POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Thesis

GEOMETRY OF MULTI-AGENTS SIMULATION

Supervisors

Prof. Barbara CAPUTO

Dott. Matteo CAORSI

Candidate

Matteo D'OSPINA

July 2020

Acknowledgements

The following part does not deal with academic topics but is a thank you to those who helped me and have been close to me in these years of study. It is written in Italian because it is addressed to friends and relatives who do not speak English.

Ringrazio la Professoressa Barbara Caputo per aver accettato la mia proposta e di avermi dato la possibilità di partire all'estero per fare la tesi.

Un ringraziamento va a Matteo Caorsi e tutta L2F, per aver scomesso su di me senza conoscermi affondo, per avermi dato la possibilità di fare un'esperienza all'estero come desideravo tanto, per avermi integrato in azienda e avermi fatto sentire uno di loro, sia nei momenti di duro lavoro che nei momenti di festa e di divertimento.

Vorrei ringraziare i miei genitori, mio fratello Giuseppe, mia sorella Chiara, i miei zii, la nonna e tutta la mia famiglia, che non mi ha mai fatto mancare nulla e che mi ha sempre supportato in ogni decisione.

Un rignraziamento speciale va a Loris e ad Andrea, Roberto e Ludovica per avermi accolto in Svizzera e per avermi trattato con un membro della loro famiglia.

Ringrazio i miei coinquilini Giuseppe, Andrea ed Alessio per avermi sopportato in casa in questi anni ed aiutato anche quando meno me lo aspettavo.

Ringrazio Roberto e Via Moretta 7 per tutte le serate indimenticabili passate insieme.

Ringrazio tutti gli amici incontrati sul percorso con cui ho condiviso la maggior parte dei miei ricordi felici, in particolare: Giulio, Michele, Matteo, Stefano e Giuseppe.

Ringrazio tutti gli amici di Neviano, tutti quelli incontrati prima del mio cammino universitario per esserci da sempre e anche a distanza di miglia di km.

Infine un ringraziamento di vero cuore va a Giulia per esserci sempre stata e per avermi supportato davvero in ogni momento, anche in quelli peggiori. Senza di lei ora non sarei qui.

Summary

Understanding the dynamics of stock market, has always been a goal for researchers, because investors need accurate tools to conduct their investments. In this thesis, the stock market will be modeled with a multi-agent reinforcement learning environment. Once trained, this model will allow to collect data accurately and to know everything about transactions, the financial status of investors and macroeconomic variables such as price of shares. These data will be analyzed using Topological Data Analysis (TDA), which will allow to observe the shape of the underlying structure hidden in data. Persistent entropy was used extensively, in order to have a quantitative measure of financial status of investors in an instant of time. Thanks to sliding window, it was possible to extrapolate a signal that represents the dynamics of the stock market over time. We will therefore calculate the Pearson correlation between the entropy signal in homology 1 and the momentum of the price obtaining a positive correlation value.

Table of Contents

1 Introduction 1

2 Related Works 5

3 Reinforcement Learning 9
3.1 Introduction . . . 10
3.2 Examples . . . 10
3.3 Key Concepts and Terminology . . . 11
3.3.1 Introduction . . . 11
3.3.2 States and Observation . . . 12
3.3.3 Action Spaces . . . 12
3.3.4 Policies . . . 13
3.3.5 Episodes . . . 14
3.3.6 Reward and Return . . . 14
3.3.7 Model . . . 15
3.4 The Reinforcement Learning Problem . . . 15
3.4.1 Value Functions . . . 16
3.4.2 Bellman Equations . . . 18
3.5 Kinds of RL Alghorithms . . . 18
3.5.1 Model based properties . . . 18
3.5.2 Model Free properties . . . 20
3.5.3 Proximal Policy Optimization - PPO . . . 20
3.6 Multi Agent Reinforcement Learning - MARL . . . 21
3.6.1 Introduction . . . 21
3.6.2 Environment . . . 22
3.6.3 Policies . . . 22
3.6.4 Examples of MARL applications . . . 22

4 Multi Agent Stock Exchange Environment 25
4.1 Introduction - The Stock Market . . . 25
4.2 Agent . . . 26

4.2.1 Attributes . . . 27
4.2.2 Action Space . . . 27
4.2.3 Observation Space . . . 28
4.2.4 Step . . . 29
4.2.5 Reward function . . . 29
4.3 Multi Agent Environment . . . 30
4.3.1 Introduction . . . 30
4.3.2 Step . . . 30
4.3.3 Price Disturbance . . . 30
4.3.4 Order Book - Matching Engine . . . 32

5 Library used 35
5.1 Introduction . . . 35
5.2 Gym . . . 36
5.2.1 Gym's interface . . . 36
5.3 RLlib . . . 37
5.3.1 RLlib: Scalable Reinforcement Learning . . . 37
5.3.2 Policy . . . 37
5.3.3 RLlib Training APIs . . . 38
5.3.4 Evaluating Trained Policies . . . 39
5.3.5 Tune: Scalable Hyperparameter Tuning . . . 39
5.4 Giotto-tda . . . 39
5.4.1 Simplicial and Simplicial Complex . . . 40
5.4.2 Persistent Homology . . . 42
5.4.3 From data to persistence diagrams . . . 43
5.4.4 Sliding Windows . . . 44
5.4.5 Persistence Entropy . . . 45

6 Experiments and Results 47
6.1 Implementation . . . 47
6.2 First Experiment (without price pertubation) . . . 48
6.2.1 Price always grows, why? . . . 48
6.2.2 Game Theory . . . 49
6.2.3 Game definition . . . 50
6.2.4 Theory hypothesis . . . 50
6.2.5 Nash equilibrium theorem . . . 51
6.2.6 Nash equilibrium example - Prisoner's dilemma . . . 51
6.2.7 Game of transactions between two agents . . . 52
6.3 Last Experiment (with price pertubation) . . . 54
6.3.1 Observed behaviors . . . 54
6.3.2 Topological analyzes . . . 56

6.3.3 Correlations 59
6.3.4 Bootstrapping 61
6.3.5 What does this correlateion mean? 63

7 Conclusion and future works 65

Bibliography 67

Chapter 1

Introduction

As defined by Charles Kirkpatrick, in finance, technical analysis through the study of past market data as price and volume, proposes market study methodologies for forecasting the price trend [1]. For a trader, using a more reliable tool than competitors is an important advantage to make better choices. It is for this reason that the scientific community has always studied this field of research, making it progress by applying the most advanced mathematical and statistical theories of the time. A multitude of research have been carried out trying to apply mathematical and statistical models, for the development of forecast tools. In the last few years researchers have used extensively machine learning and neural networks to build models that, starting from a dataset, were able to build models capable of predicting the trend of financial variables such as price [2][3][4].

While machine learning has already been used extensively by scientific community, Topological Data Analysis (TDA) has still found few applications in this field. TDA is a relatively new field of study in which methods based on topology and geometry of data are used to analyse big and complex collections of data [5]. One of the methods belonging to the TDA group, most used to study large and complex data systems is persistent entropy, which will be exposed and used in this thesis. Persistent entropy method is particularly appreciated because is able to describe the global dynamics of a complex system [6]. One of the main works is that of Marian Gidea and Yuri Katz [7], where TDA is used to detect early warning signals of imminent market crashes. For a detailed introduction to topology topics, that will be covered in the following chapters, it is advisable to consult the paper by Chazal and Michel [8].

The aim of this thesis is to try to understand what happens behind the price chart of a stock, understanding the dynamics of the exchanges that take place and transform them into a signal that could be related to the price. Being more aware of what is happening in the market, that is what and how traders decide to buy or sell, could be the key to recognizing patterns that were unknown and to use them

to build better forecasting tools.

In order to study the dynamics that regulate a market, it was decided to build a model that allows to simulate and to collect data about the exchanges and financial situation of agents. For this reason a Multi Agent Reinforcement Learning (MARL) (chapter 4) was built, trained through Reinforcement Learning Algorithms as PPO (chapter 3.5.3). In this environment, an agent is our model of a trader, so when we following will say agent, we are referring to a trader who owns shares and money and is able to act on the market by buying or selling shares with another agent. The environment allows any number of agents to participate in the market. However, the limit is computational, in fact increasing the number of agents increases the time and ram memory required by the process. The choice to build such an environment allowed us to build a realistic but simplified model of a real market, which to make simulations and therefore to generate an indefinite amount of data. Reinforcement learning has already been used in the past to simulate market dynamics. For example, in the work of Bao and Liu [9] it has been used in order to find optimal liquidation strategies. Another important work is that of Yagna Patel [10] where a model has been built that exploits reinforcement learning to address the problem of optimizing market making. In this case, the agents are not placed in any particular situation as in the papers mentioned above, but will face normal auctions for a stock market. Agents will be free to develop any type of strategy in order to increase their net worth. Reading this article [11] by Adam King was of fundamental importance for the construction of this model, helping a lot on how to build the model.

As already mentioned, to analyze the data generated by the environment it was decided to use TDA, which as shown in [12] is an excellent tool for giving a geometric representation of complex data sets. As suggested by Marian Gidea [7], thanks to TDA and in particular to persistent entropy it is possible to detect and quantify topological patterns that appear in multidimensional time series. In this case, as will be seen in the last chapter, the agents will be characterized based on their net worth and based on net worth value they will be represented in space. As explained later 5.4.5, persistent entropy is used to detect the presence of topological structures in space and to provide a quantitative measure, that represents the topological state of the environment at a given moment. In this way it is possible to transform the topological information in a signal, that, will then be related with a macro variable, such as the momentum of price, in order to study how the topological state influences the price trend.

In order to find a relationship between the price and the topological signal, a thousand simulations of market sessions were held in which ten traders participated and the data were collected. After carrying out the topological analyzes and extrapolating the topological signal, the Pearson correlation will be calculated for each simulation between the topological signal and the moment of the price.

In order to evaluate the quality of the measurements shown, the Bootstrapping technique will be used.

Hypotheses will be formulated on the meaning of the correlation between the two time series and some ways to deepen the research will be suggested in the last chapter.

Chapter 2

Related Works

As stated by J. Gong and S. Sun, « Stock market forecasting has always been a highly appreciated research field due to its volatile, complex and regular changing nature, making it difficult for reliable predictions » [2]. In fact, it has been studied by many researchers from different fields, who have tried to understand its nature by applying the state of the art of their study disciplines. However, it is possible to divide research into two main areas of study. The first area, called behavioral finance, analyzes the behavior of investors from a psychological, social and emotional point of view [13][14] and trying to understand how these factors influence their ability to make decisions. Second area of study is more technical and concerns the application of mathematical and statistical models for forecasting of the price trend. In the last few years, the interest in machine learning has increased and as result it is possible to find papers in abundance that apply machine learning concepts to the financial market. For example, Siew and Nordin used regression techniques to predict stock price trend [3][2][15]. However, these two areas are not strictly separate, but are often used together by researchers. Sentiment analysis, which seeks to interpret and classify the emotions of investors, is often used in combination with machine learning algorithm in order to do more accurate prediction. An example is the article "Stock Prediction Using Twitter Sentiment Analysis" by Anshul Mittal and Arpit Goel [16].

A field of research that has grown significantly in recent years is reinforcement learning and this too was used in the research field of stock market forecasting. It is important to mention the article by Adam King [11], where an agent is trained to make the best choice between buying or selling stocks basing on market trends of a real time series. This project is inspired by Adam King's article, but has been extended by adapting it to a multi-agent model. In fact the complexity of the environment in the financial markets, has encouraged the use of modeling by multi-agent platforms and particularly in the case of the stock market [17]. It is already possible to find many papers that use this approach, which allows to find

effective strategies to increase the benefit of investors in different situations. Agent models based on reinforcement learning, are very useful because once modeled and trained the environment allows you to simulate the behavior of investors in different conditions decided by researchers and to study how these behaviors affect the market. In fact, the price trend is also determined by the decisions taken by investors. This offer the opportunity to study the dynamic, heterogeneous and adaptive behavior of market players and its impact on market dynamics. In the paper "Multi-agent modeling and simulation of a stock market" addresses the problem of modeling the market using a multi-agent model, making some choices similar to those undertaken in this thesis, especially from the point of view of modeling [17]. As a result, researchers can reproduce the situation of their interest to find out which are the best strategies to follow. An example is the paper by W. Bao and X. Y. Liu, where they apply reinforcement learning for Liquidation Strategy Analysis [9]. There are already numerous published papers that use reinforcement learning for price prediction [18][19] but another paper where reinforcement learning has been used in a similar way to this thesis, that is to simulate the stock market in an artificial way and to collect data is that of Rutkauskas and Ramanauskas where reinforcemen learning algorithm combined with some evolutionary selection mechanism [20]. To understand the theory of reinforcement learning it is convenient to refer to the book "Reinforcement Learning: An Introduction" written by Richard S. Sutton, Andrew G. Barto [21] which offers a detailed explanation of the theory of reinforcement learning. For a more concise explanation, the Open AI Spinning Up site is recommended [22].

As defined by Chazal and Michel, « Topological Data Analysis (TDA) is a recent and fast growing eld providing a set of new topological and geometric tools to infer relevant features for possibly complex data » [8]. TDA was also used by researchers to study the stock market, where the main target in Topological Data Analysis (TDA) is to find the shape or the underlying structure of shapes in data [23][12]. TDA can also be used in conjunction with machine learning algorithms. Currently, in the field of the stock market, TDA has been used mainly to detect stock crashes, for example D. Basu and T. Li from their work they deduced that incorporating TDA leads to substantial improvements in timely detecting the onset of a sharp market decline [15]. Also in the article "Detect-ing stock market crashes with topological data analysis" [24] the results suggested that the periods of high volatility preceding a crash produces geometric signatures that can be more robustly detected using topological data analysis.

As you will read in the following chapters, topological analyzes will be carried out on a multi agent model of the stock market based on reinforcement learning algorithms. The two research fields mentioned above will therefore be put together to study the stock market in a totally new way than in the past. This new approach will allow to bring together the positive aspects of reinforcement learning and TDA.

In fact, it will be possible to model an environment with great flexibility, adapting it to the context you want to analyze. Being a multi agent model, it will be possible to vary the number of investors in the market. Thanks to TDA, it will be possible to analyze the agents behaviors from a geometric point of view, going to detect possible collaborations or indeed competitions between them.

Chapter 3

Reinforcement Learning

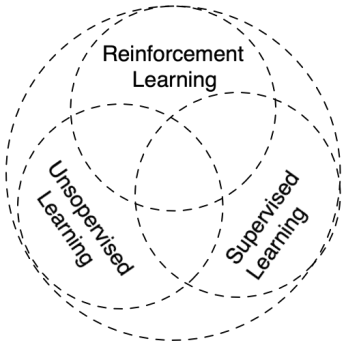

Figure 3.1: View of the relationship between Supervised Learning, Reinforcement Learning and Unsupervised Learning. Some fields of study are located at the intersection of one field of research and another. Deep reinforcement learning for example arises from the integration of neural networks into reinforcement learning models.

