A Comprehensive Review of Machine Learning Methods in Stock Market

by

Magdalena Spinu

A thesis in partial fulfilment
of the requirements for the degree of
Master of Science
(Information Systems and Technology)
in the University of Michigan-Dearborn
2022

Master's Thesis Committee

Assistant Professor Jin Lu, Chair
Assistant Professor Birhanu Eshete
Assistant Professor Zhen

Acknowledgements

This thesis would have not been possible without the support and guidance of my advisor Dr. Jin Lu; I am forever grateful for your help and contribution**.** I want to thank my husband for his constantan support and inspiration for pushing me to do better. While I was working on this thesis wonderful events happen in my life, one of which was the birth of my son. Therefore, this Thesis is memorable to me.

Preface

I came to the USA in 2014 on a music scholarship to study violin music performance. After 1 year in US, I have realized that music is not meant to be more than a hobby which led to me switching to accounting. Fast-forward to 2018, I have graduated with a Bachelor of Science in Accounting. After 1 year of work in accounting industry I felt that many of the tasks done by an accountant could be automated which would greatly improve the job satisfaction. Hoping that I can learn the tricks and master the field of technology I decided to pursue a Master in Information Science and Technology, so that I can have the knowledge needed for automating the analytics task which an accountant is bound to do for example Financial Statements at year or quarter end. My father who has passed because of cancer in 2020 was an engineer and a pioneer of learning new things and was very passionate about the stock market. Hence, I have decided to take my father's interest further and do the analysis of the different factors that affect the stock market performance and the different methods one can go about to improve the return on investment.

Table of Contents

List of Tables

List of Figures

Abstract

The analysis of the performance of different agents on 20 stocks given a 6 years' time range of the stock market, comparison between the agents, and identifying the most/least reliant agent. Identifying and evaluating the different factors and events that contribute to the changes in the stock market and proposing a potential solution for better stock performance

Chapter 1 Introduction

The advancement of technology led to an expansion across all industries. The discovery of intelligent systems is a big step forward for humankind, leading to a more productive and efficient world. As the famous saying goes, “Work smarter, not harder.” The progress in the technical world led to a new distribution of work. Now, one can design algorithms that will do repetitive tasks instead of spending countless hours working out each task by hand. The invention of algorithms led to a more precise and optimal output, which resulted in better decisions, and hence more profit when it comes to stock trading.

Similar work that we have looked at is a comprehensive state-of-the-art investigation of the recent advances in data science in emerging economic applications (Nosratabadi). However, this work focuses on the hybrid models. Le, D. Y. N., Maag et al provides an extensive research on Deep Learning and Machine Learning models. The authors extracted required quantitative information, applications, and results on different methodologies. Sarangi et al presented a detailed review of different learning models used to predict the stock market for last 50 years (1970 to 2020) in the paper “A Study on Stock Market Forecasting and Machine Learning Models.” In our work we focus on how the different agents perform in the stock market given a range of 6 years.

1.1 Financial Markets

Here is provided a brief introduction to the critical aspects of financial markets, starting with the definition.

An investor is a person who puts in money with the expectation of them increasing. Stock market has been commonly defined as the place where numerous investors meet to buy and sell stocks. To ensure a profitable investment, one needs to know as much as possible about the financial as well as the potential of the company they wish to put their money in.

The rise in artificial intelligence helped numerous investment companies create optimum portfolios for their clients, such as mixed portfolios: combining a wide range of investments

from the most aggressive to the safest, such as stock, bonds, cash and money security. A good investor will adjust the above based on their risk tolerance and the time available.

Even though the term “Financial market “is widely used in society with the same core meaning, its true meaning depends on the context it is used. First, it was applied in the business world in the context of the rise of the stock market, also identified as the stock market. (Adambekova).

At the core, the function of the financial market is to reallocate funds from a group that has an excess of funds to those who need tangible resources to invest in, along with the risks that they carry. Therefore, one can notice three aspects of the financial market: A traded asset does not have a fixed price, but rather is determined by the interactions between the buyers.

1. The financial market allows the investors to let go of the obligation of keeping an asset by shifting its risks and obligations onto a different investor.
2. By far the best part of the stock market is the reduction in cost of a transaction as the cost for search and information are almost non-existent.

