for the agent to risk-taking actions.

Q-Learning Equation

$$Q(s,a) = R(s,a) + \gamma \cdot \sum_{s'} P(s,a,s') \cdot V(s')$$

where:

Q = value function
V = Value
R = Reward
s = state
s' = future state
a = action
γ = discount
P = Probability

A new variable, Q, appears that measures the quality of the action from the state[4]. In contrast, the value only measures the quality of being in that state, so the Q-Learning algorithm tries to learn how much reward it will get in the long run for each pair of states and actions (s, a). We call that function the action-value function represented as Q(s, a), which returns the reward that the agent will receive when executing action a from state s, and assuming it will follow the same policy dictated by the Q function until the end of the episode, so it tries to maximize the quality of the moves. From there, it looks at what action to take. So he thinks like this: what's the best action he can handle? He compares them. Once it has reached them, it sticks with the one that gives it the maximum quality of movement and moves on, and so on until it finds the optimal solution.

$$Q(s,a) = R(s,a) + \gamma \cdot \sum_{s'} P(s,a,s') \cdot \max_{a'} Q(s',a'))$$

$$V(s) = \max_{a} Q(s,a))$$

where:

Q = value function
V = Value
R = Reward
s = state
s' = future state
a = action
a' = future action
γ = discount
P = Probability

Temporal Difference

Now we add the time difference[5], which is only the value of Q being modified over time. So it measures the increase in information.

$$TD(a,s) = (R(s,a) + \gamma + \max_{a'} Q(s',a')) - Q(s,a)$$

where:

TD = Temporal Difference
Q = value function
V = Value
R = Reward
s = state
s' = future state
a = action
a' = future action
γ = discount
P = Probability

A new variable is added, called Alpha, which allows, for example, to discard the previous information or discard it but make a hybrid to ensure that the new value does not substitute the last value.

$$Q(s,a) = Q(s,a) + \alpha \cdot TD(a,s)$$

where:

TD = Temporal Difference
Q = value function
s = state
a = action
γ = discount
α = learning rate

2.1.2 Deep Learning

The Neuron

Deep learning emerged in the 1970s, thanks to Alexey Ivakhnenko[6] who wrote the first paper on neural networks. In 1989, Yann Lecunn wrote the first paper on Efficient Backpropagation,[7] and in 2006, Geoffrey Hinton, among other researchers, programmed the first Deep learning model[8][9]. The main idea is to try to translate some of the neuroscience that governs the laws of the human brain into a computer. However, there are still many unknowns about what actually happens in the human brain from the neuroscience side.

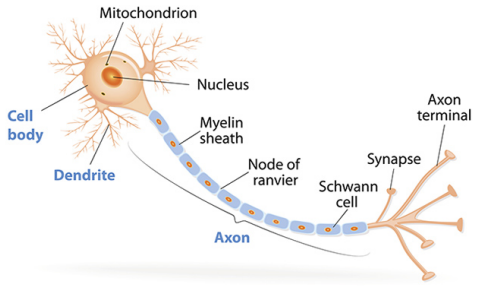

Figure 2.2: Neuron Representation. Image by The University of Queensland

In the neuron in Figure 2.2, one single neuron will receive information from a wide number of neurons through its dendrites. Further on, the synapses will decide whether that input will stimulate or inhibit the neuron activity. For each pair of dendrite and synapse, the result will be multiplied and summed with the others. If the neuron gets activated because the signal has reached a high enough value, it will send a signal through its axon. Otherwise, it will remain silent and act as a switch.

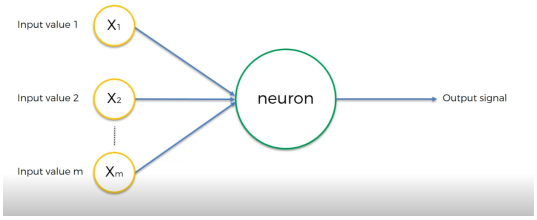

Figure 2.3: Neuron Representation. Image by Juan Gabriel Gomila

As we can see, each neuron has an input that would be our normalized data; in that neuron, the products of each weight are added by the respective input value, where all the weighted sum is summarized in a single number, as simple as multiplying and counting, for each input. Then an activation function is applied, if the neuron is activated or not. If it is activated, the information is transmitted to the next layer.

The Activation Function

There are many activation functions [10], which allow us to activate a neural network[11]. The most common ones are:

- Threshold function: if the input value is negative, it converts it to 0, but if the input value is positive, it converts it to 1. It is like a True or False.

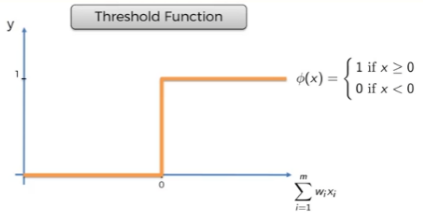

Figure 2.4: Threshold Function. Image by Juan Gabriel Gomila

- Sigmoid: Basically, what it does is to say how likely it is that the neuron is activated, so we are talking about probabilities.

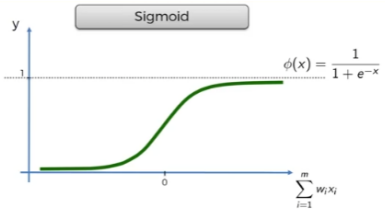

Figure 2.5: Sigmoid Function. Image by Juan Gabriel Gomila

- Rectifier(ReLu). Transforms everything negative to 0, and from there, everything positive stays the same.

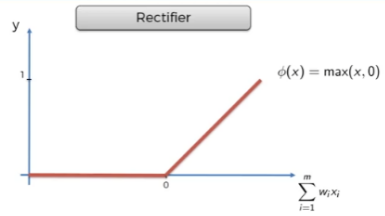

Figure 2.6: ReLU Function. Image by Juan Gabriel Gomila

- Hyperbolic tangent. Similar to the sigmoid function, only that it starts in negative. It is used when we need negative values.

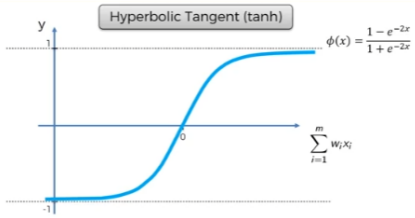

Figure 2.7: Hyperbolic Tangent Function. Image by Juan Gabriel Gomila

How do Neural Networks work?

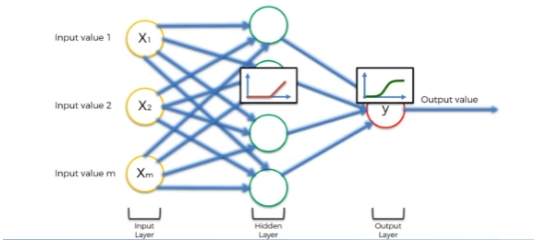

Figure 2.8: A Simple Neuronal Network. Image by Juan Gabriel Gomila

As we can see in the picture, neural networks, if they were only with one output layer, could only propagate the weighted result of the input. So, where is the magic in deep learning[12]? Layers of neurons are added. The more layers, the deeper the neural network. Within those layers, the neurons are activated through the interconnections between them. For example, in layer 1, the first neuron is activated by neurons 1 and 3 of the input layer, and so it happens consecutively for each neuron. So we could say, each neuron interprets information in its way. Separately, each neuron would be powerless, but a set of neurons makes the system work and is able to predict as a human brain works through millions of neurons.

How do Neural Networks learn?

This is the question that everyone asks when they see something related to deep learning. Neural networks learn thanks to an error function that is provided in the output, i.e., a neural network usually separates the output value with the actual current value to adjust the weights of the neural network. Neural networks, through their neurons and connections predict a value. This value does not have to resemble the actual current value, so we must adjust that error difference with the actual value, so the neural network has to correct how far are we from the actual value. This value can be quantified and updated through a function called a cost function, which is defined as half the squared difference between the actual value and the predicted value. The goal is to minimize that error, so we forward the amount of error back to the neural network through backpropagation.

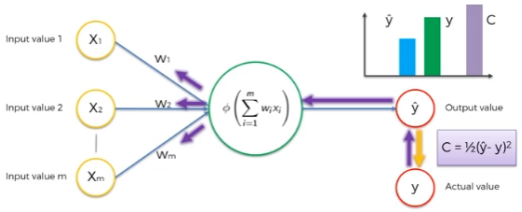

Figure 2.9: A Simple Neuronal Network. Image by Juan Gabriel Gomila

Gradient Descent

How does the neural network optimize the cost function? In the neural network, we have the problem of dimensionality, which means that the dimension of the problem grows so much that it is impossible to solve it. For that, we use gradient descent.

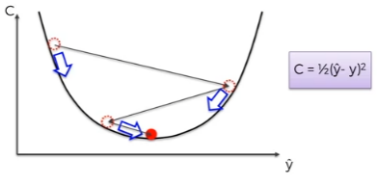

Figure 2.10: Descent Gradient. Image by Juan Gabriel Gomila

The idea of gradient descent[13] is to find the minimum of a function, indicating the tangent (direction) through the calculation of the gradient (the derivative), to try to minimize the function. We only change the weights of the neural network, which forces the cost function to be smaller. Randomness is added to this function to improve the algorithm, stochastic gradient. This helps us if a function is not convex. While the gradient descent in a multidimensional space would look for the local minimum (a sub-optimal solution), the stochastic gradient descent looks for the global minimum, the optimal solution to the problem. At the data level, we do gradient descent to give you the whole block of data, so it would not be optimal. Whereas in the stochastic gradient, we give you a set of data, and then we correct the weights of the neural network, and so forth. The problem is that this would have a very high computational cost, so we do a mixed one, we introduce mini-blocks(mini-batch) to the neural network.

Back Propagation

Finally, the neural network is responsible for correcting the weights through backpropagation [7], which is the part of the algorithm that allows us to fix the global adjustment of the neural network weights so that all the weights are adjusted simultaneously. The correction is made once all the data has been passed through the neural network.