3.1 Introduction

Reinforcement Learning (RL) is a branch of artificial intelligence that trains algorithms using a system of reward and punishment. The problem involve learning the best action to take in certain situations so as to maximize a numerical reward signal. As stated by Sutton and Barto in one of the most complete books on Reinforcement Learning:

> Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards [21].

In order to collect a large dataset, we had to build a model that simulated the dynamics of a real market and allowed us to collect all data. We have chosen to build a MARL model that exploits reinforcement learning for several reasons. Firstly, during the training phase, reinforcement learning allows agents to explore different strategies, allowing them to learn cooperation or competition strategies that are very interesting to observe in following analyzes. Secondly, a MARL model is extremely flexible and has allowed us to model the agents and the environment in which the agents operate, allowing us to vary the number of agents easily.

3.2 Examples

A good way to get to know the philosophy behind Reinforcement Learning is to consider some of the examples and possible applications that have guided its development.

- Games - Reinforcement learning has literally become famous in this field of scientific research. It is often used to find winning strategies that a human player would not be able to find, even with a high number of moves. Very famous is the alpha go victory over the go world champion Lee Sedol in 2016 [25]. Another common application is Atari games like Pong, whose input is raw pixels and whose output is a value function estimating future rewards [26].

- Traffic Light Control - Reinforcement Learning is used to scheduling of traffic signals in a vehicular networks, to obtain an efficient traffic signal control policy. In order to minimising the average delay, congestion and likelihood of intersection cross-blocking [27].

- Robotics - Reinforcement Learning enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution to a problem, in Reinforcement Learning, a robot can explore the solutions to find and optimal solution [28].
- Chemistry - chooses new experimental conditions to improve the reaction outcome [29] and to design new molecules with specific desired properties. This is especially important in material design or drug screening. Currently, this process is expensive in terms of time and cost: it can take years and cost millions of dollars to find a new drug. The goal of this study is to partially automate this process through Reinforcement Learning [30].

3.3 Key Concepts and Terminology

3.3.1 Introduction

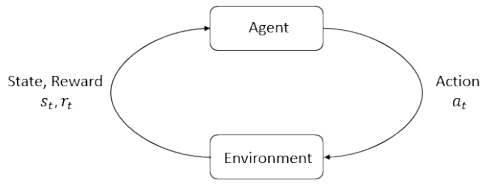

Figure 3.2: A single agent reinforcement learning model. The agent performs an action on the environment and the environment returns to the agent the new state and the reward. As in a loop the agent continues to perform actions until a termination condition such as a maximum number of actions is reached.

In a single agent RL system there are two actors, the agent and the environment. The agent is the active part of the system, which carries out the actions on the environment, receiving from it a reward value which identifies how good the action was. The sum of all rewards obtained by the agent is called *return*. Agent's goal is to maximize the final *return*. The agent also collects observations that represent part of the state of the environment in order to use this information to choose the next action. Below, the set of actions taken by agent and the states of the environment will be defined as *game*. In the training phase, as in a loop, the agent continues to perform actions on the environment for a finite number of steps or

until the agent reaches a goal previously imposed as a minimum return threshold. After the training phase, the agent has collected enough experience to achieve his goal in many of the situations in which the environment can put him.

3.3.2 States and Observation

A state s contains all the information to describe the current situation of the environment. Instead, an observation o is a partial description of a state, which may omit information. Based on the observations it is possible to define as *fully observed* an environment in which the agent is able to observe the complete state of the environment. Instead we say that the environment is *fully observed*. When the agent can only see a partial observation, we say that the environment is *partially observed* [22].

3.3.3 Action Spaces

Figure 3.3: Atari game: Pong. Very simple game to model with reinforcement learning. The arrows show the only two possible actions. At each step the agent must choose between two possible actions: up and down.

In the simplest cases, the actions that an agent can do are few and well defined. For example, in the game of Pong (figure 3.3) an agent must decide to move his racket either up or down according to the situation that is presented to him. A slightly more complex case is that of the game of Go, where it is necessary to choose where to insert a new stone in a set of $2{,}08 \cdot 10^{170}$ positions, that is a finite number of possible actions but much more numerous. The situations just described are cases where the action space is discrete and finite. There are other more complex cases, where the action space cannot be simplified as a discrete space, but a continuous space must necessarily be used, for example if it is necessary to specify a quantity.

We can formalize an action as a tuple of n values $(s_1, ..., s_n)$, where each value has its own domain and expresses information related to the action. For example, in the game of chess it is necessary to indicate which piece you want to move and the target box. So the tuple will be (s_1, s_2) where s_1 will indicate the piece to be moved and s_2 the target box. But the values are not always discrete, in more complex cases they can also be continuous. The distinction between continuous space and discrete space is very important, because some families of algorithms are usable in one case and not in the other.

3.3.4 Policies

Policy is the core of the RL problem. As defined in [31] a policy is an action plan that the agent follows to determine the next action based on the current state. The policy can be of two types: *deterministic* or *stochastic*.

- *Deterministic policy*: Is a function of the form $a = \pi(s)$ from the set of states of the environment, S, to the set of actions, A. For example, in a grid world, the set of states of the environment, S, is composed of each cell of the grid, and the set of actions, A, is composed of the actions *left*, *right*, *up* and *down*. Given a state $s \in S$, $\pi(s)$ is, with probability 1, always the same action (e.g. *up*), unless the policy changes.

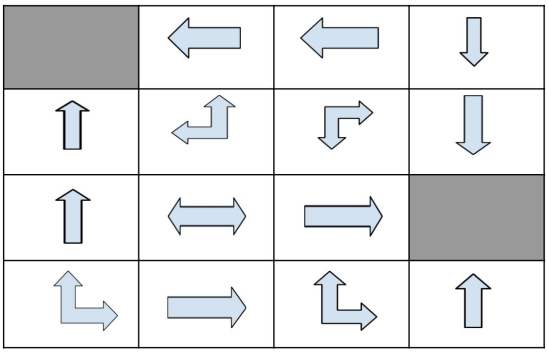

Figure 3.4: Graphical representation of a deterministic policy, where the gray boxes indicate the exit and the arrows the actions indicated by the policy. The arrows indicate the most convenient move to reach the exit as quickly as possible. The boxes with two arrows indicate two possible moves that have the same utility.

- *Stochastic policy*: A stochastic policy is often represented as a family of conditional probability distributions, $\pi_s(A|S) = P[A_t = a|S_t = s, \theta]$, where each state is associated with a probabilistic distribution function parameterized by θ.

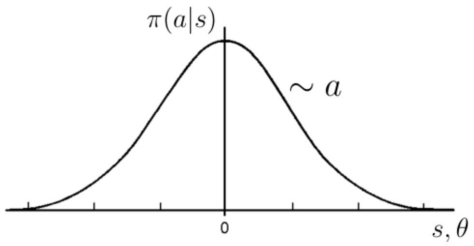

Figure 3.5: Normal distribution, normally centered in zero, describes the probability distribution of taking a given action by defining s and θ. Where the parameter θ is a parameter of the normal function, which is used to define the function while s characterizes the state, this function returns the probability of taking an action.

3.3.5 Episodes

Episodes are also called trajectories τ and are defined by the sequence of states taken by the environment and by the actions taken by the agent, according to its policy, since the beginning of the history. $\tau = (s_0, a_0, s_1, a_1, ...)$. s_0 is randomly chosen by the **start-state distribution**, denoted by ρ_0 [32].

3.3.6 Reward and Return

Reward is a value that the agent gets from the environment after taking an action on it. Reward function $r_t = R(s_t, a_t, s_{t+1})$ maps the current state, the action taken and the consecutive status after the action taken in a value, the reward. It is therefore extremely important to define this function in order to obtain positive rewards when the action leads the environment to a desired state and negative rewards when the actions are not considered useful to achieve the objective. The goal of the agent is to maximize cumulative function reward over an episode. We indicate the function with R(τ) which can differ in the following ways:

- **finite-horizon undiscounted return**, which is just the sum of rewards obtained in a fixed window of steps [33]:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + ... = \sum_{k=0}^{\infty} r_{t+k+1} \tag{3.1}$$

- **infinite-horizon discounted return**, which is the sum of all rewards ever obtained by the agent [33]:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.2}$$

  Where $\gamma \in (0,1)$ is a discount factor and is used to give different weights based on how far ahead we are in the episode. The two benefits of defining return this way is that the return is well defined for infinite series, and that it gives a greater weight to earlier rewards, meaning that we care more about imminent rewards and less about rewards we will receive further in the future. Intuitively, cash immediately could be better than cash later [34].

3.3.7 Model

In some RL systems is used the model of the environment. The model is a mathematical object used to simulate the behavior of the environment. The models are used for planning the actions to be taken, without changing the state of the environment, considering possible future situations before they are actually experienced [21].

3.4 The Reinforcement Learning Problem

The goal in RL is to select a policy which maximizes the expected return when the agent acts according to it. To calculate the expected return value we will have to define probability distributions over trajectories. Let's suppose that both, the environment transitions and the policy are stochastic. In this case, the probability of a T step trajectory is [22]:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{3.3}$$

The previous formula consists of the multiplication of three parts. The first $\rho_0(s_0)$ is the probability of being in the state s_0 at the beginning of the trajectory. The second is the probability that a certain sequence of states can follow one another. In

fact, it is possible that a certain trajectory is impossible, therefore it has probability 0, if a state cannot follow the previous one. Third is the probability that a certain trajectory will be implemented following a certain strategy π. In fact, if the agent does not consider it advantageous to carry out a certain action in a certain state then the probability will be zero.
The expected return (for whichever measure), denoted by $J(\pi)$, is then:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) \tag{3.4}$$

The central optimization problem in Reinforcement Learning can then be expressed by:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3.5}$$

3.4.1 Value Functions

Value function is a function that, defined a policy, given a state, returns a value that represents the quality of a state if the agent were acted following the defined policy. For example, in the figure 3.6, you can see how the values decrease as you move away from the end box. The concept of value function is really important, in fact is used, in different ways, in almost every Reinforcement Learning algorithm [35].

As reported on the *Open AI Spinning Up* website [35], there are four types of Value Functions, listed below:

- The On-Policy Value Function, $V^{\pi}(s)$, which gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)|s_0 = s] \tag{3.6}$$

- The On-Policy Action-Value Function, $Q^{\pi}(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s, a) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)|s_0 = s, a_0 = a] \tag{3.7}$$

- The Optimal Value Function, $V^*(s)$, which gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^*(s) = \max_{\pi} \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)|s_0 = s] \tag{3.8}$$

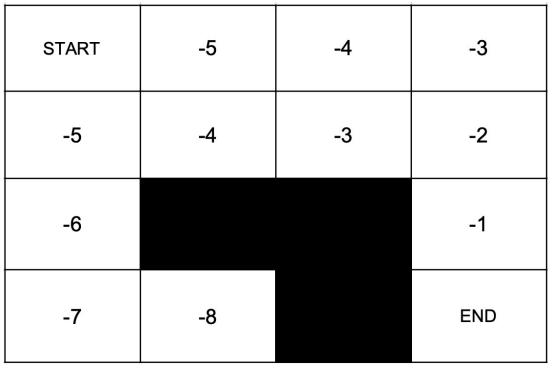

Figure 3.6: Graphical representation of an example of Value Function. Agent 's goal is to go from start to exit. In the boxes there is a value indicating how convenient it is to be in that box. The boxes closest to the exit have higher values than the more distant ones or the boxes that are in a wrong path.

- The Optimal Action-Value Function, $Q^*(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:

$$Q^*(s, a) = \max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}}[R(\tau)|s_0 = s, a_0 = a] \tag{3.9}$$

Knowing the optimal value function and its definition, it is very easy to deduce the best action to take following the optimal policy. In fact starting in s, the best action is the following:

$$a^*(s) = \arg\max_{a} Q^*(s, a) \tag{3.10}$$

3.4.2 Bellman Equations

The Bellman equations are ubiquitous in RL and are necessary to understand how RL algorithms work. The importance of the Bellman equations is that they let us express values of states as values of other states. This means that if we know the value of s_{t+1}, we can very easily calculate the value of s_t. This opens a lot of doors for iterative approaches for calculating the value for each state, since if we know the value of the next state, we can know the value of the current state [34]. The Bellman equation for a Policy π is:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{SS'}[\mathcal{R}^{a}_{SS'} +^{\pi} (s')] \tag{3.11}$$

where P is the transition probability. If we start at state s and take action a we end up in state s' with probability $\mathcal{P}^{a}_{ss'}$:

$$\mathcal{P}^{a}_{SS'} = Pr(s_{t+1} = s' | s_t = s, a_t = a) \tag{3.12}$$

and $\mathcal{R}^{a}_{ss'}$ is another way of writing the expected (or mean) reward that we receive when starting in state s, taking action a, and moving into state s'.

$$\mathcal{R}^{a}_{ss'} = \mathbb{E}[r_{t+1} | s_t = s, s_{t+1} = s', a_t = a] \tag{3.13}$$

Instead, the Bellman equation for the action value function is:

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{SS'}[\mathcal{R}^{a}_{SS'} + \sum_{a'} \pi(s',a') Q^{\pi}(s',a')] \tag{3.14}$$

3.5 Kinds of RL Alghorithms

The main property that differentiates Reinforcement Learning algorithms is the presence or absence of a model. Algorithms that use a model are called model based, those that don't use it are called model free. [36]

3.5.1 Model based properties

The algorithms that can leverage a model of the environment are often much more efficient than those that can't. An example is AlphaZero which uses a model-based algorithm. These algorithms are much more performant, because they have the possibility of making better choices, because they can test their choices on the model to see what reactions they will have in the future. To do this, however, it is necessary to have a perfect model of the environment. If this is absent it is necessary to build the model from experience and this is not always feasible. The main problem is that there is a risk that agents learn to perform well on the model learned and not on the real environment. « Model-learning is fundamentally hard, so even intense effort—being willing to throw lots of time and compute at it can fail to pay off » [36].