Depending on whether the funds were borrowed or provided, one can differentiate the market by the nature of the claim. The difference between the cash market and derivatives market is the timing of the claim. Another aspect of the market is the maturity of claims, as there are markets with long-term claims known as capital markets and short-term claims or money markets. Seasoning of claims is another property of the financial market. Here we can distinguish between primary markets that deal with newly issued claims and secondary markets that work with previously issued claims.

1.2 How to Trade

As Warren Buffet once said, “Investing is a way to set aside money now so that it can multiply in the future.” Today there is a wide range of possibilities to invest in, starting with online brokers who can make the investment for you for a set fee. The big companies normally use these types of investors as they come with substantial fees or percentages of transactions.

Robo-Advisors are the use of technology meant to streamline and lower investment costs through the help of technology which uses different algorithms meant to filter out and detect the best stock choices given the parameters. Another way of investing is through the employer, which given a small percentage of one’s salary is deducted before tax and invested into a retirement plan.

1.3 Stock Market

Each ownership share in a corporation authorizes the holder to one vote on the company affairs, also called stocks. Afterward, the vote is debated at the annual get-together and decided on how it may better the company's finances. There are known to be two kinds of stocks that are differentiated by the amount of influence the priority attached to the influence of the earnings. Hence, one can distinguish between preferred stockholders, who benefit from fixed dividends, and everybody else or common stockholders who receive the gains at the end (Aggarwal).

Common Stock has two characteristics differentiated by the nature of the claim: residual- meaning that unless the company is in liquidation, the shareholders will receive their share last once the interest and taxes have been paid and limited liability, which is probably the safest yet not most rewarded as the most one can lose is the amount initially invested. A limited liability shareholder is not personally liable for the obligations of the firm (Ireland).

When a company wants to expand, it issues stock so that it can gather new financial sources. This process is done through an initial public offering known as floating. This process transitions a company from being private to a public one. Further, the IPO is subsidized by an investment bank that will arrange for the shares to be recorded on the stock exchanges. *When a company* publishes its shares on trading platforms, the profits go straight to the issuing firm, a process known as primary offering, or to any private investors, a process known as secondary offering (Prasad D.).

Once the shares are available for public purchase, the funds pass unrestricted between the public investors. In this case, the stock market will act as a secondary market. The only difference is that the company will not be influenced by these changes in capital or be required to reimburse the money to the individuals who have invested. The lenders must undergo all the ups and downs that the market might take. Hence, the market capitalization or the value is determined by the total of all the shares (Dias). In the case of a common investor, the profit gained from one of the shares comes from two different sources:

1. Dividend payment- a payout which is typically done by more mature companies in the shape of cash or extra shares of stock. This payment is not mandatory and hence ignored by the newly established companies.
2. Changes in the price of the stock- the differences between the purchase price and the current price at which the shares are being traded determine either the capital gain or capital

loss. In essence, the owner of the shares is determining the gain or loss considering the sale at that specific moment of time.

Some investors take a different yet still popular approach to trading stocks known as short selling. As a result of this the investor makes profits when the prices fall rather than when they rise. Short selling is done by borrowing shares from a broker and selling them to a third party, afterwards the seller buys back the shares from the third party and gives them back to the original broker. If during this time the price has gone down, then the equivalent the investor made in profits (Woolridge).

Supply and demand are just one aspect of the financial market that decides the prices. To understand these other factors, analysts and investors implement the knowledge gained acquired by fundamental analysis and technical analysis. The analysis of the financial state of a company to determine its value is known as Fundamental Analysis (Abad). This idea is that even if a stock has a price that does not correspond to the share's actual value, eventually, the correct price will reach in the end. Hence, by following this idea, an investor can buy or sell a stock at the wrong price and then wait until the market naturally reprices the share. On the other hand, technical analysis uses past data to predict the trend of prices. This method depends on techniques known as time series analysis, also statistical and signal analysis.

The techniques mentioned above were a go-to for a long time. However, a new idea recently emerged, which states that market prices are basically impossible to predict. This theory is partly based on the efficient market hypothesis that states that the market prices reflect the information available, Hence the idea that it is not possible to beat the market (Malkiel).

The new ideas directed investors and analysts toward algorithmic trading, where investors tried to automate the process using computer programs *b*ecause these algorithms can handle a lot more input at a parallel time and are more efficient in analyzing the information than a human trader. A few of these strategies are:

1. Trend following strategies: such as moving averages, channel breakouts, price level movements, and related technical indicator. Hence the exchange is done based on the trends that occur.
2. Arbitrage opportunities: another approach to selling is when there are two markets, purchase the stock from the market which is at a lower price, and sell it at the second market

for a higher price, hence generating a profit. Designing an algorithm that can perform this operation creates numerous beneficial occasions.