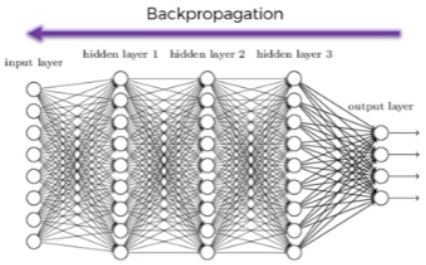

Figure 2.11: "Neural Networks and Deep Learning". Michael A. Nielsen

The image above shows the backward propagation technique, where each of the layers must update weights between neurons in order to minimize the error in the next prediction.

2.2 Reinforcement learning with Finance

This section looks at all the state-of-the-art seen above, together with the latest deep reinforcement learning algorithms applied to finance. It reviews some existing and recently published methods that also address the trading problem. The techniques used will be briefly mentioned and explained to put this project in perspective and get a broader context. It is worth differentiating that there are previously conducted studies where recurrent neural networks or some statistical analysis is used to predict an asset's price, while these studies try to intelligently automate the buying and selling of assets without human interaction.

As I mentioned in the introduction, trading automation is not a new problem. In 2012, before the rise of neural networks, reinforcement learning was explicitly introduced to the Q-Learning algorithm [14] applied to artificial and real-time daily financial asset prices.

Some new methods [15, 16] based on Deep Learning appeared in 2019. For example, a new policy based on gradient descent and new algorithms such as Advantage Actor-Critic and Proximal Policy Optimization was introduced, optimizing state generation to the agent. The same year another study introduced Asynchronous Advantage Actor-Critic with recurrent neural layers to simulate agent memory.

A year later, using the agent-based on Deep Q-Learning [17] was included in different assets, obtaining promising results using good optimization. Finally, at ICAIF 2020, a paper [18] was presented where all types of reinforcement learning algorithms were used, including for the first time the use of technical indicators and the use of the OpenAI Gym library to generate the appropriate environment.

A new study in the Chinese stock market [19] indicated Reinforcement Learning with Convolutional Neural Networks[20] to obtain the agent's policy. They were getting better results and faster training.

Currently, two studies [21, 22] have obtained good results—both trained agents based on the Proximal Policy Optimization algorithm. The latter research has included using multi-agents configured with a novel rule-based policy approach to improve their

decision-making by adjusting their choice of action in the face of state uncertainty. As the same paper indicates, learning based on risk curiosity acts as an intrinsic reward function. It is heavily loaded with signals to find salient relationships between stock and market behaviors so that its actions constantly improve.

Data Analysis

How do we receive the data? What data will we receive? Where will we receive it from? All these questions are addressed in this section. Information is one of the most critical parts of an artificial intelligence project and takes up 80% of the time. In this project, the data is straightforward to obtain through the exchanges API.

3.1 Candles

The candlesticks we see in a financial market, called Japanese candles[23], are a graphical representation that allows us to understand the behavior between buyers and sellers of assets in the markets. This candlestick represents the relationship between the opening price and the closing price that form the candlestick's body. When the candle's closing price is higher than the opening price, the candle's body will be green. When the opposite is the case, the candle will be red. The thin lines marked outside the candle's body are called shadows where they indicate the high and low that was reached in that candle and can be of different lengths, short or long. Each candle will close according to the time we mark, and there are candles for minutes, hours, and days.

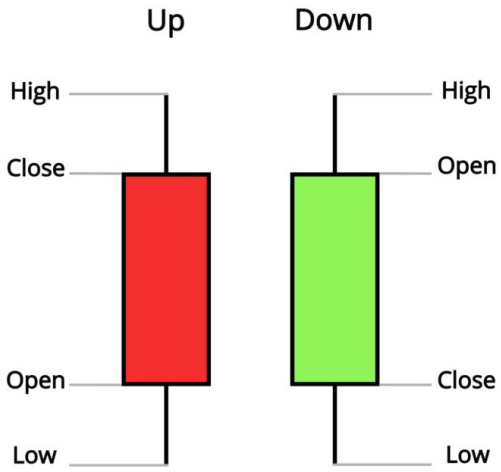

Figure 3.1: Basic trading candlesticks. Image by Hallblazzar

3.2 API

To obtain these candlesticks, we will connect to the Binance API (Application Program Interface).API is a set of definitions and protocols for the development and integration of application software, and the communication between two software applications is realized through a set of rules. Therefore, we can call the API a formal specification, which determines how one software module communicates or interacts with another module to implement one or more functions. But, of course, it all depends on the application that will use them and the permissions granted to third-party developers by the API owner.

Binance is a Centralised Exchange. An Exchange is an access point to buy and sell these cryptocurrencies. If we read the Binance API documentation[24], we will find that the information perceived will be through a JSON structure:

```
'''
    [
        [
            1499040000000,      # Open time
            "0.01634790",       # Open
            "0.80000000",       # High
            "0.01575800",       # Low
            "0.01577100",       # Close
            "148976.11427815",  # Volume
            1499644799999,      # Close time
            "2434.19055334",    # Quote asset volume
            308,                # Number of trades
            "1756.87402397",    # Taker buy base asset volume
            "28.46694368",      # Taker buy quote asset volume
```

```
            "17928899.62484339" # Can be ignored
        ]
    ]
'''
```

The code above is all the information we will receive about a candle. One candle equals one row in our table, so this candle must be formatted so that each piece of information is a column in our data-set. Opening price, highest price, lowest price, and closing price (OHLC), this type of chart is often used for technical analysis and means the following:

- Open Time: Time and date of candle opening in UNIX system format.
- Open: Opening price.
- High: Highest candle price.
- Low: Lowest candle price.
- Close: Closing price.
- Volume: Volume is the number of shares traded in a given period of time.
- Close Time: Time and date of candle closing in UNIX system format.

3.3 Indicators

Trading indicators are mathematical calculations represented as lines on a price chart and can help traders identify specific signals and trends within the market.
Many mathematical indicators can be found in different libraries of any programming language. In our case, we will use TA-LIB[25], where the following list of mathematical indicators is provided:

- Overlap Studies
- Momentum Indicators
- Volume Indicators
- Volatility Indicators
- Price Transform
- Cycle Indicators
- Pattern Recognition
- Statistic Functions
- Math Transform
- Math Operators

In our case, we will focus on two types of indicators because of how we want to address the problem. The following definitions are provided by Ta Library documentation [25]:

3.3.1 Momentum Indicator.

Indicates the speed or strength of a move. How fast the price is growing. It means by its definition that they are used in short-term trading mainly.

Average Directional Movement Index (ADX).

The positive direction indicator (+DI) and negative direction indicator (-DI) are derived from the smoothed average of these differences and measure the direction of the trend over time. These two indicators are usually collectively referred to as the Directional Movement Index (DMI).
The Average Directional Index (ADX) is derived from the smoothed average of the difference between +DI and -DI and measures the strength of the trend over time (regardless of the direction). Using these three indicators in combination, the chartist can determine the direction and strength of the trend.

$$ADX = \frac{(+DI - (-DI))}{(+DI + (-DI))}$$

Listing 3.1: ADX Code

```
real = ADX(high, low, close, timeperiod=14)
```

Relative Strength Index (RSI).

Compare the range of recent gains and losses in a specific period to measure the speed and change of securities prices. It is mainly used to try to identify overbought or oversold conditions in asset trading.

$$RSI = 100 - \frac{100}{1 + RSI}$$

Listing 3.2: RSI Code

```
real = RSI(close, timeperiod=14)
```

3.3.2 Volume Indicator.

Volume indicator helps to determine the price direction of security and the strength of the price change. By definition, it uses the number of ticks (price change) that have appeared during a time interval or, in other words, the volumes of transactions made.

Volume-price trend (VPT)

Based on the running cumulative volume, increase or subtract the multiple of the stock price trend and the percentage change of the current trading volume, depending on the upward or downward movement of the investment.

$$VPT = PreviousVPT + Volume \cdot (\frac{Today'sClosingPrice}{PreviousClosing})$$

Listing 3.3: RSI Code

```
VPT = VolumePriceTrendIndicator(close, volume)
```

3.4 Indicator study

In variable studies, one of the first rules to learn is that adding more information through variables is not the best option, but that quality rather than quantity is more important in order to avoid the problem of dimensionality. There are many indicators in trading. I have taken the three above that I know and that a beginner usually uses. To study these indicators, we can use Pearson's correlation on the variables calculated with the cryptocurrency dataset. The correct way to analyze these types of indicators is to do it independently for each type of indicator, for example, to study all the variables of Momentum Indicators. Still, here I show you that there can also be problems between two types of indicators when it comes to putting them together.

The Pearson correlation coefficient aims to indicate the degree of association between two variables, therefore:

- Correlation is less than zero: If the correlation is less than zero, it is negative. That is, the variables are negatively correlated. When the value of one variable is high, the value of the other variable is low—the closer to -1, the more precise the extreme covariance. If the coefficient is equal to -1, we are referring to a complete negative correlation.
- The correlation is more significant than zero: If the correlation is equal to +1, it means it is entirely positive. In this case, it means that the correlation is positively correlated. That is, the variables are directly correlated. When the value of one variable is high, the value of another variable is also high, and the same is true when it is low. If it is close to +1, the coefficient will be covariant.
- Correlation is zero: When the correlation is zero, it means that a specific covariant meaning cannot be determined. However, this does not mean that there is no non-linear relationship between the variables.

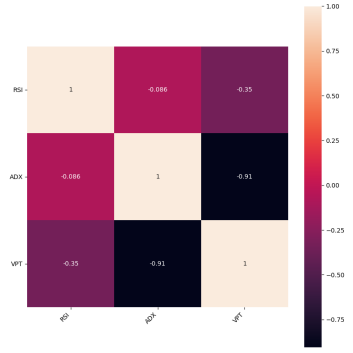

Figure 3.2: Indicators in Bitcoin

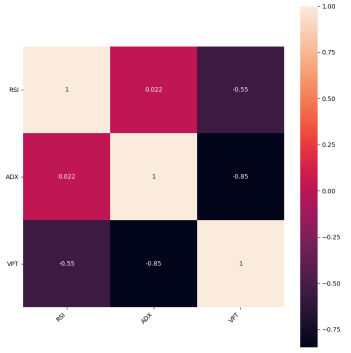

Figure 3.3: Indicators in ADA

As we can see, the results are practically the same, although it varies slightly because there are different types of currencies. With these images, we can conclude that adding more indicators does not mean obtaining better information, as there are indicators that

may contradict each other. Therefore, we must follow the KISS (Keep it Simple, Stupid!) pattern, which means that the more difficult it is to make it easy, the more minimalist our set of variables, the better, as long as those variables are of quality.