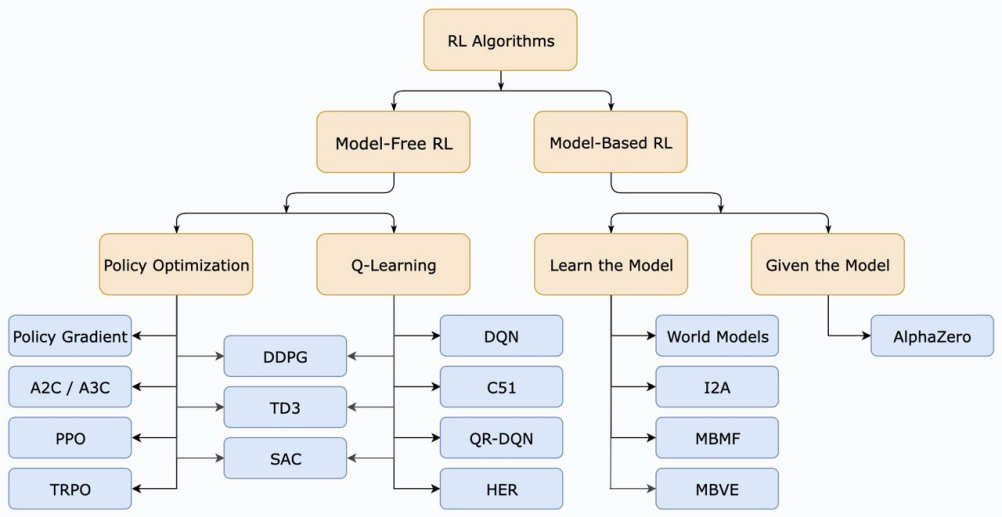

Figure 3.7: Graphical representation of taxonomy of algorithms in modern RL [37]. In this figure, the division between model-free and model-based is very clear. In this thesis we will use the PPO algorithm that descends from the policy optimization algorithms which is of the model-free type.

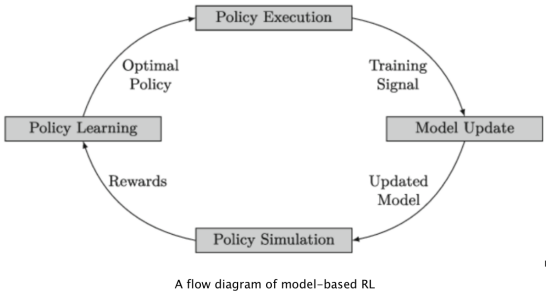

Figure 3.8: An example of a workflow of a model based algorithm [38]. As you can see in the figure, there is a model which is updated at each step and which simulates the policy.

3.5.2 Model Free properties

Model free algorithms mainly use two different techniques: Q-Learning and Policy Optimization. As part of this project, an algorithm called PPO was used, which is part of the Policy Optimization algorithms. In order to understand the difference between the two techniques, it is necessary to define the difference between *off-policy* and *on-policy*. As excellently defined by David Poole and Alan Mackworth: « An off-policy learner learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner. An on-policy learner learns the value of the policy being carried out by the agent, including the exploration steps » [39]. The **Q-Learning** algorithms are always off-policy algorithms, where we try to find the action that maximizes the $Q_\theta(s, a)$.

$$a(s) = \arg\max_a Q_\theta(s, a) \tag{3.15}$$

Policy Optimization algorithms are always on-policy algorithms, where as reported in [40] « Methods in this family represent a policy explicitly as $\pi_\theta(a|s)$. They optimize the parameters θ either directly by gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by maximizing local approximations of $J(\pi_\theta)$ ».

3.5.3 Proximal Policy Optimization - PPO

In recent years, researchers have focused mainly on the study of algorithms belonging to the family of policy optimization. What they were looking for was an algorithm that could overcome the problems of existing algorithms. In fact, Q-learning fails in many simple problems and is difficult to understand. Vanilla policy gradient methods have poor data effiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that includes noise [41]. These needs led to the formalization of a new family of algorithms called Proximal Policy Optimization (PPO). The basic idea of PPO is that one performs not just one but multiple minibatch gradient steps with the experience of each iteration [42]. As stated by Schulman in his research, based on experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing. He shows that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time [41].

3.6 Multi Agent Reinforcement Learning - MARL

3.6.1 Introduction

Multi Agent systems are an approach to building complex distributed applications. A multi agent system consists of a population of autonomous entities (agents) situated in a shared structured entity (the environment)[43]. The agents can have cooperative, competitive, or mixed behaviour in the system [44].

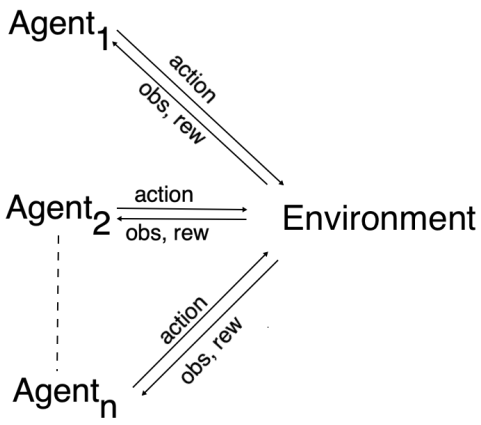

Figure 3.9: Illustrative figure of a Multi Agent System. In the figure there are n agents who carry out actions in the environment and receive rewards and observations. The model is very simple, it extends the single agent model by adding other agents.

3.6.2 Environment

Intuitively, the environment can be seen as a place where agents interact with domain objects and resources, and with other agents. As reported in [45], important responsibilities of the environment are as follows.

- Structuring: The environment is first of all a shared common 'space'
- Managing resources and services: The environment acts as a container for resources and services
- Maintaining environmental processes: Besides the activity of the agents, the environment can assign particular activities to resources as well.
- Enabling communication: The environment defines concrete means for agents to communicate. The most used scheme is a message-passing style from one agent to the other.
- Ruling the multiagent system: The environment can define different types of rules or laws on all entities in the multiagent system.
- Providing observability: The environment must be observable, agents must have access to resources and services. Agents may even be able to observe the actions of other agents; to be accessed. Resources are objects with a specific state.

3.6.3 Policies

Compared to a single agent scenario where the agent has only one policy, in multiple agent scenarios it is possible to model agents with different criteria, in order to make them react to events differently. For example, if we wanted to model traffic, we would have to think differently about cars than bicycles, for this purpose it is logical to assign different policies to agents who have different roles in the environment.

3.6.4 Examples of MARL applications

MARL systems allow to model different aspects of an environment and agents with different characteristics. For this reason they come much closer to a real scenarios than a single agent system. MARL systems have already been applied to different contexts by scientific research, some of these have already been documented and reported below [44]:

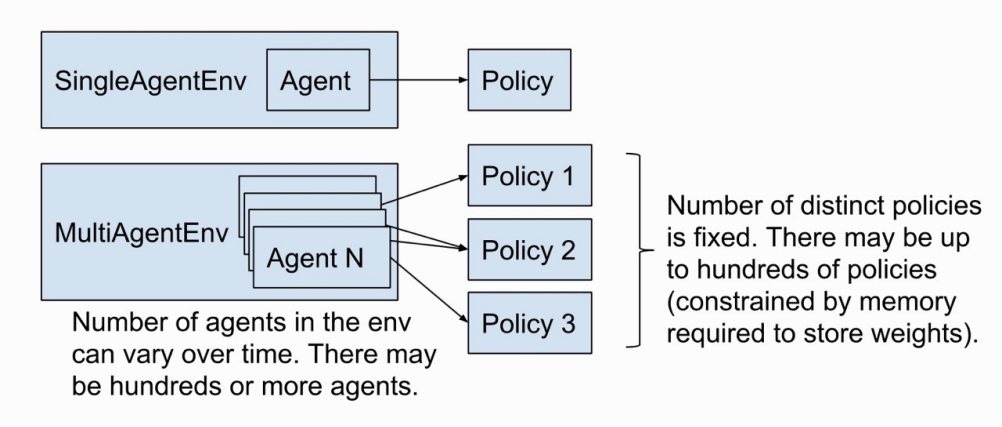

Figure 3.10: The figure represents the difference between a SingleAgentEnv and a MultiAgentEnv. In particular, multiple agents are present in a MultiAgentEnv. Each agent can either have their own policy or share the policy with another agent [46].

- Online Distributed Resource Allocation: Applying MARL to come up with effective resource allocation in a network of computing [47].
- Cellular Network Optimisation: Applying MARL in LTE networks, guide base stations to maximise mobile service quality [48].
- Smart Grid Optimisation: Applying MARL to control power flow in an electrical power grid with optimum efficiency [49].
- Smart Cross Light: Applying MARL to control traffic lights to minimise wait time for each car in a city, making them more adaptable based estimates of expected wait time [50].

Chapter 4

Multi Agent Stock Exchange Environment

4.1 Introduction - The Stock Market

«A stock market, equity market or share market is the aggregation of buyers and sellers of stocks (also called shares), which represent ownership claims on businesses» [51].

The concept behind how the stock market works is pretty simple. Operating much like an auction house, the stock market enables buyers and sellers to negotiate prices and make trades.

Buyers offer a *bid* or the highest amount they're willing to pay, which is usually lower than the amount sellers *ask* for in exchange. This difference is called the bid-ask spread [52].

To create a model of a stock market close to reality, we have chosen to use a MARL system, because it allows the interaction of multiple agents, just as happens in real stock markets. The environment allows us to simulate auctions where it is possible to choose all the starting parameters. It is therefore possible to choose the number of agents participating in the auctions, the number of auctions in a *game* to which they must participate and also the starting conditions with which the agents will have to deal.

Great attention was paid to respect the main characteristics and methods that regulate the environment in order to ensure a good consistency with the real world.

Once all the parameters have been set and the environment has been trained, agents can play a large number of games and collect the data. What we want to observe is the presence of patterns that are repeated during the game and how they influence the price trend. By pattern we mean precise strategies adopted by agents to ensure the highest possible reward at the end of the game. Agents could

for example learn when the share price will drop and adopt buffering strategies. In literature there are, for example, models such as that of Lotka-Volterra [53] or predator prey model, where two populations of different classes follow precise models of growth or decrease in the size of the population.

4.2 Agent

An agent is a model of a stock trader. An attempt has been made to represent the conditions as close as possible to reality, but this is made impossible by the computational limits and the impossibility of representing all the factors considered in a trader's decisions, like personal and emotional characteristics. In the real stock market agents can interact with each other by buying and selling different stocks, but in this model it was chosen to analyze the behavior of agents focusing on a single stock, therefore limiting the number of actions that they can perform, but leaving the possibility of extending the project also implementing this functionality hereafter.

An agent is characterized by its attributes, exposed in the next chapter 4.2.1. Attributes are values that define some properties of the agent such as balance and the number of shares held, which, based on the current share price, will shape its ability to act on the market. It's very important to define it coherently with the scenario that you want to study.

In this case the dynamic we want to analyze is that of buying and selling shares among agents. It is therefore important to immediately clarify the way with which a transaction takes place and what are the degrees of freedom with which an agent can act. In fact, an agent can only make an offer to sell or buy shares. The transaction occurs only when a purchase offer is consistent with a sale offer. It's called *bid* an offer that proposes to buy shares, while *ask* an offer that proposes to sell shares. The transaction will only occur when an agent has made a bid with a higher price than an ask, that is, when an agent is willing to sell at a lower price than another agent wants to buy. The transaction price will be the ask price and the current price will also be updated with the current transaction price.

4.2.1 Attributes

All agents are modeled in the same way, have the same initial skills and compete in the market with the same initial resources. In particular, the attributes that define the financial state, initialized with standard values, are:

1. *nshares* indicates the number of shares owned by the agent and it is always a positive integer.
2. *balance* indicates the amount of cash in £ owned by the agent.
3. *net worh* indicates the value owned by the agent. It consists of the sum of the balance and the number of shares for the price at that precise moment.

The value with which the balance and the number of shares are initialized is very important, because it defines the context in which the agents will act. Obviously if the value of the initial balance is very low compared to the price of the initial shares, the agents will find themselves in a situation of scarcity, on the contrary, if they have a lot of balance available, they will find themselves in a situation of abundance, this will lead them to develop different strategies, coherent with the environment. It is therefore very important to define in which situation the agents are made to *live*. In the case of our analyzes, it was decided to initialize the balance at one thousand and the number of shares held at one hundred, the agents will therefore start playing in a situation of abundance.

4.2.2 Action Space

An agent can only do 3 actions: *buy*, *sell* and *wait*. if it decides to buy it fixes a *bid*, if decides to sell it fixes an *ask*.
In order to buy or sell, the agent must be able to indicate the type of action, The amount of shares he decided to sell or the amount of money he decided to invest, and the price chosen. The action is modeled through a tuple of three continue values thus constructed: [a, b, c]:

- a $\in$ [0, 3] indicate the type of the action.
  - a $\in$ [0, 1] the action chose is buy;
  - a $\in$ (1, 2] the action chose is sell;
  - a $\in$ (2, 3] the action chose is wait;
- b $\in$ [0, 0.5]
  - if a $\in$ [0, 1] then b indicates the percentage of the balance that the agent intends to invest in the purchase of new shares;

  - if a ∈ (1, 2] then b indicates the percentage of the owned shares that the agent intends to sell;

- c ∈ [-1, 1] it indicates the percentage change in price from the current price

  - if a ∈ [0, 1] the chosen price indicates the maximum price that an agent is willing to pay to buy a share;
  - if a ∈ (1, 2] the chosen price indicates the minimum price that an investor is willing to accept if he decides to sell his own shares;

4.2.3 Observation Space

Observation space is defined as a continuous set of normalized values bounded between 0 and 1, with the shape *(6, lookback-window-size + 1)*. For each agent and for each time step in the window, we will observe the following value:

- *the current price*: It is the current stock price;
- *balance*: It is the amount of cash held by the agent;
- *shares held*: It is the number of shares owned by the agent and it is always an integer value;
- *total shares sold*: It is the total number of shares owned since the beginning of the epoch;
- *maximum net worth value reached*: The net worth is the sum of the balance and the value of the shares owned by the agent. This value stores the maximum value reached by net worth of the agent;

It has been chosen to use these attributes as observations, because they can give a representation of the agents' current financial situation and history. By default lookback-window is set to 10, so for each step the agent will observe the values of the above variables in the previous ten days. In this way, agents will be able to observe the trend over time of the environment, returning a behavior more similar to that of a real agent, in fact a trader always observes the price trend over time. Based on past experience, agents will be able to recognize patterns and develop strategies.

4.2.4 Step

A step consists of three phases:

- The first one is decided by the agents, in which they can perform their actions and therefore make their bets, which will be collected in an Order Book;
- The second in which, based on the bets made, the environment performs the matching engine and solves the Order Book;
- The third in which the outcome of the transactions is finally distributed among the agents and the corresponding rewards are calculated.

4.2.5 Reward function

The reward function is the function according to which the environment returns feedback of the goodness of the action taken by the agent. It is important to indicate to agents which behaviors are considered positive and which are less positive or negative. The last thing to consider is reward function that is:

$$R = (2 \cdot networth - initialnetworth - pastnetworth) \cdot \gamma \tag{4.1}$$

$$\gamma = \frac{currentstep}{MAXSTEPS} \tag{4.2}$$

The reward function focuses on the value of net worth. Recall that the value of the net worth is given by the sum between the balances and the value of the shares held. Was chosen to use net worth as core of reward function because it fully represents an agent's financial status. The reward value is therefore positive if the current net worth is greater than the initial value and the previous value. The purpose of γ is to allow to explore during the first steps and to be more cautious during the last steps. This function wants to incentives choice that brings profits during a long period of time. In fact, the agents will be free to make risky moves at the beginning of a game precisely because γ will be a very small value and therefore any negative reward will not be so decisive. During the first steps the agent will be able to explore more risky moves. While at the end of the game gamma will be close to one and therefore it is better not to take risks. In order to better understand the behavior of the agents that will be seen in the following chapters, it is necessary to note that the value of the net worth is directly proportional to the value of the price of the shares. When the share price increases, the net worth also increases.