3. Mean revision: this is an approach where an investor defines a range between which the prices must fall. Hence, *defining* a price range will allow for the exchange to happen accordingly whenever the price falls within the defined criteria.
4. Volume-weighted and Time-weighted average price: These are two strategies that are formed by dividing a bigger *order* and using a volume that is specific for the stock in case published to the market in smaller increments by ensuring the timing and volume is split into historically appropriate groups.

Algorithmic trading is an essential part of the stock market world since automation of the needed tasks to trade has improved their performance significantly (Nuti). Therefore, helping investors be more efficient

Chapter 2 Machine Learning

With the tendency to move repetitive tasks to machine-designed processes, machine learning appears to be the best progressive move. This method involves making decisions by considering three critical aspects of any exchange: time, price, and volume. Hence, machine learning is the best way to go about such tasks (Choudhry). The tendency is to have a pre-programmed machine learn and execute trading decisions on its own. The applications of machine learning in the recent past that have been explored are:

Stock price forecasting is predicting the price of a share of stock is a challenge. Even though for a long time it has been believed, and it still is at times, that the changes in the market situation are just a random walk, technical analysis assumes that all the necessary information is stated in a company's recent prices. Therefore, going by the belief that the price reflects the company's financial well-being, one can notice a trend, and therefore, prices can be forecasted. Stock market prices are affected by a wide range of factors, such as political situation, company progress, new consumer trends, credit rates, and community events. Despite all these factors, machine learning can be divided based on its models. The earlier models used were LSTM, ARIMA, and Support Vector Machine. Recently, traders have preferred Neural Networks for their greater capabilities (Hsu).

2.1 Portfolio Optimization

Depending on the investor's priorities, such as maximizing profit or minimizing risk, one can select a group of stocks accordingly. The issue with selecting these portfolios manually is that they generate an enormous possibility of outcomes, which makes it unappealing when it is done by a human (Tola). However, it is a wonderful opportunity for managing risk when it is done automatically by a computer algorithm.

2.2 Sentiment Analysis

Sentiment Analysis has a greater impact because of the development of social media. In the earlier days, the newspapers were the primary source of information about anything new

happening, whether it was a new product developed by a company, a new car model designed by Ford, or General Motors, some great or pessimistic news (Chan). Then it became the TV, and with the entrance of the internet into modern lives, the news has gotten quicker at catching public attention. In the machine world, this information is obtained using Natural Language Processing (Luccioni). Even though the public may be guided by the news alone, for a more precise outcome, in the world of data scientists this is an addition to the information coming from the numbers, rather than a factor alone.

Chapter 3 Reinforcement Learning

In this section, we will present the *field* of Reinforcement Learning. We will start with the traditional model for sequential decision making, the model in which one decision impacts all the following options and results (Ni). Afterward, we will transition toward more modern solutions such as Dynamic Programming. To be efficient at solving the above-mentioned problems, the solution comes in the fashion of state-of-the-art algorithms like Neural Networks.

3.1 Markov Decision Process

MDP or Markov Decision Process are supposed to be a simple arrangement to formulating a problem by interchanging operations to realize a final target with the help of an object called agent (Feinberg). The entity the interaction is happening with that consists of all the elements except the agent is called the environment. In the perpetual process of the agent interacting with the environment, the agent is given rewards based on its responses to the different scenarios given by environment, where the agent is constantly strives to maximize the rewards.

3.1.1 Agent-Environment Interface

The interaction between agent and environment happens in the form of a clearly defined progression of occurrences based on time (Wang). For instance, at each subsequent time t = 0,1,2,3, the agent gets one depiction of the surroundings and records its corresponding reaction. Afterward, the agent is numerically rewarded based on its action, which puts it in a new state. Hence, in the end, the model creates a succession that is noted as follows: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

The following schema provides a visual representation of the above-mentioned method.