3.5 Normalized Data

In this section, we will proceed to normalize the data. Nowadays, Bitcoin is the reference. So we will study its trend. In the following image, we can visualize the Bitcoin history since 2011. Now, we have to proceed to make the study and normalize to adapt the data-set to the environment.

Figure 3.4: Historical Bitcoin

Once normalized, we can see that the values are between 0 and 1, but the trend is still displayed and depending on the time. This is wrong.

Figure 3.5: Bitcoin Normalized

One of the ways to remove seasonality is to make the difference between the current value and the previous value (yield difference) and then use the sigmoid function to

normalize.

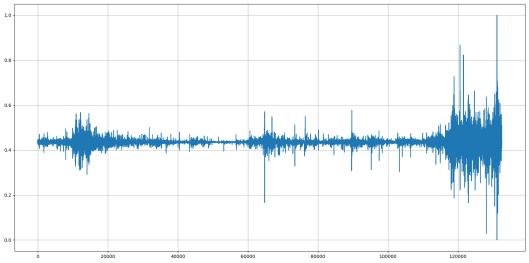

Figure 3.6: Basic trading candlesticks. Image by Hallblazzar

We have removed the dependence on time, but we can still observe that there is some seasonality. Therefore, in trading, as can be seen in the original image, the logarithmic scale is used to remove the seasonality to add the logarithmic function at each time difference.

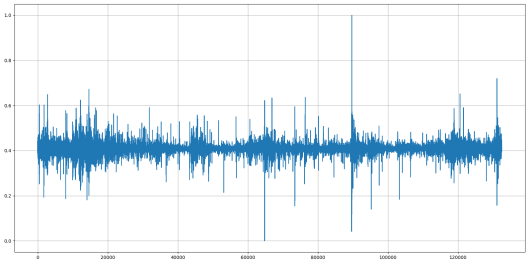

Figure 3.7: Logaritmic Bitcoin Normalized

After analyzing the data-set, we now have the correct data for the input of our future bots.

Methods

4.1 Model Classification

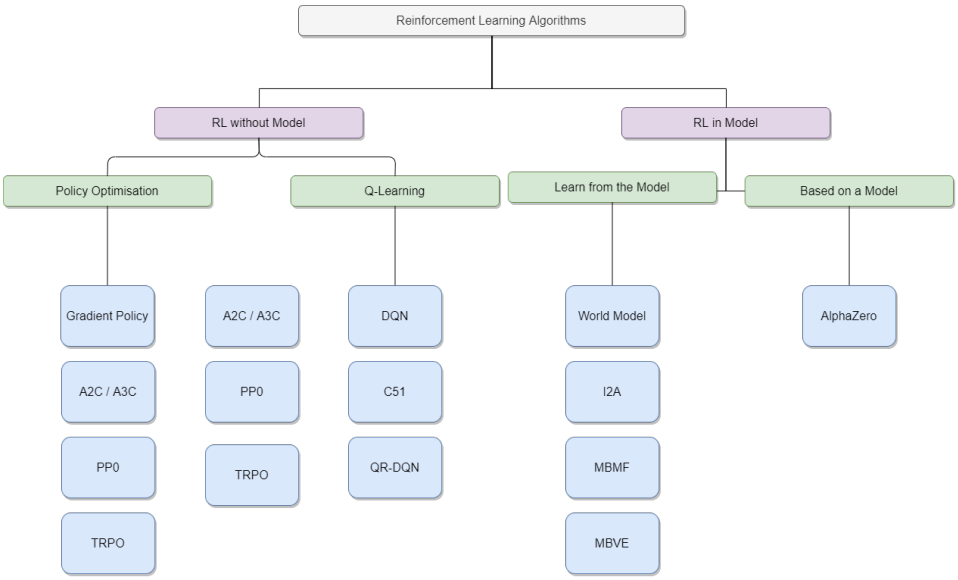

Figure 4.1: Model Classification

As we can see in the previous image, we have algorithms based on non-models and models, and this means that the algorithm learns how the environment works and if it uses its dynamics of changes in it, so it has all the provision of the set of probabilities to move from one state to another. Therefore, for our problem it is not viable.

We find two differences on the side of model-free algorithms based on policy optimization (on-policy) and algorithms based on Q-Learning (off-policy). In our case, we will mainly focus on the off-policy models, although we will implement some on-policy models; basically, they are algorithms that learn through the value function seen above, and their behavioral policy and their objective policy can be different while on the other side they must be the same.

4.1.1 Deep-Q-Learning

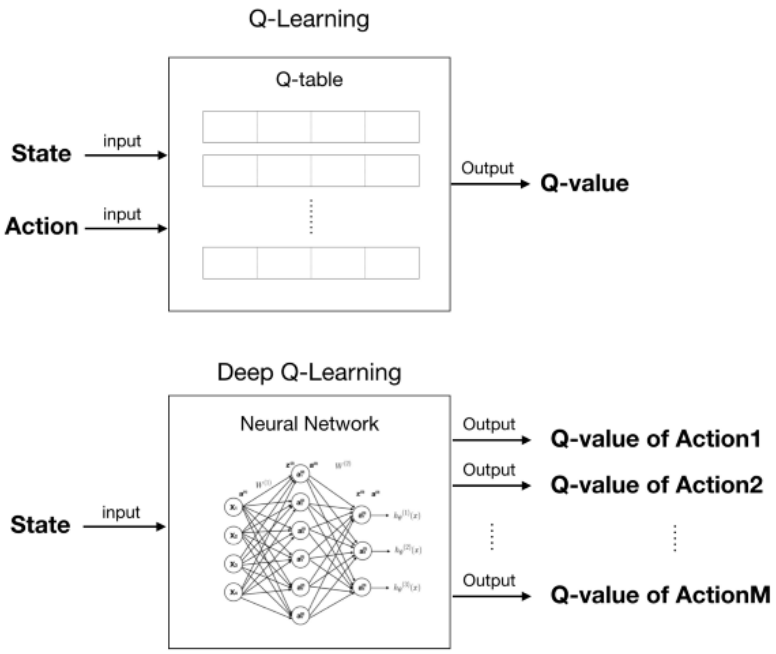

Figure 4.2: Q-Learning vs Deep-Q-Learning. Image by Amber

The Q-Learning algorithm seen above works very well when the environment is simple. For example, the function Q(s, a) can be represented as a table or matrix of values, for instance, in mazes like Pacman or austere 2D environments. But when the number of states and actions is much more complex in computational speed and space, it becomes infeasible. Thus arose Deep-Q-Learning[26] where neural networks allow us to approximate non-linear functions. Thanks to its great potential, it is used to approximate the Q-function; in other cases, it is also used to approximate the objective function.

Now we take the states where our neural network will input and predict our Q values. In the neural network, there is no time difference as in Q-Learning. We do not compare and make the difference of Q values. The neural network can predict and remember what has happened before because the agent has previously passed through that state.

To train the neural network, one more function must be added to the equation.

$$L = \sum(QTarget - Q)^2$$

So basically, we measure the error with a loss function before we measure the time difference, so it becomes the expected value minus the prediction. Each of the values of that difference can be positive or negative, so what you do is square it so that difference indicates how likely the error is.

Experience Replay

Listing 4.1: A space in our tuple

```
[observation, action, reward, done flag, next state]
```

Experience[27] allows us to remember solutions to previously solved problems and not change the state and action, at every moment. As the neural network predicts for every state change, for example, when a vehicle is in a straight line, as a consequence giving the neural network the same state and action, in our case if an asset is in total rise, it would be to maintain the action of not selling. This, in turn, means that we stop visiting the environment episodically first to collect some data on the states saw in the past and then train our neural network on the accumulated experiences. The main idea is to store the agent's experience in the form of a tuple <state, action, reward, next state> and then extract batches from those experiences to train the neural network to increase the robustness of the learning. These extractions are done randomly to learn from a much broader context of past experiences and avoid learning from what has happened recently. This memory can be implemented in the form of a buffer, like a cyclic queue, as new memories arrive and old ones are removed since the memory capacity of the buffer is fixed.

Prioritized Experience Replay

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

$$w_i = ((\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$$

$$p = priority, P = probability, w = weight, N = numberofexperiences$$

Prioritize [28] our buffer helps speed up the problem-solving process because samples drawn with higher weights are drawn more frequently. In DQN, we randomly sample experiences in a linear distribution, which means that we only need a container to store experiences. We don't have to worry about how the buffers are sorted. In the repetition of priority experiences, we need to associate each experience with additional information, priority, probability, and weight. The focus is updated according to the loss obtained after the neural network is forwarded. This calculate the probability based on the priority of experience. In contrast, the weight (correcting the bias introduced by non-uniform sampling during the neural network backpropagation) is calculated based on probability.

4.1.2 Double Deep-Q-Learning

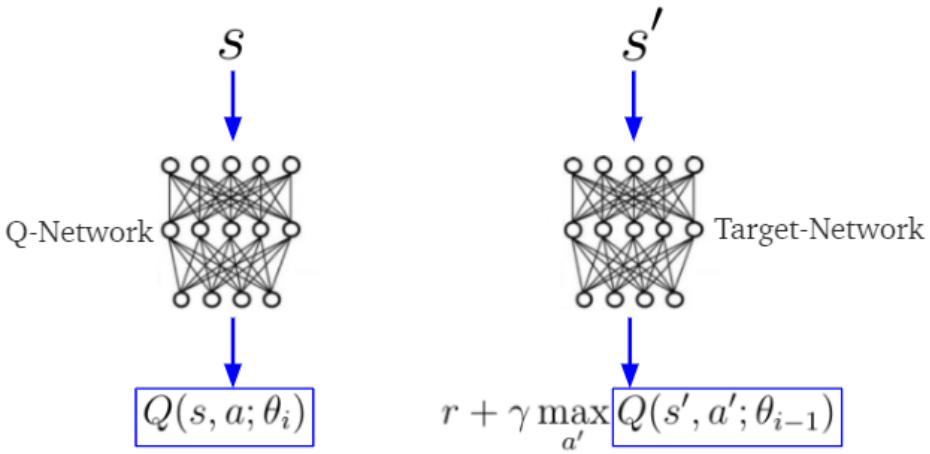

Figure 4.3: Double Deep Q-Learning. Image by Artem Oppermann

The problem with the DQN algorithm is that it tends to overestimate the rewards, i.e., the Q-values it learns to think that they will get a higher reward than they actually will. So the paper [29] shows us an improvement of the DQN. The procedure is to separate the search and selection of the action. On the other hand, the evaluation of the action shows that the first neural network chooses what will be the best possible action, and the following network will evaluate that action to get the Q-value.