4.3 Multi Agent Environment

4.3.1 Introduction

In this section, will be explained how agents interact with each other and how transactions actually take place. This role is entrusted to the MultiAgentEnv class which acts like a real Stock Market, receiving bids and asks and making transactions. At the beginning of a trading session the market reset all agent parameters and chooses randomly an open price between a determined range of values. The price is different for each epoch in order to always provide different scenarios during the training phase.

4.3.2 Step

The step simulates a trading session, with a first phase in which the agent make their bet by adding a bid or ask to the book and a second session in which the book is resolved.

During the first phase, the *bet-an-offer-wrapper* method of the agent, is called for each agent in order to give everyone the opportunity to bet based on current environmental conditions such as the share price.

During the second phase, stock market, as we will see in the next section, will take care of solving the book of bets, after which it will call the step method in which it will pass the transactions that took place, in order to update the portfolio of agents that get involved in a transaction.

4.3.3 Price Disturbance

In economics and in the stock market, there are external factors, called exogenous variables that are capable of disturbing the price of shares. There are extremely negative events such as wars, crises and pandemics that can bring the share price down, but also positive situations that can quickly raise the share price. To have a similar effect, a function called *random walk* was used. Random Walk is a mathematical function that takes a random value at each step, given by the sum of the previous value and the value assumed by a stochastic function. As his name suggests, his behavior is very similar to that of a walk in which each step is done randomly [54] [55].

In this case the function used has this form:

$$\alpha = \alpha + \mathcal{N}(0,0.1) \tag{4.3}$$

$$price = price + price \cdot \alpha \tag{4.4}$$

The reason why we chose to use the random walk was his ability to maintain his value for some time steps. We will therefore never have rapid positive or negative changes. If alpha becomes positive, it may turn negative but it is likely that it will remain positive for some time. This type of behavior is visible in 4.1

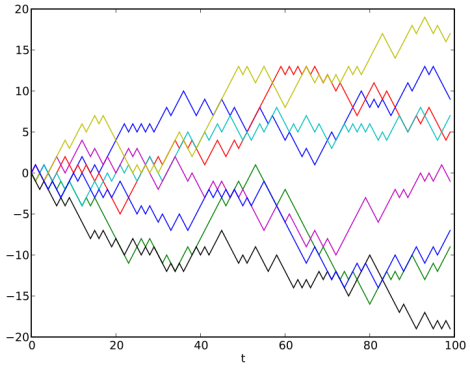

Figure 4.1: Example of eight Random Walks starting from 0 [56]. From the figure it is easy to understand that each step takes the previous one as a starting point, making a random choice both on the direction and on the amplitude of the next step.

Alpha will not have the opportunity to act on the price at each step, but can act, based on a random choice only in certain steps. In the analyzed case, it was decided to make alpha act only every 100 or 150 steps, in order to not make these phenomena permanent but related only to specific moments and to understand the strategies that agents put in place to respond to these events.

4.3.4 Order Book - Matching Engine

In the first phase of a step, the agents set their own bet, a bid or an ask, which are collected within a structure which is called order book. At this point it is important to specify that agents cannot decide with which other agent to carry out the transaction, but they will only be able to decide the quantity of shares they want to buy or sell and the price at which they are willing to buy or sell. Once all the bets have been collected, it will be the order matching system, an automatic algorithm to match asks and bids and to carry out transactions.

The first step that the order matching system carries out is to sort the bids by decreasing price and ask by increasing price obtaining a situation like the one in Figure 4.2.

The purpose of the sort is to satisfy first who is willing to spend more than others

	BIDS	
	Quantity	Price
A0	10	2.22
A3	5	2.11
A8	10	2.08
A5	9	2.05

	ASKS	
	Quantity	Price
A1	5	1.98
A4	7	2.05
A2	10	2.09
A6	2	2.11

Figure 4.2: A list of bids and a list of asks sorted by price. Bids are sorted by decreasing price while asks are sorted by increasing price.

to buy and who is willing to sell by offering the lowest price by beating competitors. By calling bp the price offered by the generic buyer and ap the price offered by the generic seller, then a transaction can only take place if $bp >= ap$, that is, if the buyer is willing to spend more than the seller expects to receive.

Each agent can be involved in multiple transactions respecting the only constraint of $bp >= ap$. In this case we chose to set the transaction price at the price offered by the seller.

For example in figure 4.3 we can see how the order matching system would have *solved* the offers in the book of figure 4.2. We can see how agent $A0$ buys 10 shares, buying 5 shares from $A1$, at the best available price and 5 shares from $A4$.

At the end of each step the share price is updated by making a weighted average of the transactions that have taken place. In the case of the transactions shown in figure 4.3 the price will be updated to the value

$$price_{t+1} = \frac{1.98 \cdot 5 + 2.05 \cdot 5 + 2.05 \cdot 2 + 2.09 \cdot 3}{15} = 2.03 \tag{4.5}$$

TRANSACTION			
Buyer	Seller	Price	Quantity
A0	A1	1.98	5
A0	A4	2.05	5
A3	A4	2.05	2
A3	A2	2.09	3

Figure 4.3: List of transaction obtained from the solution of bids and asks. agent A0 makes two transactions, one with agent A1 and agent A4.

Chapter 5

Library used

5.1 Introduction

In the previous chapter, MARL model was exposed from a theoretical point of view, in particular the reasons and the ways in which the agents act within the environment. In this chapter, you will see how this model has been implemented in practice. In particular we will see all the libraries that have been used and what role they play.

In this chapter, however, we will explain how the model was implemented in practice. As mentioned in the introduction, to build the model we started from the single agent model built by Adam King and exposed in [11], making substantial changes and making it a MARL model. The example exposed by Adam King was very important because it explained how to implement in practice a model that exploits reinforcement learning using very complete existing libraries. Another important source was the documentation of the Ray library [57], which excellently explains how it is possible to build a reinforcement learning environment by putting together different libraries. Figure 5.1 illustrates the stack of libraries, showing how they are put in communication with each other. At the base of the stack there is Ray, which makes it possible to distribute the execution of the code, but in this project no use was made of these features. Immediately above is RLlib which provides a large and updated set of algorithms and all the abstractions necessary to train an environment. At the top instead we find Gym that provides classes to build your own environment and many examples to be inspired by.

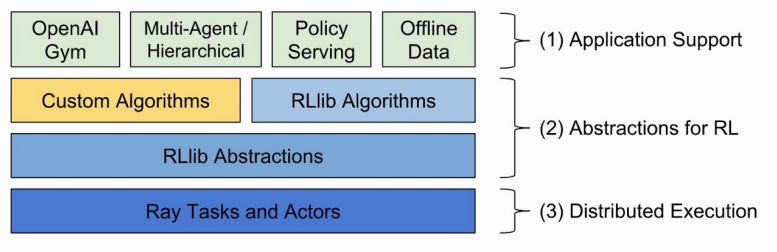

Figure 5.1: Stack of libraries [58]. The stack is divided into three levels: distributed execution, abstractions for RL and application support.

5.2 Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms [59]. The library is a collection of test problems environments that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms.

5.2.1 Gym's interface

[60] The core gym interface is *env*, which is the unified environment interface. The following are the *env* methods that would be quite helpful to us:

- *env.reset()*: Resets the environment. The process gets started by calling reset(), which returns an initial observation.
- *env.step(action)*: Step the environment by one timestep. Returns:
  - observation (object): an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.
  - reward (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
  - done (boolean): whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)

  - info (dict): diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning. [59]

- *env.render()*: Renders one frame of the environment, helpful in visualizing the environment.

5.3 RLlib

5.3.1 RLlib: Scalable Reinforcement Learning

RLlib is an open-source library for reinforcement learning that offers both high scalability and a unified API for a variety of applications. RLlib natively supports TensorFlow, TensorFlow Eager, and PyTorch, but most of its internals are framework agnostic [58]. To better understand the potential of the Rllib library, it is recommended to refer to the article by Eric Liang [61].

5.3.2 Policy

Policies are a core concept in RLlib. In a nutshell, policies are Python classes that define how an agent acts in an environment. Rollout workers query the policy to determine agent actions. In a gym environment, there is a single agent and policy. In vector envs, policy inference is for multiple agents at once, and in multi-agent, there may be multiple policies, each controlling one or more agents [62].

The policy class of rllib library is shown in figure 5.3 and it is visible how the policy class is made up of three objects: Model, Action Distribution and Policy Loss.

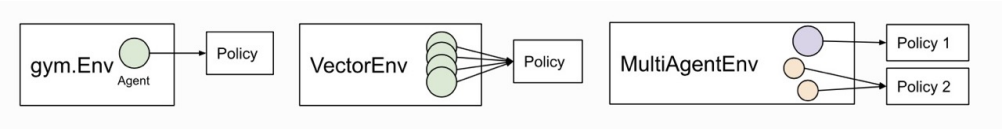

Figure 5.2: The figure represents the relationship with the policy of a gymEnv object, a VectorEnv and a MultiAgentEnv [62]. While the agents of a VectorEnv must share the policy and therefore only one will be present, in a MultiAgentEnv, there can be categories of different agents characterized by different policies.

5.3.3 RLlib Training APIs

Figure 5.3 shows the workflow of a worker, that is, of the class that takes care of running the environment. The figure highlights the classes that are editable in green and those that are not in purple, showing the great flexibility of the RLlib library. At a high level, RLlib provides a Trainer class which holds a policy for

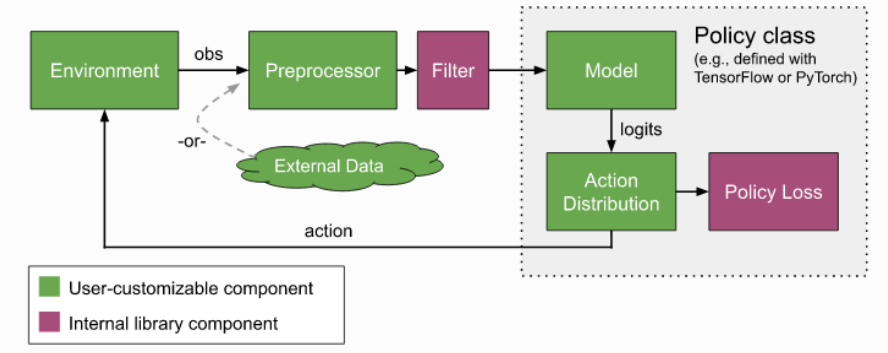

Figure 5.3: Block model of a worker workflow [63]. A worker is an instance of the model that has editable components such as the environment and user-editable components.

environment interaction. Through the trainer interface, the policy can be trained, checkpointed, or an action computed. In multi-agent training, the trainer manages the querying and optimization of multiple policies at once.

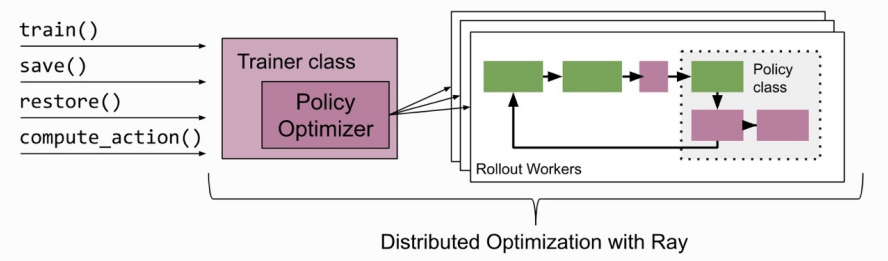

Figure 5.4: The figure shows how the trainer class that deals with improving the policy can simultaneously manage multiple workers each with their own environment[64]

.

You can train a simple DQN trainer with the following command:

```
rllib train --run DQN --env CartPole-v0
```

5.3.4 Evaluating Trained Policies

In order to save checkpoints from which to evaluate policies, set –checkpoint-freq (number of training iterations between checkpoints) when running rllib train.

An example of evaluating a previously trained DQN policy is as follows:

```
rllib rollout \
~/ray_results/default/DQN_CartPole-v0_0upjmdgr0/checkpoint_1/checkpoint-1 \
--run DQN --env CartPole-v0 --steps 10000
```

5.3.5 Tune: Scalable Hyperparameter Tuning

[65] Tune is a Python library for experiment execution and hyperparameter tuning at any scale. Core features:

- Launch a multi-node distributed hyperparameter sweep in less than 10 lines of code.
- Supports any machine learning framework, including PyTorch, XGBoost, MXNet, and Keras. See examples here.
- Visualize results with TensorBoard.

5.4 Giotto-tda

Topological Data Analysis (TDA) uses tools from algebraic and combinatorial topology to extract features that capture the shape of data[66] [5].In recent years, algorithms based on topology have proven very useful in the study of a wide range of problems. In particular, persistent homology has had significant impact on data intensive challenges[66]. Giotto-tda is a high-performance topological machine learning toolbox in Python built on top of scikit-learn and is distributed under the GNU AGPLv3 license. It is part of the Giotto family of open-source projects. [67]

5.4.1 Simplicial and Simplicial Complex

As excellently defined by C. Kelleher and A. Pantano [68]: an n-simplex is a geometric object with (n + 1) vertices which lives in an n-dimensional space (and cannot fit in any space of smaller dimension). The vertices of the simplex *generate* the simplex through a simple geometric construction which we illustrate below. The idea is easy: one vertex generates a point, two vertices generate a segment (by connecting the two points), three vertices generate a triangle (by connecting every pair of points with segments and filling the space in between) and so on.

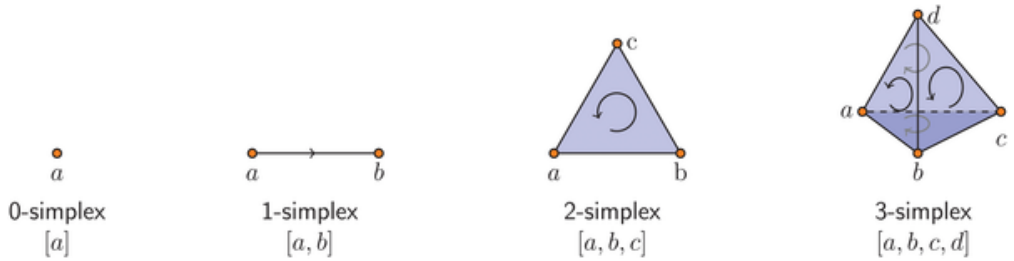

Figure 5.5: Simplexes of increasing size [68]. a 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron.