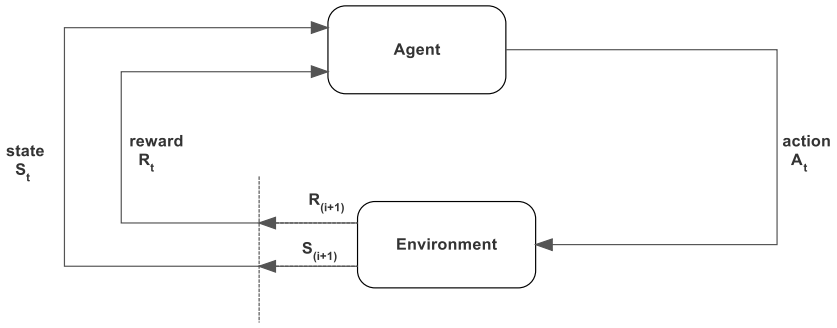

Figure 1:Schema of the agent-environment interaction in MDP (Sutton)

In the case of a defined MDP, the elements are well defined, in which case the random variables are defined accordingly based on their anterior state and action.

3.1.2 Rewards And Returns

When it comes to reinforcement learning, the main purpose of the agent comes in the form of an incentive or reward given by the environment to the agent. A reward is a real number given by the environment to the agent. Hence, the purpose of the agent is to boost the aggregated amount of the rewards. By adjusting the incentive in a clear way of what our final goal is, we can set up the environment to reach our goal by the agent maximizing its rewards (Wen). Hence, the critical need to be specific in the goal determination.

We can interpret the return as some specific function of the sequence $G_t = f(R_{t+1}, R_{t+2}, \dots R_T)$, given *T* is to be considered as the final time step.

$$G_t = R_{t+1} + \cdots + R_T = \sum_{k=t+1}^{T} R_k$$

Hence this method is valid for situations in which the final time step T exists, or when the exchange between the agent and environment can be divided into smaller chains called episodes, the ending of each episode is known as terminal state St which is thereafter accompanied by a readjustment to the beginning position. Following is the new episode, which is *independent* of the preceding one. Hence, the idea is that episodes finish within the same state, however, with different rewards depending on the result. These tasks are known by the name of episodic tasks. The process of discounting is a process where the agent purposefully chooses actions that maximize the discounted reward. To be more specific, the agent would pick so that the expected discounted return is at its maximum. This process is needed to prevent the scenario where the

agent-environment interaction does not break on its own, avoiding a moment where T would reach infinity.

$$G_t = R_{t+1} + \gamma R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=t+1}^{\infty} \gamma^{k-(t+1)} R_k$$

The discounted rate is called when $0 \leq \gamma \leq 1$.

If the reward chain is linked, the infinite sum has a finite value, and in the worst-case scenario where $\gamma = 0$, the agent's job is only maximizing immediate rewards as soon as it becomes closer to 1, the goal shifts onto future returns as well (Hu).

It can be beneficial to establish a notation that would cover both finite and infinite cases at the same time. The following can be done if we view the end of the episode as a new start (Seijen).

3.1.3 Policies And Value Functions

The primary purpose of the algorithms that are determined by the expected return in reinforcement learning is to determine the benefit of the agent in a particular state, as the rewards to be obtained are based on policies. Formally, a policy is a formulation from states and actions to the likelihood of the agent choosing to take those specific actions $\pi : S \times \mathrm{A} \rightarrow [0,1]$. In other words, if the agent goes along with the policy π at time t, then $\pi_t(a|s)$ becomes the probability of $A_t = a$ will be the choice when $S_t = s$. (Ding).

Reinforcement learning defines how an agent's policy will change depending on the experience it has gained. Hence, the specific policy will be attributed to a specific value. Therefore, we can define a policy π with a function with value v_π. The expected return when starting in *s* followed by a policy π . Hence,

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$$

It is important to keep in mind that the value of the final state would always be 0. The same way, the expected return of the action *a* taken in the state *s* following policy π

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

Where q_π is the action-value function for policy π . Hence, the relation

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

$$\Rightarrow v_\pi(s) = \sum_{a\in A} \pi(a|s)\mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

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

Is satisfied. It is important to note that even the recursive relationship should still be satisfied.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

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

$$\Rightarrow v_\pi(s) = \sum_{a\epsilon A} \pi(a|s) \sum_{a\prime\epsilon S} \sum_{r\epsilon R} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s']]$$

$$\Rightarrow v_\pi(s) = \sum_{a\epsilon A} \pi(a|s) \sum_{a\prime\epsilon S} \sum_{r\epsilon R} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

This is known to be the Bellman equation for the action value function.