4.2 Gradient Policies Concept

On one hand, these algorithms will learn a neural network that returns us the Q values. On the other hand, we have a second neural network taking the states of the environment and returning the action to execute. Therefore, we only define a policy function that estimates the probability of taking each possible action from each state. The advantage we gain over the value-based algorithms seen previously is that we can represent continuous actions; for example, in a car, we could predict how fast we want to accelerate. Also, they work better in stochastic environments, so we could significantly improve our problem where there is high volatility. Finally, they directly optimize the function to be optimized and tend to converge faster. The gradient policy[30] aims to maximize the expected return, which means that it tries to obtain the maximum rewards as fast as possible in the short term. Its update is given by intermediate weights and not with Q values through the ascending gradient in the direction of the increasing function, thus increasing the expected return.

4.2.1 Deep Actor-Critic

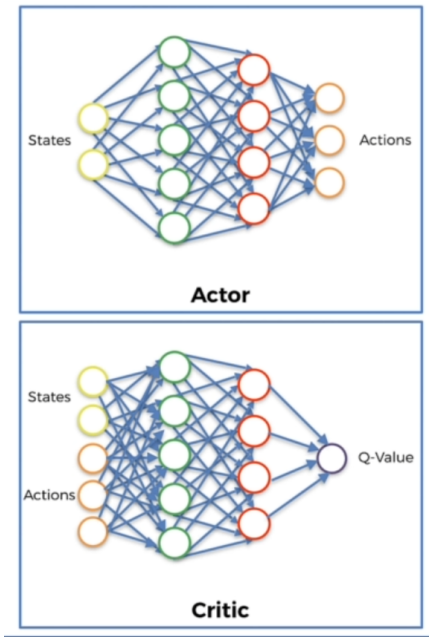

Figure 4.4: Architecture Actor-Critic. Image by Juan Gabriel Gomila

We have two distinct neural networks, Actor and Critic [31], The actor is directly in charge of the policy, and it will take as inputs the states of the environment and return the actions it has to execute. In contrast, the critic considers the states and the actions, so they are concatenated, then it evaluates and returns the quality of the actions, i.e., the Q-value. For example, a student takes an exam (Actor) and corrects the exam (Critical). In our case,

we will define that the actor will invent future finances, and the critic will decide whether they can be valid or not.

4.2.2 Proximal Policy Optimization

Proximal Policy Optimization[32] is an algorithm whose goal is to optimize the strategy for a given state and action pair (s, a) without always knowing the complete state and action. In the case where we have an unknown conversion function, we can use *inverse strategy* to obtain the strategy. The inverse strategy is the strategy opposite to the transfer function. Similarly, the near-end strategy is received by the neighboring strategy as a function of the action. For any strategy, the strategy gradient method is an iterative algorithm that iteratively updates the strategy.

This feature aims to perform a gradient ascent step (equivalent to doing a reverse gradient descent). In this way, our business representatives are forced to take actions that bring higher returns and avoid bad behavior. After the update, the distance between the new strategy and the old strategy should not be too far. For this reason, PP0 uses clipping to avoid excessive updates. In the near-end strategy optimization, the distance is accumulated and used to construct a decompression learning model. In this way, in each training step, the system can download the optimal value of the actor more efficiently than other techniques.

4.2.3 Deep Deterministic Policy Gradient

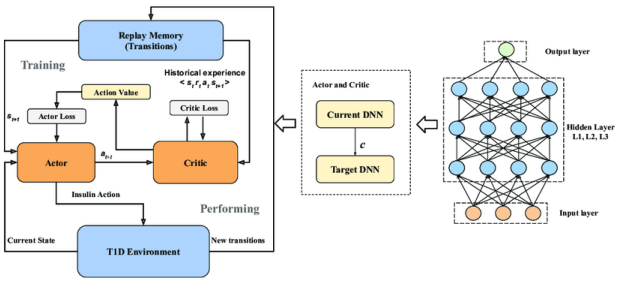

Figure 4.5: Architecture DDPG. Image by Zhu, Taiyu and Li, Kenneth and Kuang, Lei and Herrero [33]

In traditional reinforcement learning, we are interested in maximizing rewards in each period. In DDPG[34], we are traditionally only interested in maximizing rewards in one episode. This algorithm is called a "deterministic" algorithm because it does not make decisions based on the past. Instead, the decision is made based on the current observations, and the policy gradient is calculated based on the recent observations. Deep Deterministic Policy Gradient[35] is an algorithm used to solve the policy gradient problem. Strategy gradient is a well-known technique used in reinforcement learning based on the calculation of strategy gradient. The idea is that the policy gradient of the policy function is a direct result of the gradient of some loss function concerning the proxy policy, and the

gradient can be calculated efficiently by finding the derivative with respect to the policy. Here, a new algorithm is proposed to calculate the policy gradient effectively. This technology is based on the idea of deterministic policy gradient. The algorithm includes calculating the loss function of certain policy functions and the gradient of the policy of a single agent. The idea is to update the strategy using the strategy gradient algorithm, which can be obtained by finding the derivative of some deterministic strategy. This is done by finding a deterministic strategy sequence because it does not depend on previous actions. The gradient of the strategy will be calculated, and the strategy sequence will be updated using the derivative strategy algorithm. Therefore, the technique is easy to implement because the algorithm is computationally simple. The algorithm is based on the following two key ideas: The policy gradient is calculated for the entire state space, not just for the current state. There are many more states in the state space than actions. Therefore, there is no need to calculate the current state and following state strategies. The calculation can be done efficiently by using the action space. The action space is much smaller than the state, and the state is much less than the action. When we want to calculate the gradient of the strategy, we must calculate the derivative of the action at the current time step. Therefore, the strategy is updated by calculating the derivative of the loss function under specific actions, and the strategy gradient algorithm must be used to calculate the derivative. We introduce a new deterministic action gradient technique, which uses the action gradient algorithm to calculate the action gradient. For example, we can calculate the action gradient of the current step and any action performed in the action sequence. This can be done by using the motion gradient algorithm to calculate the motion gradient. However, this requires us to know the order of operations, which we do not know in advance. Therefore, we cannot use the operation update (operation sequence) in the next step to calculate the policy gradient. In our deterministic algorithm, the action is updated at the same time as the target action. Therefore, it is not necessary to calculate the derivative at each time step. The derivatives of the loss functions are always stored in a table and can then be used in any decision-making step. This makes the algorithm very easy to implement. Further differences between the deterministic and stochastic algorithms are described below. More precisely, the strategy gradient of deterministic calculation is always stored in the matrix. In other words, we can calculate all policy gradients. The randomly calculated policy gradient can be stored in a vector. On the other hand, stochastic gradients cannot be stored in vectors. In this case, we must calculate all gradients at once. It is possible to perform calculations at the same time. However, sometimes the analysis is performed longer than the calculation, for example, when calculating a random value function.

4.2.4 Twin Delayed DDPG

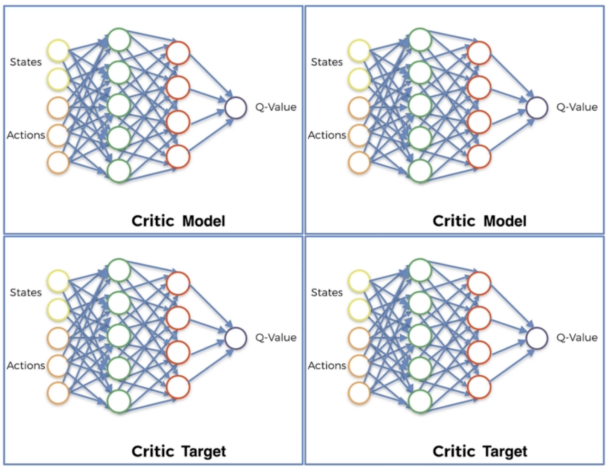

Figure 4.6: Architecture Twin Delayed DDPG. Image by Juan Gabriel Gomila

Although DDPG can sometimes achieve excellent performance, it is usually fragile in hyper-parameters and other types of adjustments. The typical failure mode of DDPG is that the learned Q function begins to significantly overestimate the Q value, which will cause the strategy to be interrupted because it takes advantage of the error in the Q function. Double Twin Delay DDPG (TD3) is an algorithm that solves this problem by introducing three key techniques:

- Tip 1: Tailor double-Q learning. TD3 learns two Q functions instead of one (hence the "twin") and uses the smaller of the two Q values to form the target in the Bellman error loss function.
- Tip 2: "Delay" policy updates. The frequency of the TD3 updating strategy (and target network) is lower than the Q function. Therefore, the paper recommends a policy update every two Q function updates.
- Tip 3: Smooth the target policy. TD3 adds noise to the target action and smoothing Q, and the changes in the action make it more difficult for the strategy to take advantage of Q function errors. In summary, these three techniques have greatly improved the performance of the baseline DDPG.

4.2.5 Soft Actor-Critic

Soft Actor-Critic[36] is a training technique in which actors do not rely on evaluating their limited control rights but rely on supporting actors. The actor has auxiliary functions that can determine its growth radius and direction. People can use it instead of the feedback function. Soft Actor-Critic (SAC) is an algorithm that optimizes stochastic strategies in a non-strategic way, thus building a bridge between stochastic strategy optimization and DDPG style methods. It is not a direct successor to TD3 (direct concurrent release).