Instead, a simplicial complex K is a collection of simplices such that:

- If K contains a simplex σ, then K also contains every face of σ.
- If two simplices in K intersect, then their intersection is a face of each of them.

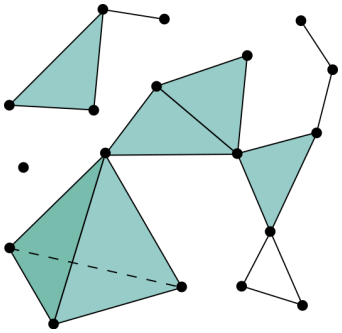

Figure 5.6: An example of Simplicial Complex. It is composed of points, line segments, triangles and a tetrahedron.

5.4.2 Persistent Homology

Persistent homology is one of the main tools in TDA. It measures the presence of topological invariants like connected components, holes, and voids across varying length scales. The *birth* and *death* of these invariants is summarised via a *persistence diagram* or a *persistence barcodes*, which is the most common and intuitive way to generate new types of features to feed to downstream machine learning tasks [69].

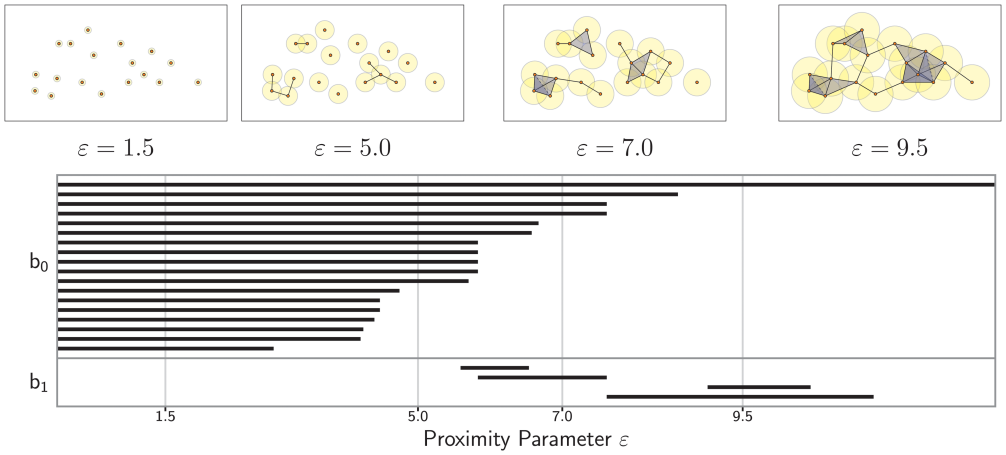

Figure 5.7: The filtration process applied to a 2-dimensional point cloud. At each step m, the distance threshold ϵ is increased, in fact, the circle around the point gets bigger and bigger. Each bar indicates a topological structure. At the first step there are as many b0 bars as there are points. By increasing the ϵ threshold, the b_0 structures are destroyed and generate new b_1 structures. The bars indicating the point of birth and death [70][71].

Taking figure 5.7 as a reference, where there is a simplicial complex K, as explained by Christian Block [70], a filtration is a process in which K *grows* as the epsilon threshold increases. In fact, as the threshold increases, as you can clearly see in figure, connected components are destroyed and new ones are formed. It is now possible to define the *birth* of a topological features as the epsilon value to which it was formed, while *death* is the epsilon value to which the component was destroyed. In the persistence barcodes, there will therefore be as many bars as there are topological features that are generated in the simplex.

5.4.3 From data to persistence diagrams

The first step is to represent the data as points in the space having coordinates (x, y, z). The set of points thus represented form a point cloud, as shown in figure 5.8, from which it is possible to extract important information. A good way to visually represent the information of the persistent homology is to use a collection of points represented on a Cartesian plane, where the x axis represents birth and the y axis death [72]. As we can see in figure 3.4, the points that are on the dotted line are the points that arise and die immediately, while a point that is at the top of the line is a long-lived topological feature. The color of the points indicates the dimension of the topological features. For example, a 0-dimensional topological feature is a connected component or cluster, a 1- dimensional topological feature is a hole, and so on as reported in [73].

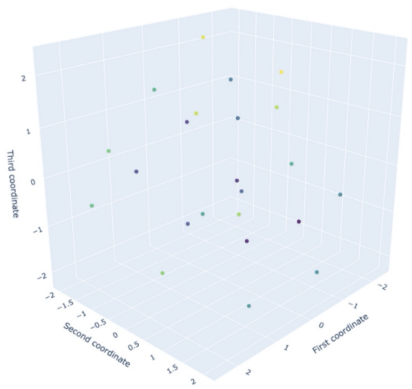

Figure 5.8: A point cloud for a set of data.

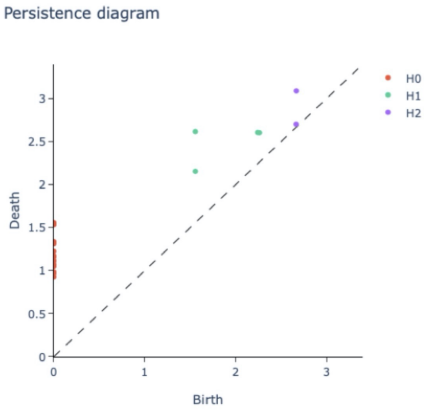

Figure 5.9: Persistence diagram of the point cloud in the previous figure. Connected components (H_0) are shown in red, holes (H_1) in cyan, and voids (H_2) in purple.

Thanks to *Giotto* library it is possible to calculate the persistent diagram simply by using the VietorisRipsPersistence function as follows where X windows is a point cloud:

```
homology_dimensions = (0, 1, 2)
VR = VietorisRipsPersistence(metric='euclidean', max_edge_length=100,
    homology_dimensions=homology_dimensions)
X_diagrams = VR.fit_transform(X_windows)
```

5.4.4 Sliding Windows

Sliding Windows is useful in time series analysis to convert a sequence of objects (scalar or array-like) into a sequence of windows of the original sequence. Each window has a *width* and stacks together consecutive objects, and consecutive windows are separated by a constant *stride*. This method is useful for segmenting a sequence of objects and analyzing them one by one. In fact, it is possible to study the evolution of the different topological features that may be present in the windows of a time series. To learn more about this technique it is useful to read Perea's work [74].

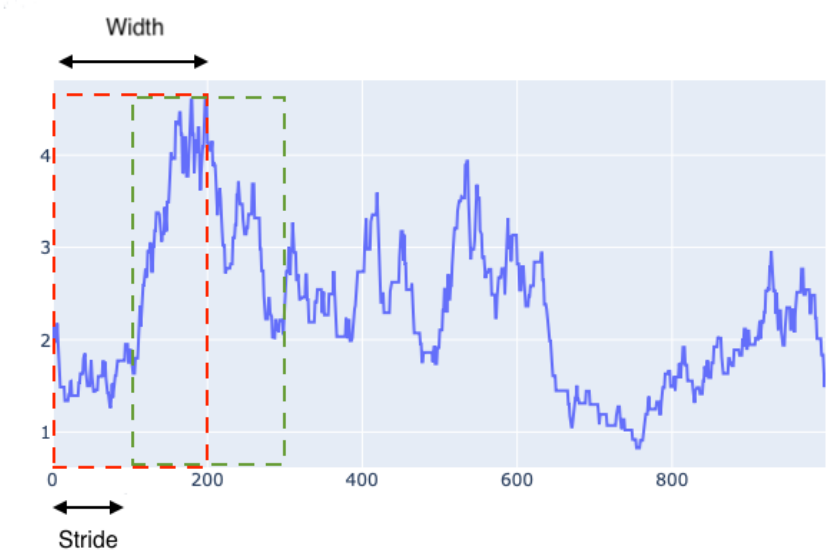

Figure 5.10: An example of the sliding window technique applied to a time series. The meaning of the width and stride parameters are highlighted. The width of the window indicates the number of time steps that are analyzed, while the stride is the number of time steps that is moved forward the window.

With *Giotto* library it is possible to obtain a sequence of windows as follows:.

```
window_width = 40
window_stride = 10
SW = SlidingWindow(width=window_width, stride=window_stride)
X_windows = SW.fit_transform(point_cloud)
```

- *width (int, optional, default: 10)* – Width of each sliding window. Each window contains width + 1 objects from the original time series.
- *stride (int, optional, default: 1)* – Stride between consecutive windows.

There is no method for choosing the parameters optimally, but the choice should be made on the basis of the sequence of data to be analyzed.

5.4.5 Persistence Entropy

Persistent entropy is a summary statistic of the information obtained from the persistent homology.

As defined in [75] Persistent entropy is an stable topological statistic [76] and can be seen as an adaptation of Shannon entropy (Shannon index in ecology) to the persistent homology context. Given a barcode without infinite bars $B = [a_i, b_i]_i = 1...n$ consider the length of the bars $l_i = b_i - a_i$ and their sum $L(B) = l_1 + ... + l_n$. Then, its persistent entropy is:

$$PE(B) = \sum_{i=0}^{n} -\frac{l_i}{L(B)} \log \frac{l_i}{L(B)} \tag{5.1}$$

in the following chapter, entropy will be differentiated into PE_1, PE_2 and PE_3 and will refer to homologies of sizes 0,1 and 2.

Persistent entropy therefore makes it possible to transform the persistent diagram into a quantitative measure.

Chapter 6

Experiments and Results

In order to understand how agents behave in different contexts, focusing on aspects of collaboration and competition, the model has been trained in two different scenarios: *without external perturbation* and *with external perturbation*. This chapter discusses about the analyzes that have been carried out in this two scenarios. The first section describes the implementation of the environment and in particular how it was trained and how the data was collected. The second section discusses the scenario in which there is no price perturbation. In the third, however, we will consider an alpha coefficient that disturbs the price. In both cases, we will try to understand how the agents' behavior is linked to the price trend. In the first case we will discover, thanks to game theory, that the agents collaborate. In the second, thanks to TDA, that there is a direct correlation between persistent entropy and price momentum.

6.1 Implementation

This paragraph explains in more specifically how the model shown in chapter 4 was implemented from a technical point of view. When setting the parameters of the environment and agents, it is important to be consistent with the scenario that needs to be analyzed.

It is necessary to set the initial parameters of the agents, defined in 4.2.1, such as the balance and the number of shares. But it is also necessary to set the common parameters, such as the initial price of the shares and the space to be given to agents in the actions they can take. In fact, it must be decided, the maximum quantity of shares that an agent can sell or want to buy and the maximum price variation, from the current price, with which to make the bet. Important parameters to keep in mind are the number of agents participating in the game and the number of steps that make up a game.

In this model two equal scenarios have been studied, the only difference being the presence of an exogenous variable (4.3.3) which disturbs the price. In particular, 10 agents were chosen with balance set at 1000 and number of shares at 100. The initial price of the shares is a random value between 1 and 5. We chose these values to give all agents the same initial financial position, with a good chance to buy or sell shares, thus starting from a situation of abundance. In fact, our interest is not to study how one agent manages to win over another, but to study how the relationships between agents are reflected in the share price and how they manage to respond to external disturbances, ensuring as little economic damage as possible. Each game consists of 1000 steps in which the agents are called to perform an action, in which they will be able to buy or sell a quantity of shares equal to half of those owned at a price having a maximum variation of 10 percent with respect to the current price.
Before collecting data from the environment it is necessary to train the agents. The training phase is carried out using the Tune library which reports all the parameters of the policy and the rewards, including the maximum, average and minimum rewards obtained in a game. The environments that were used in the data collection were trained until the parameters obtained as outputs were stable.

6.2 First Experiment (without price pertubation)

The first simulations were carried out without the presence of an exogenous variable that modifies the price of the shares. The price is therefore determined only by the transitions that take place between the agents. Recall that in this model agents can act on only one stock, and therefore it is impossible to observe particularly clever strategies. We also remind you that the agents' goal is to maximize the reward and that the formula used to calculate the reward shown in paragraph 4.2.5 is based on the net worth which is directly linked to the share price.

6.2.1 Price always grows, why?

Carrying out these analyzes, the first thing that can be noticed is that during the game the price of the shares always rises. In fig 6.1 we can see how in 1000 steps the price has almost tripled.

It might seem like an anomalous behavior, but also looking at figure 6.2 we can see how the net worth of agents generally has a growing trend. The agents may have understood that by increasing the share price their net worth grows and therefore they manage to obtain positive rewards.

To provide theoretical justifications for this behavior, was used game theory to allow us to simplify our model and study agent choices. From game theory we

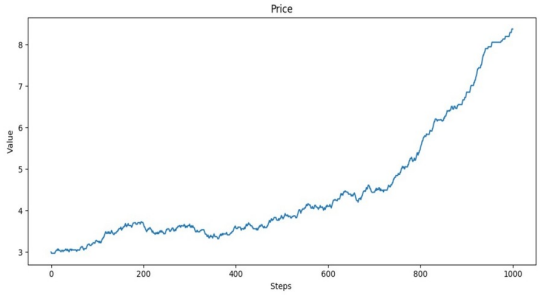

Figure 6.1: Example of the price trend during a one thousand steps game. As it is possible to observe the price increases rapidly without ever decreasing, this means that agents tend to always bet bids or asks with a raised price.

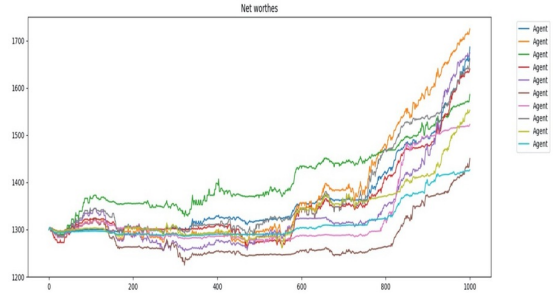

Figure 6.2: Example of the net worth trend of agents during a one thousand steps game. The game to which this figure refers is the same to which the figure 6.1 refers. In this figure it is visible how the value of the net worth directly correlated with the price of the shares. Increasing the value of the shares means increasing the value of the net worth and therefore generating positive rawards.

understood that the choice of agents to raise the share price is rational. Game theory and the simplified model are set out in the following paragraphs.