Still, it contains tailored double-Q techniques, and due to the inherent randomness of the strategy in SAC, it also benefits from target strategy smoothing.

A core feature of SAC is entropy regularization. The strategy is trained to maximize the trade-off between the expected return and entropy, a measure of randomness in the strategy. This is closely related to the trade-off between exploration and development: increasing entropy will lead to more exploration, speeding up future learning. It also prevents the strategy from prematurely converging to a bad local optimum.

Design of experiments

In this section, we will explain the whole design of experiments performed. First of all, we create an environment, as this will give us an advantage over working directly on the data. Using the environment, we can manipulate all the behavior we want the algorithms to have. It is like defining the rules of the game, which would be to create a local Exchange adapted so that the different algorithms can be executed. It also has a "check" function that allows you to see if your environment is well created, so it is advantageous. By inheriting from class(gym.Env), we get the following functions. On the other hand, all these designs will be run on a machine with 16GB of RAM, an Intel i7, an Nvidia RTX 3090 Founders Edition GPU, and a SATA SSD. All these algorithms run on Google Colab, but as you might get kicked out of the session, I decided to run it locally to see the graphs on the Tensorboard (to see if the algorithms are learning).

5.1 Environment

We can create different types of environments. Python environment, OpenAI Gym environment[37], Tensorflow[38] environment, and so on. In my case, I use the OpenAI Gym environment, as it has a lot of documentation and many predefined environments like Atari machines.

Listing 5.1: Custom Environment Example

```
def __init__(self)
def step(self, action)
def reset(self)
def render(self, mode = 'human')
```

In the init function, we must modify the space of observations, with the minimum and maximum we need, based on the needs of the environment and the space of actions according to the environment's needs. In addition, it is advisable to define the range of rewards. Those specific algorithms need a discrete action space while other algorithms need a normalized and continuous action space. We must also declare all the necessary variables for our environments, such as the data frame, the commissions, portfolio, and initial value. We will use discrete and Box spaces.

Listing 5.2: Custom Environment Example

```
Box -> R^n (x1,x2,x3,...,xn), xi [low, high] #gym.spaces.
#gym.spaces.Box(low = -10, high = 10, shape = (2,)) # (x,y), -10<x,y<10

# Discrete -> Integers between 0 and n-1, {0,1,2,3,...,n-1}
#gym.spaces.Discrete(5) # {0,1,2,3,3,4}

#Dict -> Dictionary of more complex spaces
#gym.spaces.Dict({
# "position": gym.spaces.Discrete(3), #{0,1,2}
# "velocity": gym.spaces.Discrete(2) #{0,1}
# })


# Multi Binary -> {T,F}^n (x1,x2,x3,...xn), xi {T,F}
# gym.spaces.MultiBinary(3)# (x,y,z), x,y,z = T|F

# Multi Discrete -> {a,a+1,a+2..., b}^m
#gym.spaces.MultiDiscrete([-10,10],[0,1])

# Tuple -> Product of simple spaces
#gym.spaces.Tuple((gym.spaces.Discrete(3), gym.spaces.Discrete(2)))#{0,1,2}x{0,1}

# prng -> Random Seed
```

Step function. This function executes the action determined at each step to guide the agent in the environment. The reset method will also be executed at the end of each episode, and it does the game over function when we have no more states in the space of observation. It is essential to note that this function calculates the action taken. In our case, we have separated the action into two functions according to the type of algorithm (discrete or continuous). This also calculates the reward based on the action. The following observation configures the state if it is a terminal state (done) and optionally defines the values to be persisted within the info dictionary.

Reset Function. This function resets the environment variables, and we return the initial observation after having configured a whole episode. Thus, on one hand, we also separate it into two functions, in one function all the variables related to the environment and in another function all the variables of the session user.

Render function. This is very important since we can represent on screen all the values of the environment as well as its graphical representation. In the case of making an application, it would be the output that the user would get. In my case, we take the idea of a market representation, and I also create a return of several variables that interest me.

5.2 Methods

All methods have been extracted from the stable-baselines3 library[39] written in Pytorch[40]. It should be noted that the Double-Deep-Q-learning model is not found in this library but in its first version[41] written in Tensorflow. As my GPU is not compatible with Tensorflow version one, I have decided to use Google Colab[42] for this specific method so that training times may vary. Furthermore, as we are only studying the be-

haviour of these algorithms in trading, we will only adopt particular values such as the memory size, the number of episodes, and the block size so that all these algorithms are compatible with my equipment, as some of them consume huge amounts of memory. Furthermore, all algorithms use MlpPolicy, meaning that their inner layers are Dense layers instead of, e.g., Convolutional Layers. Total timesteps are assigned by 200,000 units, which is the number of steps in total the agent will do for any environment. The total timesteps can be across several episodes, meaning that this value is not bound to some maximum.

5.2.1 Discretize Action Algorithms

Here the important thing to note, as in theory, is the importance of the replay buffer where we have allocated a space of 100000 units. In addition, the action space will be defined by action space = spaces.Discrete(3) in the environment will have a 3-dimensional shape (buy, sell and hold).

Listing 5.3: Q-Learning Function

```
stable_baselines3.dqn.DQN(policy, env, learning_rate=0.0001,
buffer_size=1000000, learning_starts=50000, batch_size=32, tau=1.0,
gamma=0.99, train_freq=4, gradient_steps=1, replay_buffer_class=None,
replay_buffer_kwargs=None, optimize_memory_usage=False,
target_update_interval=10000, exploration_fraction=0.1,
exploration_initial_eps=1.0, exploration_final_eps=0.05, max_grad_norm=10,
tensorboard_log=None, create_eval_env=False, policy_kwargs=None, verbose=0,
seed=None, device='auto', _init_setup_model=True)
```

This algorithm is a derivation of the original deep q learning. To activate it, we only need to set double q=True.

Listing 5.4: Double-Q-Learning Function

```
stable_baselines.deepq.DQN(policy, env, gamma=0.99, learning_rate=0.0005,
buffer_size=50000, exploration_fraction=0.1, exploration_final_eps=0.02,
exploration_initial_eps=1.0, train_freq=1, batch_size=32, double_q=True,
learning_starts=1000, target_network_update_freq=500,
prioritized_replay=False, prioritized_replay_alpha=0.6,
prioritized_replay_beta0=0.4, prioritized_replay_beta_iters=None,
prioritized_replay_eps=1e-06, param_noise=False, n_cpu_tf_sess=None,
verbose=0, tensorboard_log=None, _init_setup_model=True, policy_kwargs=None,
full_tensorboard_log=False, seed=None)
```

5.2.2 Continuous Action Algorithm

These algorithms must be defined in the environment with action space = spaces. For example, box (low=-1, high=1, shape=(3,)), this means that space will be normalized between -1 and 1 and will have a 3-dimensional shape (buy, sell and hold). Recall that the model we will see below relies on tracking estimated future reward returns (our value function) and learning new and more complex strategies to follow in order for our agent to be in a longer time horizon and get higher rewards. With the algorithm now optimizing two functions simultaneously it quickly becomes a more complex problem.

Listing 5.5: Actor-Critic Function

```
stable_baselines3.a2c.A2C(policy, env, learning_rate=0.0007, n_steps=5,
gamma=0.99, gae_lambda=1.0, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5,
rms_prop_eps=1e-05, use_rms_prop=True, use_sde=False, sde_sample_freq=-1,
normalize_advantage=False, tensorboard_log=None, create_eval_env=False,
policy_kwargs=None, verbose=0, seed=None, device='auto',
_init_setup_model=True)

If you find training unstable or want to match performance of
stable-baselines A2C, consider using RMSpropTFLike optimizer from
stable_baselines3.common.sb2_compat.rmsprop_tf_like. You can change
optimizer with A2C(policy_kwargs=dict(optimizer_class=RMSpropTFLike,
eps=1e-5))
```

Listing 5.6: Proximal Policy Optimization Function

```
stable_baselines3.ppo.PPO(policy, env, learning_rate=0.0003, n_steps=2048,
batch_size=64, n_epochs=10, gamma=0.99, gae_lambda=0.95, clip_range=0.2,
clip_range_vf=None, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5,
use_sde=False, sde_sample_freq=-1, target_kl=None, tensorboard_log=None,
create_eval_env=False, policy_kwargs=None, verbose=0, seed=None,
device='auto', _init_setup_model=True)
```

For DDPG and Twin Delay, we must add noise to our actions. Otherwise, the algorithm will not work, and it is vital to limit the memory usage as they come with a very high default value, which can exceed 16GB.

The reason we added noise is because the action space is very simple, so the algorithm would not explore the space of observations.

Listing 5.7: Action Noise

```
NormalActionNoise(mean=np.zeros(n actions),
                  sigma=0.5 * np.ones(n actions))
```

Listing 5.8: Deep Deterministic Policy Gradient Function

```
stable_baselines3.ddpg.DDPG(policy, env, learning_rate=0.001,
buffer_size=1000000, learning_starts=100, batch_size=100, tau=0.005,
gamma=0.99, train_freq=(1, 'episode'), gradient_steps=-1,
action_noise=None, replay_buffer_class=None, replay_buffer_kwargs=None,
optimize_memory_usage=False, tensorboard_log=None, create_eval_env=False,
policy_kwargs=None, verbose=0, seed=None, device='auto',
_init_setup_model=True)
```

Listing 5.9: Twin Delayed DDPG Function

```
stable_baselines3.td3.TD3(policy, env, learning_rate=0.001,
buffer_size=1000000, learning_starts=100, batch_size=100, tau=0.005,
gamma=0.99, train_freq=(1, 'episode'), gradient_steps=-1,
action_noise=None, replay_buffer_class=None, replay_buffer_kwargs=None,
optimize_memory_usage=False, policy_delay=2, target_policy_noise=0.2,
target_noise_clip=0.5, tensorboard_log=None, create_eval_env=False,
policy_kwargs=None, verbose=0, seed=None, device='auto',
_init_setup_model=True)
```

Listing 5.10: Soft Actor Critic

```
stable_baselines3.sac.SAC(policy, env, learning_rate=0.0003,
buffer_size=1000000, learning_starts=100, batch_size=256, tau=0.005,
gamma=0.99, train_freq=1, gradient_steps=1, action_noise=None,
replay_buffer_class=None, replay_buffer_kwargs=None,
optimize_memory_usage=False, ent_coef='auto', target_update_interval=1,
target_entropy='auto', use_sde=False, sde_sample_freq=-1,
use_sde_at_warmup=False, tensorboard_log=None, create_eval_env=False,
policy_kwargs=None, verbose=0, seed=None, device='auto',
_init_setup_model=True)
```

5.2.3 Procedure

1. Connect to the Binance API with our credentials to obtain the Bitcoin and ADA history. To facilitate this connection, we will use the python-binance library[43].

2. We convert each candle in rows and create the following columns: Open time, Open, High, Low, Close, Volume, Close time, Quote asset volume, Number of trades, Taker buys base asset volume, Taker buys quote asset volume, Ignore.

3. Remove the "ignore" column, change Open Time and Close Time from UNIX format to DateTime.

4. Calculate RSI, ADX, VPT indicators for our dataset. IMPORTANT: We must choose the days we want our indicators to take.

5. We create two environments: Discrete and continuous thanks to a variable that we have previously defined, and we split the dataset into training and test. Then, we introduce the data frame by parameter, and the Windows Size (THIS IS IMPORTANT) means how many rows we want to take as a window for the neural network. In my case, I have taken a window of 2 weeks in candles of 15 minutes. So it would be 4 candles of 15 min in 1-hour x 24h x 14 days = 1344 rows.

6. Once the environments are created, we must validate with the function check env that the environments are correctly created.

7. Thanks to a benchmark function, we agglutinate all the models in a function so that they are executed iteratively.

8. For each model, we create it with the predefined parameters and define the verbose=2, so that all the information and the path of the Tensorboard is displayed on the screen. Then we call the function to learn passing it the parameter timesteps. This way, it will learn our model.

9. Finally, we save the model in a path with the same function.

10. (Optional) if we want to test our model, we only have to follow this pattern with the test data frame:

Listing 5.11: Test DQN model example

```
obs = env_val_discretize.reset()
for i in range(len(df_val)):
    action, _states = dqn.predict(obs)
    obs, rewards, dones, info = env_val_discretize.step(action)
    env_val_discretize.render(mode="system")
```

Results Analysis

This section discusses the results obtained from the previously explained architectures and their later implementation. It introduces a standard way to measure the loss value agent quantitatively and also the results in the test dataset, human-perceived point of view. Examples of where the network gives good results and fails to produce the expected results will also be given and analyzed.

6.1 Discretize Action Algorithms

BTC is taken as a reference in these two algorithms because of the problems that I will discuss below.

6.1.1 Deep-Q-Learning

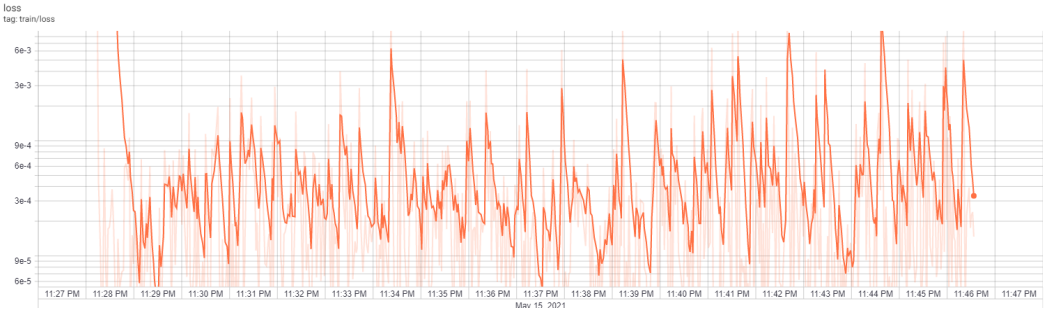

Figure 6.1: DQN Loss

This is not uncommon for reinforcement learning and does not indicate any problems. However, as the agent gets better at the game, it is indeed more challenging to estimate the reward (because it is no longer always 0). In addition, as the reward becomes higher and the average episode length becomes longer and longer, the amount of variance in the reward will also become more extensive, so even if it is necessary to prevent the loss from increasing, it is very challenging.

6.1.2 Test Results

As we have seen, one of the severe problems of brokers who only predict appropriate actions is that they bet or sell all at every step because they have no control over the bet amount, so this is not the best algorithm for trading. Therefore, the agent hardly executes any actions because he learns that betting everything carries significant risk.

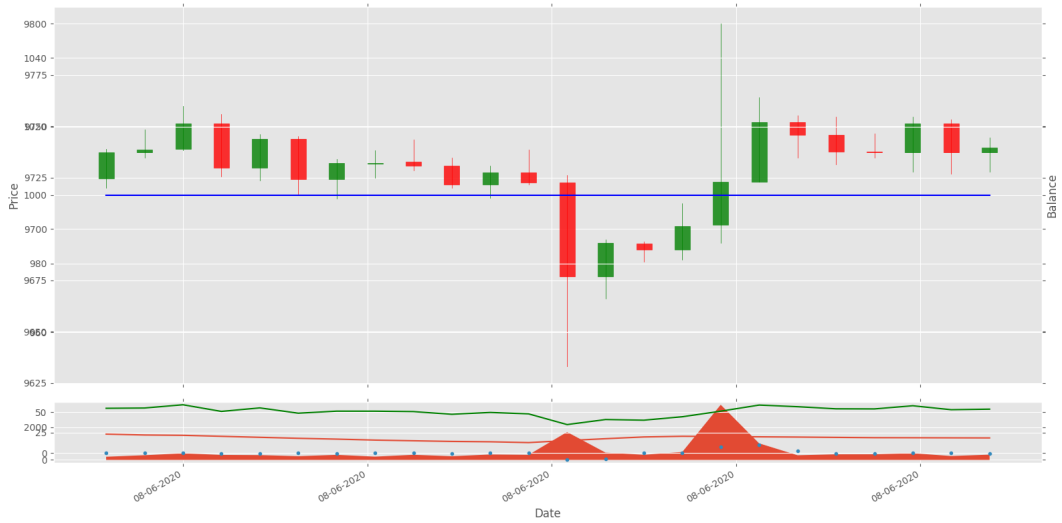

Figure 6.2: DQN Playing in Data Test

6.1.3 Double Deep-Q-Learning

This model learning improves a lot as it does not overestimate rewards by having two neural networks. There is less noise, and the loss is more stable, and it tends to 0, which means that our agent is learning, but as we have commented in the previous model, these architectures do not have control over the amount to bet. They learn not to take risks because a bad move loses all the money, and they will not be able to continue playing.

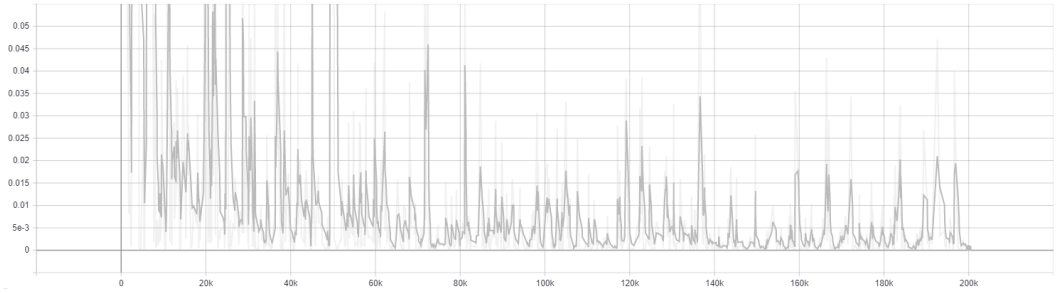

Figure 6.3: DDQN Loss

6.2 Continuous Action Algorithms

Recall that these algorithms have two sets of actions, the action to take and the amount percentage to sell or buy.

6.2.1 Deep Actor-Critic

It was trained with the Adam optimizer and then tested by the optimizer featured in the library, RMSpropTFLike.

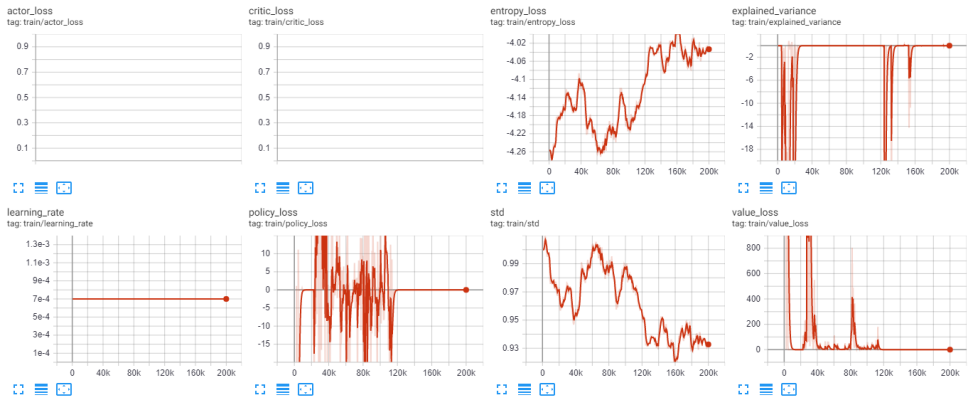

Figure 6.4: A2C tensorboard

It does not really play its due role here. Instead, it treats the problem as a classification, so we get the value entropy loss. There is a serious problem here: the entropy loss gives extraordinary values, which means that it fails to converge, so it does not learn anything.

6.2.2 Proximal Policy Optimization

With both ADA and BTC, we get the same results. Therefore, PPO is able to learn in such circumstances.