6.2.2 Game Theory

Game Theory is a mathematical concept, which deals with the formulation of the correct strategy that will enable an individual or entity (i.e., player), when confronted by a complex challenge, to succeed in addressing that challenge [77]. As defined by Giacomo Bonanno in the introduction of his book on game theory [78]:

> Game Theory provides a formal language for the representation and analysis of interactive situations, that is, situations where several "entities", called players, take actions that affect each other. The nature of the players varies depending on the context in which the game theoretic language is invoked: in evolutionary biology (see, for example, John Maynard Smith, 1982) players are non-thinking living organisms;1 in computer science (see, for example, Shoham-Leyton-Brown, 2008) players are artificial agents; in behavioral game theory (see, for example, Camerer, 2003) players are "ordinary" human beings, etc. Traditionally, however, game theory has focused on interaction among intelligent, sophisticated and rational individuals. For example, Robert Aumann describes game

> theory as follows: « Briefly put, game and economic theory are concerned with the interactive behavior of Homo rationalis – rational man. Homo rationalis is the species that always acts both purposefully and logically, has well-defined goals, is motivated solely by the desire to approach these goals as closely as possible, and has the calculating ability required to do so. » (Aumann, 1985, p. 35.)

6.2.3 Game definition

Let G be a game. Usually G is characterized by the following components:

- The agents who play the game and who then make a decision within the game, receiving a payoff. We indicate them with I, with $i = 1,2,3, ..., I$.
- The actions that each agent can perform in the game, which we indicate A_i.
- The presence of strategies: strategy is simply a predetermined *way of play* that guides an agent as to what actions to take in response to past and expected actions from other agents (i.e., players in the game) [77].
- The utility function (also called agent payoff) is a concept that refers to the amount of satisfaction that an agent derives from an object or an event [77].

6.2.4 Theory hypothesis

Our scenario is a symmetric and simultaneous game. Symmetrical means that all players can choose their moves from the same set of actions, and if the two players interchange their moves, the payoffs are also interchanged [79]. Instead, a simultaneous game or static game [80] is a game where each player chooses their action without knowledge of the actions chosen by other players [80]. As well defined in [81] the most important, and maybe one of the most controversial, assumption of game theory which brings about this discipline is that individuals are *rational*. An individual is *rational* if she has well defined objectives (or preferences) over the set of possible outcomes and she implements the best available strategy to pursue them.
One of the possible strategies that an agent can follow is *maxmin* strategy, where *maximin*, as defined in [82] value is the highest value that the player can be sure to get without knowing the actions of the other players; equivalently, it is the lowest value the other players can force the player to receive when they know the player's action.

6.2.5 Nash equilibrium theorem

As well defined in [83], it's possible to define nash equilibrium as: « A collection of strategies, one for each player in a social game, where there is no benefit for any player to switch strategies. In this situation, all players of the game are satisfied with their game choices at the same time, so the game remains at equilibrium. » Following this, where the natural world is governed by the laws of physics, the social world is governed by the Nash equilibrium. The concept is named after the American Mathematician John Nash, who won the 1994 Nobel Memorial Prize in Economic Sciences for his work on game theory.

6.2.6 Nash equilibrium example - Prisoner's dilemma

One of the most famous examples of game theory is the prisoner's dilemma. Which as well described in [77] is set out below.
In this game, two prisoners were arrested and accused of a crime; the police do not have enough evidence to convict any of them, unless at least one suspect confesses. The police keep the criminals in separate cells, thus they are not able to communicate during the process. Eventually, each suspect is given three possible outcomes:

- If one confesses and the other does not, the confessor will be released and the other will stay behind bars for ten years (i.e. -10);
- If neither admits, both will be jailed for a short period of time (i.e. -2,-2);
- If both confess, both will be jailed for an intermediate period of time (i.e. six years in prison, -6).

	Cooperate	Defect
Cooperate	(-2,-2)	(-10,0)
Defect	(0,-10)	(-6,-6)

Figure 6.3: Scheme prisoners dilemma game. On the rows there is the action performed by one of the two prisoners and on the columns by the other prisoner. For each cell there is the value returned by the utility function. For example for the action (Cooperate, Cooperate) both prisoners will have 2 years in prison. From this scheme it is possible to note that this game is a symmetrical game, in fact the utility of the actions (Defect, Cooperate) are the same as the actions (Cooperate, Defect).

To solve this game, we must find the dominating strategy of each player, which is the best response of each player regardless of what the other player will play. From player one's point of view, if player two cooperates (i.e. not admitting), then he is better off with the defect (i.e. blaming his partner). If player two defects, then he will choose defect as well. The same will work with player two. In the end, both prisoners conclude that the best decision is to defect, and are both sent to intermediate imprisonment.

6.2.7 Game of transactions between two agents

In the model studied there are 10 agents in the market, but we want to study what happens in a transaction and each transition is made by a pair of agents, so from the point of view of game theory there are 2 players. The set of actions from which an agent can choose are the following:

- 0 : the agent does nothing;
- B+ : the agent buys at a higher price than the current price;
- B- : the agent buys at a lower price than the current price;
- A+ : the agent sells at a higher price than the current price;
- A- : the agent sells at a lower price than the current price;

To be consistent with the model used, it was decided to use net worth as utility function. Like the reward function defined in paragraph 4.2.5, agents will prefer to perform actions that improve their net worth. Figure 4.3 shows a table showing the actions taken by two agents who make a transaction. The table values indicate the satisfaction of the agents for having made that transaction. The values are qualitative and not quantitative. They are used to express preferences between actions. The value 0 in some boxes of the table indicates the transactions that cannot take place. Recall that a transaction takes place only if those who want to buy offer a higher price than those who want to sell, for this reason the box (A +, B -) is 0.

The only actions that lead to a transaction are:

- (A+, B+): In this case both players are satisfied, because their net worth increases in value. In fact, the agent who buy will convert the money into shares, while the agent who sell will convert the shares into money. But what will both benefit is the increase in the price of the shares, which will increase the value of the shares owned and therefore the net worth. Both agents will have the same benefit, so the same value has been given as utility value, that is, one.

0	0	0	0	0	0
A-	(-1,-1)	(-1,-1)	0	0	0
A+	(1,)	0	0	0	0
B-	0	0	0	(-1,-1)	0
B+	0	0	(1,1)	(-1,-1)	0
	B+	B-	A+	A-	0

Figure 6.4: Scheme that shows the model of transactions following game theory. On the rows there is the action performed by one of the two prisoners and on the columns by the other prisoner. For each cell there is the value returned by the utility function. As with the prisoner's dilemma, this game is also symmetric. In fact, the cell (A +, B +) has the same utility values as the cell (B +, A +).

- (A-, B+) and (A-, B-): These two cases are both possible and can be considered equal, because the price of the transition is always the one offered by the seller. In this case both agents will get a negative value because lowering the price, they will also lower the value of the shares they own and consequently also their net worth.

As you can see from the figure 6.4, the agents have no reason to make different choices other than A + and B +, because they guarantee profits. The moves (A +, B +) and (B +, A +) have in fact (1, 1) as utility values which is the highest value that can be found in the cells of the diagram. While the moves (A +, A +) and (B +, B +) both have a utility equal to (0, 0) that do not cause a loss of value of their net worth. We can therefore say that the moves (A +, B +) are Nash's equilibria. It is therefore plausible that agents continue to pursue this strategy causing the price increase seen in figure 6.1.

Game theory was used to provide justification for an empirically observed compartment that is systematically repeated in all simulations that is, the increase in the price of the shares.

6.3 Last Experiment (with price pertubation)

After studying the behavior of agents in an environment in which they have full control of the price of shares, we studied an environment in which there is a exogenous variable, called alpha, able to modify the trend of shares price (chapter 4.3.3). In this case the goal is to understand if the agents manage to predict the sudden change in the price and how these strategies are reflected in the price trend.

6.3.1 Observed behaviors

After training the environment, some empirical tests were carried out to evaluate the price trend and the agents behavior. In figure 6.5 and in figure 6.6 it is possible to observe the trend of the price and the alpha coefficient in a thousand steps game. In figure 6.5 it is possible to see the trend of the alpha coefficient, which in this case is negative and therefore will act by decreasing the price value. This coefficient does not act at every step but only in the steps in which its value changes, from figure 4.4 it is possible to deduce what these steps are by looking at the steps the alpha coefficient changes value. Although the value of alpha coefficient is negative, so it negatively affects, it is possible to observe that the price trend in figure 6.6 has positive peaks due to the strategies adopted by the agents. In fact, observing the trend of the net worth in figure 6.6 it is possible to observe the trend of the net worth of the 10 agents during the 1000 steps that make up a game. Recall the formula used to model the reward in the chapter 4.2.5, the reward depends by net worth. At the end of the game the average reward obtained by the agents, step by step, is positive. We can therefore deduce that the agents are adopting strategies that allow to reach the end of the 1000 steps with a medium positive reward, therefore, on average, an increase in the net value of agents is guaranteed. Unfortunately it is difficult to deduce from these charts, if collaborative or competitive strategies are present and if there is actually a correlation between the actions taken by the agents and the price trend, for this reason it was decided to use topological data analysis (TDA), in order to extract information more accurately from high-dimensional datasets.

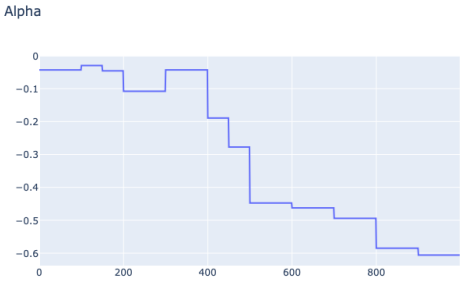

Figure 6.5: Trend of the alpha coefficient in a thousand steps game. It is possible to recognize the random walk as seen in figure [56].

Figure 6.6: Trend of the price with the influence of alpha in a thousand steps game. The case represented is the same in which the alpha coefficient of the figure 6.5 acts.

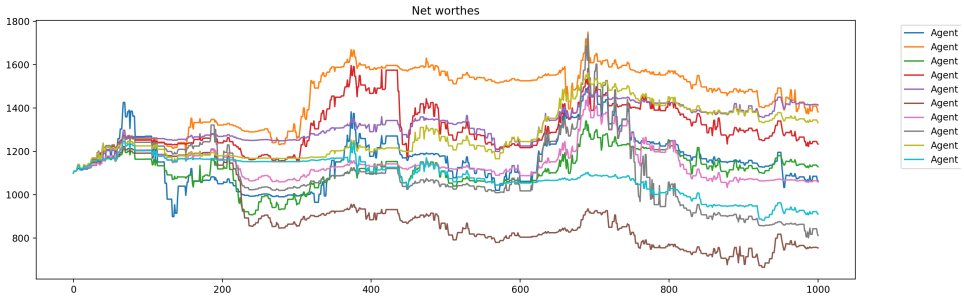

Figure 6.7: Trend of net worthes of the 10 agents in different colors. The case represented is the same in which the alpha coefficient of the figure 6.5 and 6.6 acts. As can be seen, just before the 800 step, some agents follow the price trend in the figure 6.6, this probably because their net worth depends almost exclusively on the shares held. The agents that have a more constant net worth value are those who base their wealth on balance.

6.3.2 Topological analyzes

The topological analysis was carried out by analyzing the trend of the net worth for each agent, step by step. We have chosen to analyze the value of the agents net worth for each step because this value is considered to be the most representative of the agents financial status. The analyzed values were extracted from a dataset of 1000 games. Once the net worth values are collected, they are sampled with a period of 10, in order to lighten the calculations by eliminating periods in which no transaction took place and there are no changes. After that, to obtain a sequence of point clouds that represent topologically the evolution of the financial state of the agents during the game, the sliding window technique was used (chapter 5.4.4), setting the width of a window to 40 and the stride to 10. Then we proceeded with extrapolating the persistent diagrams of each windows and with the calculation of the persistent entropy (chapter 5.4.5) of each diagram. From the calculation of the persistent entropy of a diagram, 3 values are obtained, representing the value of entropy for the components in homology 0, 1 and 2. In figure 6.8 you can observe the trends of these 3 values during 1000 steps. While entropy in homology 2 always remains at 0, entropy in homology 1 has peaks, also called spikes. While entropy in homology 2 always remains at 0, telling us that components in homology 2 never form, entropy in homology 1 has peaks or spikes, which tell us that in those windows, components in homology 1 were formed. In figure 6.9 it is possible to visually observe the point cloud of window 14, that relating to the first spike, while in figure 6.10 it is possible to study its persistence diagram. In order to interpret the signal of homology 1, an attempt was made to relate it to the time series of the price.

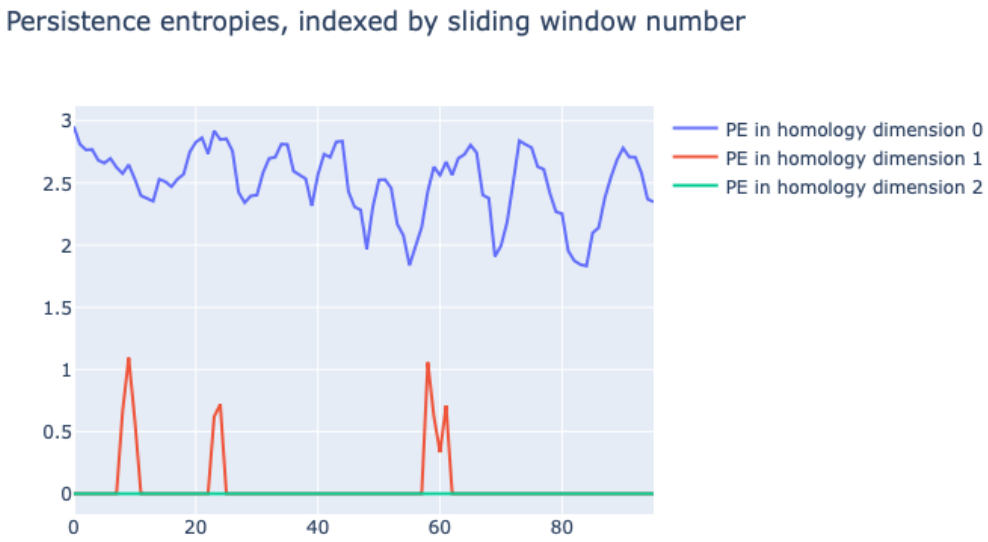

Figure 6.8: Persistent entropy signal indexed by the sliding window number. In red, the signal of persistent entropy in homology 1 clearly shows the presence of intact peaks at window 10, 20 and 60, highlighting the formation of components in homology 1.

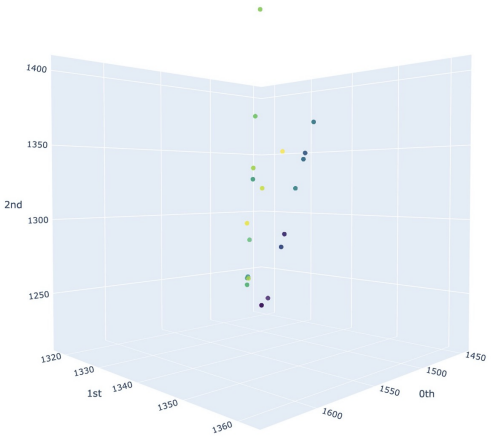

Figure 6.9: Point cloud, show one point for each step.

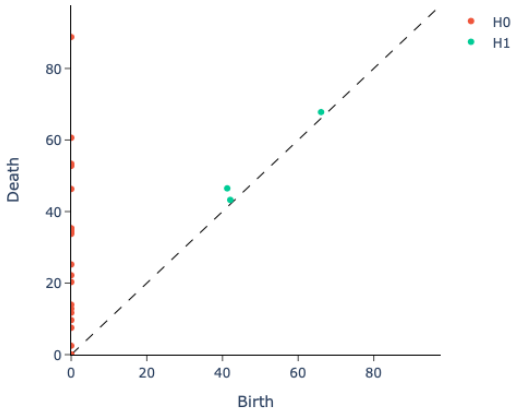

Figure 6.10: Persistence Diagram of window number 14. The value of birth is present on the abscissa axis, while death is on the ordinate axis. The red points are the homologies 0 and as you can see they all arise with zero abscissa and die when they connect to another topological structure.

6.3.3 Correlations

In the previous paragraph have been exposed the topological analyzes that have been carried out and the results that have been obtained. As you can see in figure 6.8 a clear topological signal has been obtained. In this section we will try to understand if the topological phenomenon that have been exposed have an effect on macroscopic variables such as the price of shares. We assume therefore that there is a direct relationship between the topological signal in homology 1 and the price, if this is true this relationship can be quantified by calculating the Pearson correlation.

To calculate the correlation, it was decided to relate the first derivative of the signal PE_1 and the first derivative of the price, that is, the momentum. As defined in [84] momentum is the speed or velocity of price changes in a stock, security, or tradable instrument. Momentum shows the rate of change in price movement over a period of time to help investors determine the strength of a trend. It is possible to observe the two time series in figure 6.11 where they are overlapped.

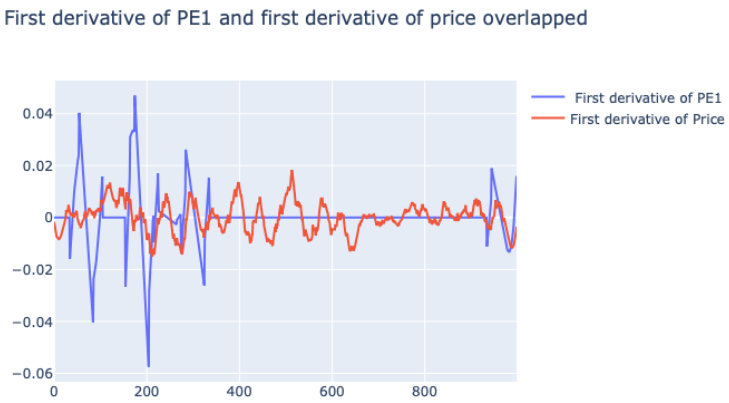

Figure 6.11: The figure shows the time series of the persistent entropy derivative in homology 1 in blue and the price derivative in red. Pearson's correlation will be calculated between these two time series.

To calculate the correlation, the entropy time series were moved with respect to the other two time series in a range of 40 positions forward and 40 backwards from its initial position, calculating the correlation and collecting only the maximum value.

In figure 6.12 it is possible to see the histogram of the maximum correlation between the price derivative and the PE_1 derivative calculated on the basis of

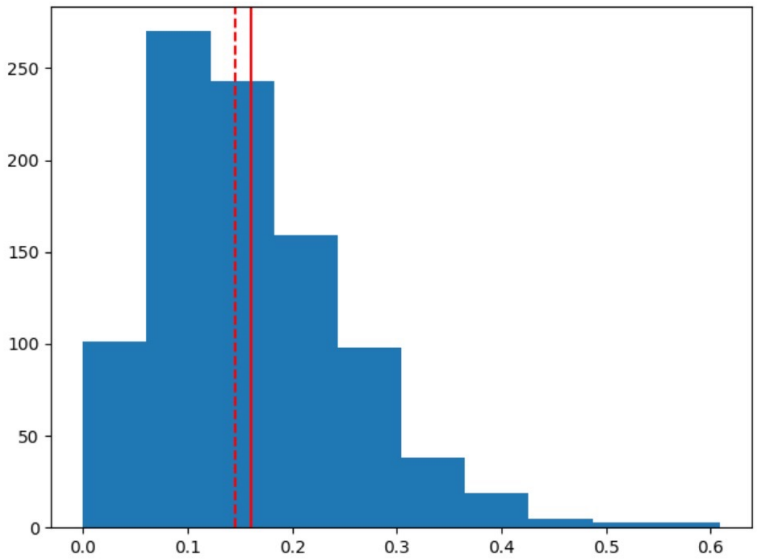

Figure 6.12: Histogram of the values of the correlation between the price derivative and the PE_1 derivative obtained over one thousand games. This histogram is the starting histogram used in the Bootstrapping procedure exposed below.

about a thousand simulations. The red line represents the average of the correlation values and stands at around 0.16. This value indicates the presence of a positive correlation and therefore confirms our hypothesis. In the next paragraph the result obtained will be validated using the Bootstrapping technique.

6.3.4 Bootstrapping

As excellently explained in [85], the bootstrap is a technique for evaluating the properties of the estimators (variance, bias) through re-sampling with replacement from the collected sample. In this case we will evaluate the mean of the correlation histogram obtained in previous paragraph. The algorithm used by bootstrapping can be summarized in these 4 points:

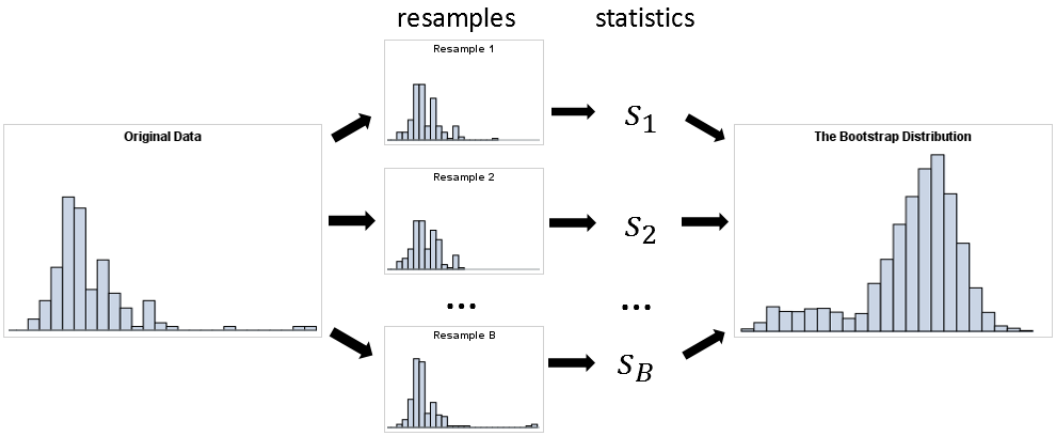

Figure 6.13: Illustrative figure of the bootstrapping workflow [86]. As can be seen, B resample with replacement is carried out from an initial data sample, generating B new datasets. For each of these B datasets the statistics will be calculated, in this case the correlation, generating B new values. With these B values it is possible to calculate a histogram, in this case it is shown in the figure 6.14.

- A sample $X_1^*, ..., X_n^*$ of number n is extracted with replacement from $X_1, ..., X_n$ which constitute the population of samples collected.
- The correlation is calculated as $Corr(X_1^*, ..., X_n^*)$
- The first two steps are repeated B times, obtaining $Corr_{n,1}, ..., Corr_{n,B}$
- The significance level of the data was assessed

In this case, the first two steps were repeated 10.000 times obtaining the same number of correlation values between momentum and H_1 signal. The histogram of this set of values is visible in figure 6.14. The histogram of the correlations that we want to validate in figure 6.12 has an average value of 0.16. To calculate the level of significance of the data, we will calculate the percentage of the correlation values obtained by re-sampling which are greater than 0.16 that is the average of

the histogram in figure 6.12. In figure 6.14 it is possible to see the histogram of the correlation values calculated on the one thousand re-sampling samples. As you can see, no value is greater than 0.16. The bootstrapping therefore allows us to say that the value 0.16, that is the average value of the correlation calculates empirically, has a significance level of 100%.

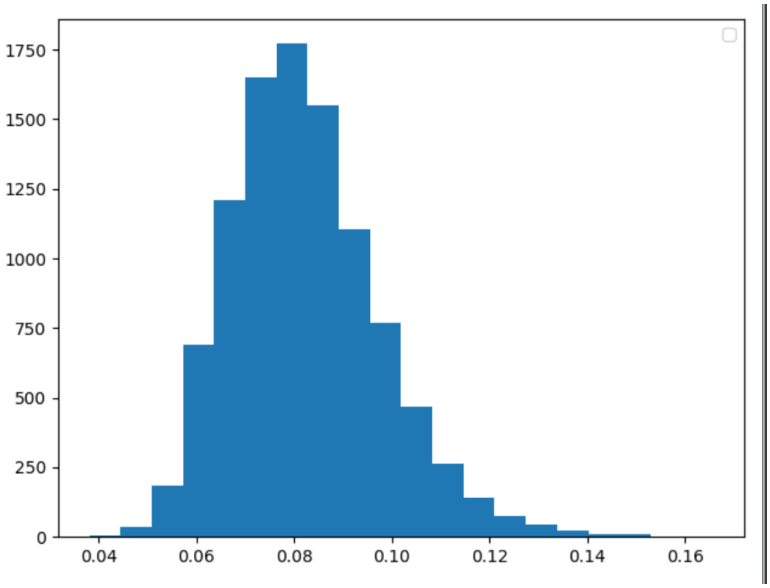

Figure 6.14: Histogram of correlation values obtained from one thousand re-samples. This histogram is obtained following the Bootstrapping procedure as shown in the figure 6.13.

6.3.5 What does this correlateion mean?

By studying the literature it is possible to find examples of interaction dynamics between agents that can influence the price trend. An example is the prey predator model which has been excellently treated by Miquel Montero in [87]. In particular Miquel states that «One of the most appealing properties of such a system is the presence of large oscillations of the number of agents sharing the same perspective, what may be linked with the existence of bullish and bearish periods in financial markets». In the case of this thesis, the dynamics with which the agents interact have not been specifically studied, it is therefore impossible to say whether the market model analyzed is predatory prey or not, but the existence of the topological signal in PE_1 confirms that the agents do not act randomly but follow a precise dynamic. But it was important to be able to quantify the existence of these dynamics. The paragraph 6.3.2, using topology, has demonstrated the existence of a signal in Homology 1. It is possible to interpret this result as the measurement of the dynamics with which the agents interact. Topology and in particular persistent entropy has been an excellent and easy tool for quantifying a dynamic that would otherwise be difficult to represent. Thanks to the PE_1 signal we can know when the agents are following certain strategies and correlate this signal to the price trend. In the paragraph 6.3.3 it has been explained how the correlation between the topological signal and the momentum of the price has been calculated. The result was positive and was subsequently validated in paragraph 6.3.4.

The existence of the correlation certifies that the agents do not act casually, but learn to follow strategies that have direct effects on the momentum of the price. This tells us that persistent entropy is able to *read* these strategies and quantify them through persistent entropy and that there is a direct relationship between these two signals, which can for example be used for the construction of forecasting models.

Chapter 7

Conclusion and future works

As anticipated in paragraph 6.3.5, the most interesting result is certainly the positive correlation between the trend of entropy in Homology 1 and the time series of the momementum of the price. For example because it could be a useful tool to read the time series of the price and deduce that kind of strategies were adopted by the agents. I therefore imagine that the direction to follow for future project developments is to continue to investigate the meaning of the correlation and to understand in which situations topological structures are formed and try to interpret what they mean from a practical point of view. A spike in PE_1 means that the agents are cooperating or competing? If so, how? It is in fact acceptable that agents put into practice certain behavior models that allow them to defend themselves from price fluctuations. As reported in 6.3.5 an example can be the predator prey model. Another point to be improved is certainly to make the model even closer to reality. In fact, it is possible to insert in the observation of the agents, other variables that an agent should take into account when making a choice. In fact, it would be useful to carry out a survey among traders to understand what are the technical variables they take into consideration when making a choice and then bring them back into the model. Improving the model could mean finding an even higher correlation value. Once better understood the nature of the topological signals, researchers could study after how many steps the spikes have effects and build an algorithm that exploits machine learning, which starting from the times series of price reconstructs the times series of persistent entropy and therefore also of the events that happened.

There are certainly other points in the model that can be improved to create new study possibilities. For example, in this model trading takes place on a single stock, it would be useful to extend the model, allowing you to trade on multiple stocks.

This would give agents the opportunity to implement more complex strategies which can then be analyzed by topology.

Bibliography

[1] Julie R. Kirkpatrick Charles D.; Dahlquist. *Technical Analysis: The Complete Resource for Financial Market Technicians.* Financial Times Press., 2006. ISBN: 978-0-13-153113-0 (cit. on p. 1).

[2] Shengtao Sun Jibing Gong. «A New Approach of Stock Price Trend Prediction Based on Logistic Regression Model». In: *International Conference on New Trends in Information and Service Science* (2009). DOI: 10.1109/NISS.2009.267 (cit. on pp. 1, 5).

[3] Md Jan Nordin Han Lock Siew. «Regression Techniques for the Prediction of Stock Price Trend». In: (Sept. 2012). DOI: 10.1109/ICSSBE.2012.6396535 (cit. on pp. 1, 5).

[4] Tugrul U.Daimb Erkam Guresena Gulgun Kayakutlua. «Using artificial neural network models in stock market index prediction». In: (Feb. 2011). DOI: https://doi.org/10.1016/j.eswa.2011.02.068 (cit. on p. 1).

[5] Gunnar Carlsson. «TOPOLOGY AND DATA». In: *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY* 46.2 (Apr. 2009), pp. 255–308 (cit. on pp. 1, 39).

[6] Emanuele Merelli, Matteo Rucco, Peter Sloot, and Luca Tesei. «Topological Characterization of Complex Systems: Using Persistent Entropy». In: (Sept. 2015), p. 17. eprint: 6872-6892 (cit. on p. 1).

[7] Yuri Katz Marian Gidea. «Topological Data Analysis of Financial Time Series: Landscapes of Crashes». In: (Apr. 2017). DOI: arXiv:1703.04385 (cit. on pp. 1, 2).

[8] Bertrand Michel Frédéric Chazal. «An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists». In: (Oct. 2017). DOI: arXiv:1710.04019 (cit. on pp. 1, 6).

[9] Xiao-Yang Liu Wenhang Bao. «Multi-Agent Deep Reinforcement Learning for Liquidation Strategy Analysis». In: (June 2019). DOI: arXiv:1906.11046 (cit. on pp. 2, 6).

[10] Yagna Patel. «Optimizing Market Making using Multi-Agent Reinforcement Learning». In: (Dec. 2018). DOI: arXiv:1812.10252v1 (cit. on p. 2).