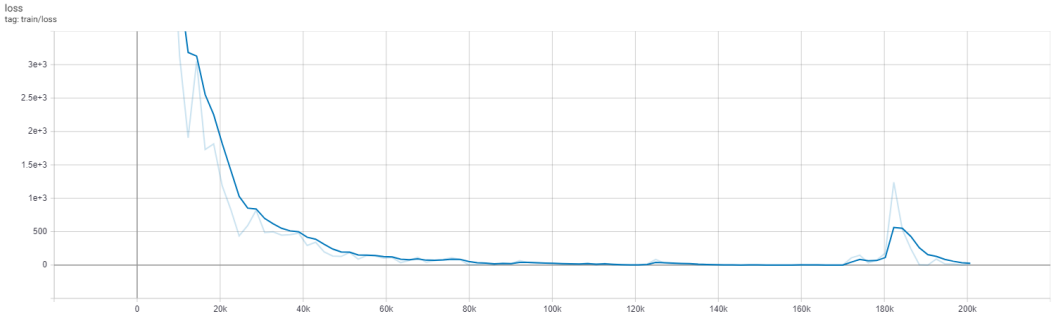

Figure 6.5: PPO Loss

6.2.3 Test Results

As we can see, the blue bar has moved, this means that it has been buying and selling, as it has decreased the total equity we have by -30$ and then increased to -23$, so it is buying and selling.

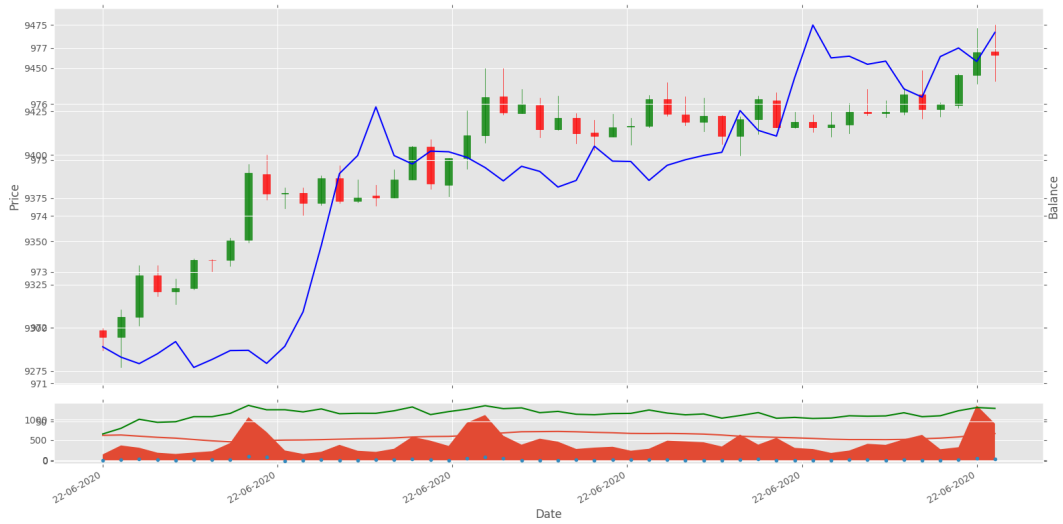

Figure 6.6: PPO Playing in Data Test

6.2.4 Deep Deterministic Policy Gradient

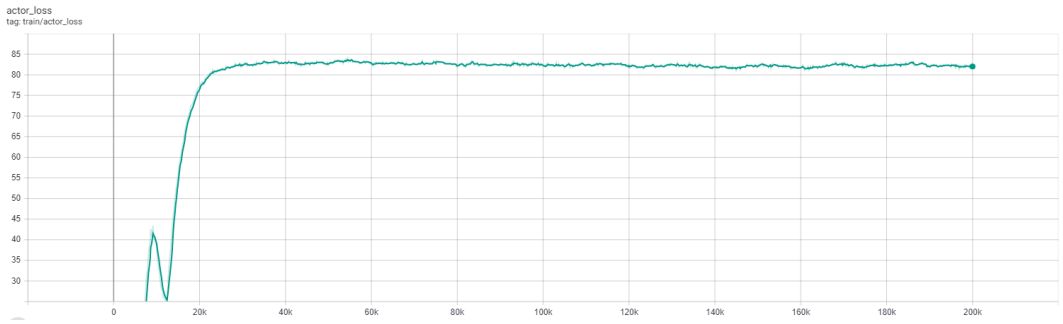

Figure 6.7: DPPG Actor Loss

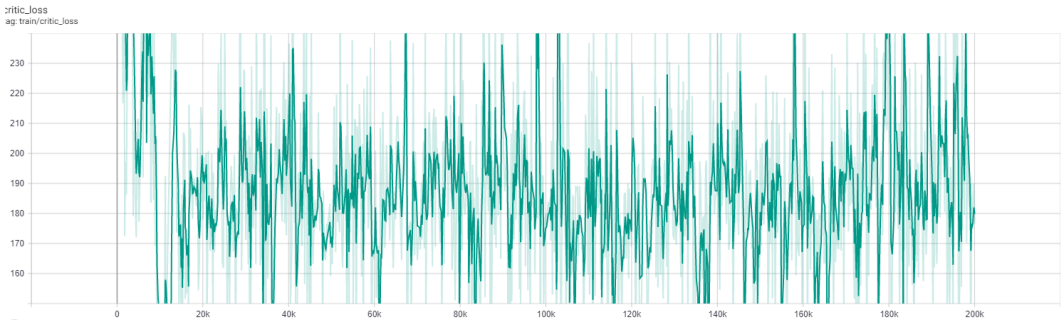

Figure 6.8: DDPG Critic Loss

Actor losses increase, which is good, but on the critic's side, we can see a lot of noise, and it cannot go down. This means that the critic cannot communicate to the Actor whether his decisions are correct.

6.2.5 Twin Delayed DDPG

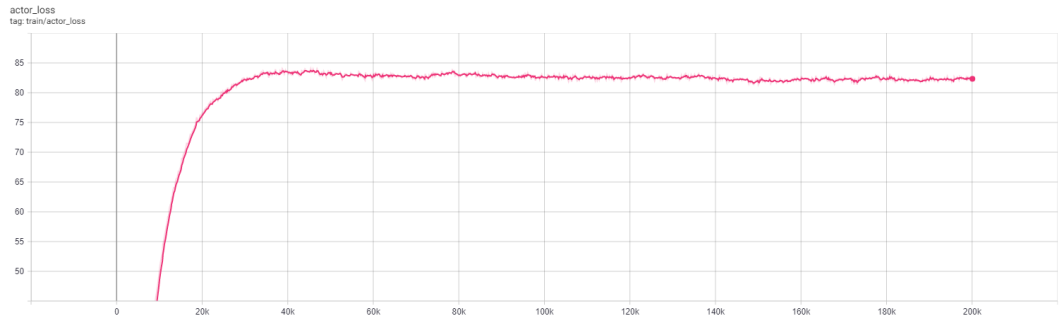

Figure 6.9: TD3 Actor Loss

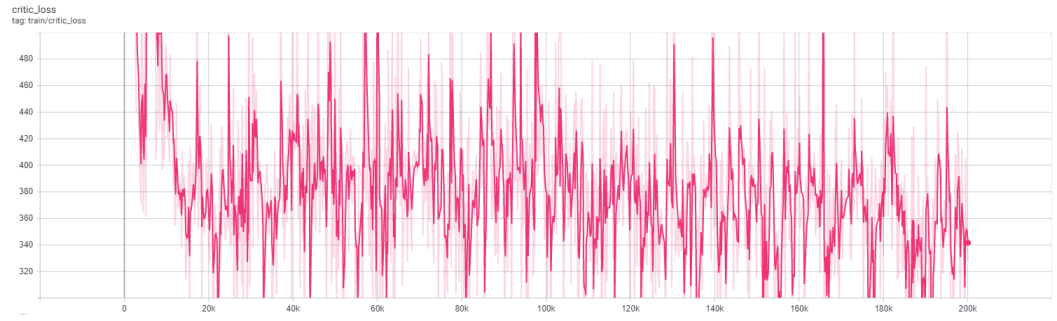

Figure 6.10: TD3 Critic Loss

Here is the same situation, the actor losses increase, which is good, but on the critic's side, we can see that there is a lot of noise. Yet, in this case, we can see a downward trend.

Perhaps more training time would improve this curve.

6.2.6 Soft Actor-Critic

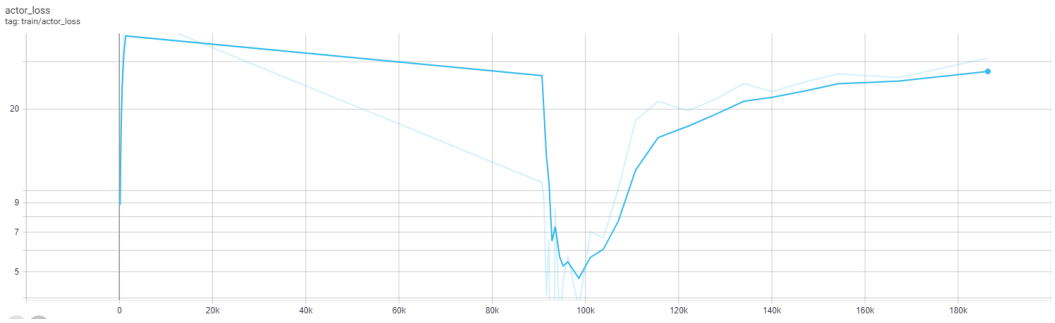

Figure 6.11: SAC Actor Loss

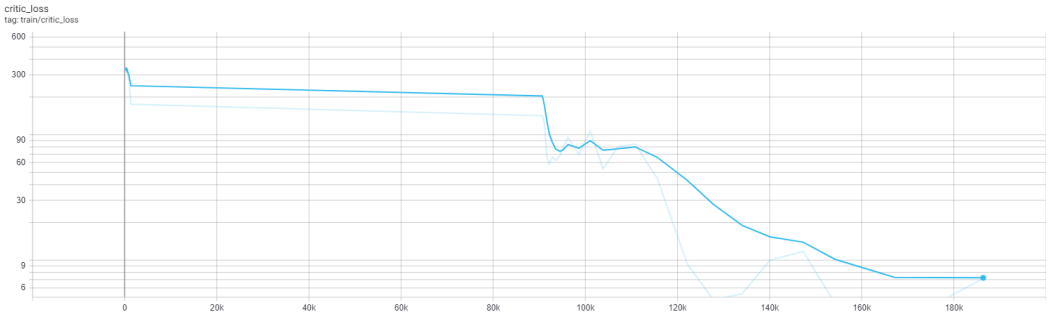

Figure 6.12: SAC Critic Loss

SAC has strange behavior with a linear trend, it is difficult to explain what is happening, but given the results, it is ruled out.