[11] Adam King. «Create custom gym environments from scratch — A stock market example». In: *Towards Data Science* (2019). URL: https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e (cit. on pp. 2, 5, 35).

[12] et al. Lum P. Y. «Extracting insights from the shape of complex data using topology». In: *Scientific Reports* (2013). URL: https://doi.org/10.1038/srep01236 (cit. on pp. 2, 6).

[13] David Hirshleifer. «Behavioral Finance». In: *Annual Review of Financial Economics* (2015), pp. 133–59. DOI: 10.1146/annurev-financial-092214-043752 (cit. on p. 5).

[14] Victor Ricciardi and Helen K. Simon. «What is Behavioral Finance?» In: *Business, Education and Technology Journal* (2000) (cit. on p. 5).

[15] Tieqiang Li Devraj Basu. «A Machine-Learning-Based Early Warning System Boosted by Topological Data Analysis». In: (June 2019). URL: http://dx.doi.org/10.2139/ssrn.3394704 (cit. on pp. 5, 6).

[16] Anshul Mittal. «Stock Prediction Using Twitter Sentiment Analysis». In: (2011) (cit. on p. 5).

[17] Rachid Ellaia Mohamed Amine Souissi Khalid Bensaid. «Multi-agent modeling and simulation of a stock market». In: *Investment Management and Financial Innovations* (Nov. 2018), pp. 123–134. DOI: 10.21511/imfi.15(4).2018.10 (cit. on pp. 5, 6).

[18] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. «Reinforcement Learning for Optimized Trade Execution». In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 673–680. ISBN: 1595933832. DOI: 10.1145/1143844.1143929. URL: https://doi.org/10.1145/1143844.1143929 (cit. on p. 6).

[19] Jae Won Lee. «Stock price prediction using reinforcement learning». In: 1 (2001), 690–695 vol.1 (cit. on p. 6).

[20] Aleksandras Vytautas Rutkauskas and Tomas Ramanauskas. «Stock price prediction using reinforcement learning». In: 10.4 (2009), pp. 329–341. DOI: https://doi.org/10.3846/1611-1699.2009.10.329-341 (cit. on p. 6).

[21] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts, London, England: The MIT Press, 2014, 2015 (cit. on pp. 6, 10, 15).

[22] «Introduction to RL, Part1: Key concept in RL». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#states-and-observations (cit. on pp. 6, 12, 15).

[23] Wlodek Zadrozny Shafie Gholizadeh. «A short survey of topological data analysis in time series and systems analysis». In: (Oct. 2018). DOI: arXiv:1809.10745v2 (cit. on p. 6).

[24] U. Lupo W. Lemes De Oliveira L. Tunstall and A. Medina Mardones. «Detecting stock market crashes with topological data analysis». In: (Nov. 2019). URL: https://towardsdatascience.com/detecting-stock-market-crashes-with-topological-data-analysis-7d5dd98abe42 (cit. on p. 6).

[25] Steven Borowiec. «AlphaGo seals 4,1 victory over Go grandmaster Lee Sedol». In: *The Guardian* 4 (Mar. 2016) (cit. on p. 10).

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. «Playing Atari with Deep Reinforcement Learning». In: (2013). arXiv: 1312.5602 [cs.LG] (cit. on p. 10).

[27] T. Urbanik I. Arel C. Liu and A.G. Kohls. «Reinforcement learning-based multi-agent system for network traffic signal control». In: *IET Intell. Transp. Syst* 4.2 (2010), pp. 128–135. DOI: 10.1049/iet-its.2009.0070 (cit. on p. 10).

[28] Jan Peters Jens Kober J. Andrew Bagnell. «Reinforcement Learning in Robotics: A Survey». In: *The International Journal of Robotics Research* (Sept. 2013). DOI: 10.1177/0278364913495721 (cit. on p. 11).

[29] Xiaocheng Li Zhenpeng Zhou and Richard N. Zare. «Optimizing Chemical Reactions with Deep Reinforcement Learning». In: *ACS Cent. Sci.* 3.1 (2017), 13371344. DOI: 10.1021/acscentsci.7b00492 (cit. on p. 11).

[30] Xiaocheng Li Zhenpeng Zhou and Richard N. Zare. «Optimization of Molecules via Deep Reinforcement Learning». In: *Scientific Reports* 9 (2019). DOI: 10.1038/s41598-019-47148-x (cit. on p. 11).

[31] Chris Nicholson. «A Beginner's Guide to Deep Reinforcement Learning». In: *pathmind* (2019). URL: https://pathmind.com/wiki/deep-reinforcement-learning (cit. on p. 13).

[32] «RLlib Training APIs: Trajectories». In: *ray.io* (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#trajectories (cit. on p. 14).

[33] «Reward and Return». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#states-and-observations (cit. on p. 15).

[34] «Understanding RL: The Bellman Equations». In: *joshgraves* (). URL: joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/ (cit. on pp. 15, 18).

[35] «Value Functions». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#states-and-observations (cit. on p. 16).

[36] «Model-Free vs Model-Based RL». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#states-and-observations (cit. on p. 18).

[37] «Part 2: Kinds of RL Algorithms». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#part-2-kinds-of-rl-algorithms (cit. on p. 19).

[38] «What is Model-Based Reinforcement Learning?» In: (). URL: https://medium.com/the-official-integrate-ai-blog/understanding-reinforcement-learning-93d4e34e5698 (cit. on p. 19).

[39] Alan Mackworth David Poole. «Artificial Intelligence: Foundations of Computational Agents, second edition». In: (). URL: https://artint.info/html/ArtInt_268.html (cit. on p. 20).

[40] «What to Learn in Model-Free RL». In: (). URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#what-to-learn-in-model-free-rl (cit. on p. 20).

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. «Proximal Policy Optimization Algorithms». In: (2017). arXiv: 1707.06347 [cs.LG] (cit. on p. 20).

[42] X. Ma P. Hämäläinen A. Babadi and J. Lehtinen. «PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation». In: (Aug. 2019). DOI: arXiv:1810.02541v7 (cit. on p. 20).

[43] DANNY WEYNS, MICHAEL SCHUMACHER, ALESSANDRO RICCI, MIRKO VIROLI, and TOM HOLVOET. «The Knowledge Engineering Review». In: *The Knowledge Engineering Review* 20:2.1 (2005), pp. 127–141. DOI: 10.1017/S0269888905000457 (cit. on p. 21).

[44] «CASES FOR APPLYING MULTI-AGENT REINFORCEMENT LEARNING». In: *silo ai* (). URL: https://silo.ai/applying-multi-agent-reinforcement-learning/ (cit. on pp. 21, 22).

[45] et al. Weyns Danny. «Environments in multiagent systems». In: *The Knowledge Engineering Review* (Aug. 2005). DOI: 10.1017/S0269888905000457 (cit. on p. 22).

[46] «Multi-Agent and Hierarchical». In: *ray.io* (). URL: https://docs.ray.io/en/master/rllib-env.html?highlight=multi%5C%20agents#multi-agent-and-hierarchical (cit. on p. 23).

[47] Victor Lesser Chongjie Zhang and Prashant Shenoy. «A Multi-Agent Learning Approach to Online Distributed Resource Allocation». In: *IJCAI* 9 (2009) (cit. on p. 23).

[48] Binda Pandey. «Adaptive Learning For Mobile Network Management.» In: (2016) (cit. on p. 23).

[49] Andrew Moore Riedmiller Martin and Jeff Schneider. «Reinforcement Learning for Cooperating and Communicating Reactive Agents in Electrical Power Grids. In: Balancing Reactivity and Social Deliberation in Multi-Agent Systems. BRSDMAS 2000. Lecture Notes in Computer Science». In: 2103 (2000). DOI: https://doi.org/10.1007/3-540-44568-4_9 (cit. on p. 23).

[50] Marco A. Wiering. «Multi-agent reinforcement learning for traffic light control.» In: *ICML* (2000) (cit. on p. 23).

[51] «Stock market». In: (). URL: https://en.wikipedia.org/wiki/Stock_market (cit. on p. 25).

[52] «What Is the Stock Market and How Does It Work?» In: *nerd wallet* (). URL: https://www.nerdwallet.com/blog/investing/what-is-the-stock-market/ (cit. on p. 25).

[53] «Equazioni di Lotka-Volterra». In: (). URL: https://it.wikipedia.org/wiki/Equazioni_di_Lotka-Volterra (cit. on p. 26).

[54] «Random walk». In: (). URL: https://en.wikipedia.org/wiki/Random_walk (cit. on p. 30).

[55] Eugene F. Fama. «Random Walks in Stock Market Prices». In: *Financial Analysts Journal* 51.1 (1995), pp. 75–80. DOI: 10.2469/faj.v51.n1.1861. eprint: https://doi.org/10.2469/faj.v51.n1.1861. URL: https://doi.org/10.2469/faj.v51.n1.1861 (cit. on p. 30).

[56] Morn. «Random Walk». In: (Oct. 2008). DOI: arXiv:1707.06347v2. URL: https://it.wikipedia.org/wiki/Passeggiata_aleatoria#/media/File:Random_Walk_example.svg%20http://www.gnu.org/licenses/fdl-1.3.html (cit. on pp. 31, 55).

[57] «What is Ray?» In: (). URL: https://docs.ray.io/en/master/ (cit. on p. 35).

[58] «RLlib: Scalable Reinforcement Learning». In: *ray.io* (). URL: https://docs.ray.io/en/master/rllib.html#rllib-scalable-reinforcement-learning (cit. on pp. 36, 37).

[59] «Getting Started with Gym». In: (). URL: https://gym.openai.com/docs/ (cit. on pp. 36, 37).

[60] «Reinforcement Q-Learning from Scratch in Python with OpenAI Gym». In: (). URL: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/ (cit. on p. 36).

[61] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. «RLlib: Abstractions for Distributed Reinforcement Learning». In: (2017). arXiv: 1712.09381 [cs.AI] (cit. on p. 37).

[62] «RLlib Training APIs: Policies». In: *ray.io* (). URL: https://docs.ray.io/en/master/rllib.html#policies (cit. on p. 37).

[63] «RLlib Models, Preprocessors, and Action Distributions». In: *ray.io* (). URL: https://docs.ray.io/en/master/rllib-models.html#rllib-models-preprocessors-and-action-distributions (cit. on p. 38).

[64] «RLlib Training APIs». In: *ray.io* (). URL: https://docs.ray.io/en/master/rllib-training.html#rllib-training-apis (cit. on p. 38).

[65] «Tune: Scalable Hyperparameter Tuning». In: *ray.io* (). URL: https://docs.ray.io/en/master/tune.html#tune-scalable-hyperparameter-tuning (cit. on p. 39).

[66] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess. «giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration». In: (2020). arXiv: 2004.02551 [cs.LG] (cit. on p. 39).

[67] «Documentation». In: (). URL: https://github.com/giotto-ai/giotto-tda (cit. on p. 39).

[68] CASEY KELLEHER and ALESSANDRA PANTANO. «INTRODUCTION TO SIMPLICIAL COMPLEXES». In: (). URL: https://www.math.uci.edu/~mathcircle/materials/MCsimplex.pdf (cit. on p. 40).

[69] «Getting started with giotto-tda». In: *Towards data science* (). URL: https://towardsdatascience.com/getting-started-with-giotto-learn-a-python-library-for-topological-machine-learning-451d88d2c4bc (cit. on p. 42).

[70] Christian Bock. «A gentle introduction to persistent homology». In: (June 2019). URL: https://christian.bock.ml/posts/persistent_homology/ (cit. on p. 42).

[71] «Topological Data Analysis – Persistent Homology». In: (Mar. 2017). URL: https://analyticks.wordpress.com/2017/03/10/topological-data-analysis-persistent-homology/ (cit. on p. 42).

[72] Tamal K. Dey. «Notes by Tamal K. Dey, OSU». In: *The International Journal of Robotics Research* (2017). URL: http://web.cse.ohio-state.edu/~dey.8/course/CTDA/CTDA.html (cit. on p. 43).

[73] Andrew Marchese. «Data Analysis Methods using Persistence Diagrams». In: *The International Journal of Robotics Research* (2017). URL: https://trace.tennessee.edu/utk_graddiss/4700 (cit. on p. 43).

[74] Perea J. A. and Harer J. «Sliding Windows and Persistence: An Application of Topological Methods to Signal Analysis.» In: *Founda- tion of Computational Mathematics* (Nov. 2013). DOI: arXiv:1307.6188 (cit. on p. 44).

[75] M. Soriano-Trigueros M. J. Jimenez. «Persistent entropy: a scale-invariant topological statistic for analyzing cell arrangements». In: (May 2019). DOI: arxiv,1902.06467v4 (cit. on p. 45).

[76] R. Gonz alez-D ıaz N. Atienza and M. Soriano-Trigueros. «On the stability of persistent entropy and new summary functions for TDA». In: (2018). DOI: arxiv,abs/1803.08304 (cit. on p. 45).

[77] Hamed Al-raweshidy Omar Raoof. «Theory of Games: an Introduction». In: (Sept. 2010). DOI: 10.5772/46930 (cit. on pp. 49–51).

[78] Giacomo Bonanno. *Game Theory. 2nd Edition.* CreateSpace Independent Publishing Platform, 2015, 2018 (cit. on p. 49).

[79] Erich Prisner. *Game Theory Through Examples.* The Mathematical Association of America, 2014. ISBN: 978-1-61444-115-1 (cit. on p. 50).

[80] George Norman Lynne Pepall Dan Richards. *Industrial organization : contemporary theory and empirical applications.* Jan. 2014. ISBN: 978-1-118-25030-3 (cit. on p. 50).

[81] Efe A. Ok Levent Kockesen. *An Introduction to Game Theory.* 2007 (cit. on p. 50).

[82] Michael Maschler, Eilon Solan, and Shmuel Zamir. *Game Theory.* Cambridge: Cambridge University Press, Jan. 2013. DOI: 10.1017/CBO9780511794216 (cit. on p. 50).

[83] Stephanie Glen. «Nash Equilibrium: Simple Definition and Examples». In: (). URL: https://www.statisticshowto.com/nash-equilibrium/ (cit. on p. 51).

[84] Investopedia. «Momentum Indicates Stock Price Strength». In: (Oct. 2019). URL: https://www.investopedia.com/articles/technical/081501.asp (cit. on p. 59).

[85] Giovanni M. Marchetti. *Note a "All of Statistics"*. May 2018. Chap. Il bootstrap (cit. on p. 61).

[86] Rick Wicklin. «The essential guide to bootstrapping in SAS». In: *The DO Loop* (Dec. 2018). URL: `https://blogs.sas.com/content/iml/2018/12/12/essential-guide-bootstrapping-sas.html` (cit. on p. 61).

[87] Miquel Montero. «Predator-Prey Model for Stock Market Fluctuations». In: (Oct. 2009). DOI: `arXiv:0810.4844v4` (cit. on p. 63).