6.2.7 Analysis of Global Results

The results are not optimistic, but there are signs that the agent is indeed learning something:

- Agents. The use of frameworks allows you to implement and see the different behaviors of the agents quickly but with a big penalty as it is the understanding of how to create these models internally at the programming level and the loss of customization and experimentation. Stable-Baselines is an excellent framework for beginners who start experimenting with the world of reinforcement learning. Still, for models where production is required, such as the different tests we have created, it is necessary to create custom architectures that allow us to address the problem in the most optimal way possible. Furthermore, in the case of a bug, because we don't

know what behavior the model is having, such as the result of our Actor-Critic, it is quite complex to analyze and try to solve it only by changing hyper-parameters. One of the serious problems of using this type of frameworks is the depth of the networks, as they tend to use generic networks of few layers for simple problems and that everyone can run the library. So with this type of problems, it is a volatile market that makes agents not able to learn, agents conclude that it is best not to bet, but keep your money in your wallet so they hardly make movements of buying and selling cryptocurrencies, not to even mention that some models do not even try.

- Environment. The environment created is quite simple and has great importance, almost as important as the design of the agents because it is the room in which the agents are going to interact, using the programmed rules of the game. However, it has several problems that I have been able to conclude after experimenting with the different models. First, it has a bottleneck with the transmission of information to the GPU, which means that the GPU practically works at only 30% of its capacity. Second, it has another bottleneck in the access search of the rows to obtain the information. Finally, it has not implemented parallelism, so it is impossible to train with multi-agents which would improve trying different architectures or strategies in a much faster way.
  Another serious problem it has is the feedback of the reward to the agents. For example, that in a bull market like Bitcoin, since its creation, a programmed reward such as valuing the current total equity where is the sum between the value of the cryptocurrencies in FIAT and the own money retained in the portfolio make it too simplistic. This means that in case of interacting an agent with the market what he will do is buy and hold because he knows that the market will continue to rise. That said, it seems pretty smart to bet on the long. Still, we want to make even bigger profits and the possibility of being able to buy even more cryptocurrencies, and with such simplistic formulas, the trader becomes conservative.

Conclusions

This section is divided into two subsections. The conclusion from the first explanation The results are shown in the previous section discusses whether the initial goals have been achieved or not. The last one proposed a new way to solve the problem and set a Continue the work that has already been done.

7.1 Research conclusions

The results have been gratifying.
I have been able to observe in a rewarding way most of the algorithms in-depth and to see the different ideas of how they try to imitate the behavior of a human being. It is true that when I did this project my idea was quite optimistic about the possible results of the different agents but I have been able to know exactly that it is possible to create trading bots. I have been able to verify and analyze that the world of trading is very complicated, and that things are not so simple. Therefore any advertisement on the internet related to making easy money on trading is really a fallacy. Therefore, we must be responsible for investing in cryptocurrencies, knowing what technologies are behind, investing in risks, and that cryptocurrency does not equal profit, as many factors can make the market vary, from governments censoring cryptocurrencies to tweets from influential people like Elon Musk. That does not detract from the fact that I was pleased to see signs that agents were actually learning something and that there is a lot of improvement in this area, but we cannot forget that in the end, the market continuously interacts with humans in it, that it is not as basic as autonomous driving for example where an agent can learn the rules of the road because there the rules really exist. Here everyone tries to learn some method that buys at lows and sells at high, but it is complicated, even a trader on the same move can have an optimistic option and a pessimistic option, but the agents have shown us that buying and doing HODL is an intelligent option that we can consider.

The general objectives:

(a) Data manipulation and study. Accessing the Binance API and analyzing indicators, and adapting the data to the dataset study shows me great satisfaction for possible future personal implementations.

(b) Trading Techniques. It has been possible to simulate an automated and fast trading environment where the different agents have interacted in the purchase and sale of

cryptocurrencies and the study of the calculation of the various indicators.

(c) Environment. We have been able to simulate a simple environment that allows us to adapt to the cryptocurrency market, as we can see in an Exchange at a computational level with our own rules to feed the different learning agents by reinforcement.

So, given these pessimistic results, I wonder, is it not possible to beat the market with reinforcement learning techniques? Generally, cryptocurrencies and any stock market have a severe problem for these models that is votality. Agents in unstable environments, with many variations, do not usually work, as they do not manage to learn to beat the market, they get a lot of noise, and the only thing to do most of the time is to learn a conservative behavior or act randomly, but in the next section you will see some improvements that can be fixed to a large extent.

7.2 Improvements and Future Work

To sum up, I will explain the different ideas that have occurred to me during the project, considering the results to continue improving it.

All in all, changing the terminal state function to a more aggressive function and improving the reward function, most algorithms manage to learn more locally instead of focusing too much on retention. The problem is that the algorithm always starts at step 0 of the data set, so every time it gets to the end state, it starts from the beginning. To optimize this problem, randomness should be added from the beginning to create small pieces of local training to access the entire data set. As a result, a much more complex reward should be made. Currently, the difference between previous total equity and current equity is produced. This provides an incentive to hold. A severe idea would be to compensate the agent when the value of the cryptocurrency goes down, and it doesn't have cryptocurrencies when this happens. It would be a motivation to sell at highs and buy at lows. One could think of more ways to make this function complex, such as being conditional through the volume of the candle, having a delay through the steps to get smaller and smaller rewards if it doesn't interact with the environment.

The second one is quite obvious: creating models with a more significant number of layers to learn different market patterns. Also, modifications to the models such as using 1D Convolutional Layers or using LTSM[44] layers to learn a more extensive sequence of candlesticks.

On the other hand, in order to create a much better environment we should allow parallelism for training and execution of the agents and remove bottlenecks. Also, create a different environment for production where it is connected in real-time to the API, to make real use, with a cyclic queue for x number of rows that allows to calculate the different indicators and not to fill the memory infinitely. On the other hand, environments could be created for the various products, such as Liquidity trading, where you bet on whether an asset is going to go up or down.

Nevertheless, it might also be a good idea to ask more complex questions than trading alone: can the network learn to trade and apply a different style or user-motivated strategy? For example, it is possible that the user may wish to for a more aggressive or more

conservative strategy depending on the user's personality or how the market is doing. Would it be possible to add sentiment analysis of tweets from influencers in these markets, with architectures such as BERT[45], and add it to the dataset as a variable? Would the agent learn to use it?

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. Reinforcement learning i: Introduction, 1998.

[2] Richard Bellman. The theory of dynamic programming, 1954.

[3] Richard Bellman. A survey of applications of markov decision processes, 1993.

[4] M. V. Otterlo and M. Wiering. Markov decision processes: Concepts and algorithms. 2012.

[5] RICHARD S. SUTTON. Learning to predict by the methods of temporal differences, 1988.

[6] P.A. Karnazes and R.D. Bonnell. System identification techniques using the group method of data handling. *IFAC Proceedings Volumes*, 15(4):713–718, 1982. 6th IFAC Symposium on Identification and System Parameter Estimation, Washington USA, 7-11 June.

[7] Yann LeCun. Efficient backpro, 1998.

[8] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[9] Geoffrey E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434, 2007.

[10] CASPER HANSEN. Activation functions explained - gelu, selu, elu, relu and more. 2019.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[12] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[13] iamtrask. Gradient descent. `https://iamtrask.github.io/2015/07/27/python-network-part2/`, 2015.

[14] Francesco Bertoluzzo and Marco Corazza. Testing different reinforcement learning configurations for financial trading: Introduction and applications. *Procedia Economics and Finance*, 3:68–77, 2012. International Conference Emerging Markets Queries in Finance and Business, Petru Maior University of Tîrgu-Mures, ROMANIA, October 24th - 27th, 2012.

[15] Jonathan Sadighian. Deep reinforcement learning in cryptocurrency market making, 2019.

[16] E. S. Ponomarev, I. V. Oseledets, and A. S. Cichocki. Using reinforcement learning in the algorithmic trading problem. *Journal of Communications Technology and Electronics*, 64(12):1450–1457, Dec 2019.

[17] Thibaut Théate and Damien Ernst. An application of deep reinforcement learning to algorithmic trading, 2020.

[18] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance, 2020.

[19] Gang Huang, Xiaohua Zhou, and Qingyang Song. Deep reinforcement learning for portfolio management based on the empirical study of chinese stock market, 2021.

[20] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[21] Antonio Briola, Jeremy Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading, 2021.

[22] Badr Hirchoua, Brahim Ouhbi, and Bouchra Frikh. Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy. *Expert Systems with Applications*, 170:114553, 2021.

[23] S. Nison. *Japanese Candlestick Charting Techniques: A Contemporary Guide to the Ancient Investment Techniques of the Far East*. New York Institute of Finance, 2001.

[24] Binance. Binance official api. `https://binance-docs.github.io/apidocs/spot/en/`, 2021.

[25] Dario Lopez Padial. Ta library. `https://technical-analysis-library-in-python.readthedocs.io/en/latest/`, 2021.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[27] Shangtong Zhang and Richard S. Sutton. A deeper look at experience replay, 2018.

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[29] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[30] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018.

[31] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning(a2c), 2016.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms(ppo), 2017.

[33] Taiyu Zhu, Kenneth Li, Lei Kuang, Pau Herrero, and Pantelis Georgiou. An insulin bolus advisor for type 1 diabetes using deep reinforcement learning. *Sensors (Basel, Switzerland)*, 20, 09 2020.

[34] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning(ddpg), 2019.

[35] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR.

[36] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor(sac), 2018.

[37] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[38] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[39] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[42] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019.

[43] Welcome to python-binance. `https://github.com/sammchardy/python-binance`, 2021.

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[45] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.