3.1.4 Optimality

The goal of reinforcement learning is to determine a policy that will accomplish the task and gain the most rewards. In the case of finite MDPs value functions are defined as a policy π would be a better than or equal to a policy π' if its return is greater than or equal to that of π for all states (Bhandari). Hence, $\pi > \pi'$ if and only if $v_\pi(s) > v_\pi'(s)$ for all optimal policy which are noted as π_*. All these policies have a common function, known as state-value function.

$$v_*(s) = \max_\pi v_\pi(s)$$

For all $s \in S$ and $a \in A$.

The value function of a policy must satisfy Bellman's equation unity for the state values. However, the value function may be expressed in a different form, excluding any reference to a particular policy. Hence, the Bellman optimality equation demonstrates that given an optimal policy, the state value will equal the expected return in the case of the best action of that state.

$$v_*(s) = \max_a q_*(s,a)$$

$$\Rightarrow v_*(s) = \max_a \mathbb{E}[G_t|S_t = s, A_t = a]$$

$$\Rightarrow v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$$

$$\Rightarrow v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]$$

$$\Rightarrow v_*(s) = \max_a \sum_{s\prime\epsilon S}\sum_{r\epsilon R} p(s', r|s, a)[r + \gamma v_*(s')]$$

Similarly, when it comes to action-value function

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

$$\Rightarrow q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, +A_t = a]$$

$$\Rightarrow q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]$$

$$\Rightarrow q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a]$$

$$\Rightarrow q_*(s,a) = \sum_{s\prime\epsilon S}\sum_{r\epsilon R} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')]$$

An optimal policy is any policy where a nonzero probability can be assigned (Singh). In other words, it is a one-step search or when v_* is known the actions which are best suited after a one-step search are thought to be optimal actions. The optimal policy is easily determined if v_*, is known, also in Bellman optimality equation one will find more than one action where maximum can be obtained. Finding an optimal action when we know q_* is an easier task because now the agent can eliminate the step of doing a search ahead. Instead, the agent can find those actions that increase $q_*(s,a)$ because the above-mentioned value will determine the value in an efficient manner. Therefore, in the long run we will get values that are available straight away.

One of the draw backs of using the Bellman optimality equation is that the corresponding solution is useful in very few cases, due to the many options when it comes to searching and the corresponding probabilities (Rincon-Zapatero). Furthermore, when it comes to practice few assumptions have to be true for the solution to work the three assumptions are: the dynamics of the environment are well known, sufficient resources to fulfill the computation, and lastly the

system must satisfy Markov property. In a perfect world where all three of these assumptions are true it is easy to implement Bellman's optimality equation. However, in practicality, one of the three is usually not satisfied. On the other hand, other reinforcement learning methods are thought to be close to Bellman's equation, the only difference being that the methods tend to use actual experienced transition and not expected transitions.

3.2 Dynamic Programming

Dynamic Programming refers to an algorithmic technique that breaks a problem into multiple subproblems and finding the optimal solution to each of the smaller problems, hence at the end we have a perfectly optimal solution for the problem (Eddy). The importance of the dynamic programming cannot be underlined enough as it is at the core of the theory. However, given the computational expense they require, one may agree they are of very little use when it comes to reinforcement learning.

Policy evaluation is the calculation of the v_π for a policy π. If the dynamics of the environment are fully known, then we can use Bellman equation for a system of $|S|$ simultaneous linear equation in $|S|$ unknowns *(the v_π (s), s ∈ S)* (Cappen). Hence, the solution we can implement is a simple calculation. However, we must consider for the purpose of our case an iterative method may be more appropriate. Suppose a case of value functions v_0 , v_1, v_2,…, each mapping S^+ to R. The beginning value v_0 is any randomly chosen value, compared to the ending state where a value of 0 must be assigned. To update the algorithm, we will use Bellman's equation.

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) = \sum_{s' \epsilon S} \sum_{r \epsilon R} p(s', r|s, a)[r + \gamma v_k(s')]$$

for all $s \in S$. The value of each step is updated at every repetition to the new value function. Bellman equation guarantees the equality $v_k = v_\pi$. Therefore, the above-mentioned approach is implemented till the point where the value function estimate exceeds the threshold θ.

Afterwards, the value function can be implemented to reach policy improvement. Such as, incase when we need to find the value function v_π for a policy π. Sometimes, the state s may require a different approach, such as the policy may need to be able to make a choice using deterministic approach to find an action. In other words, we consider choosing a in s and then following an established policy π. Here, we have a value that is defined as $q_\pi(s,a)$ and an important step, to

check if it is greater or less than v_π(s). In the case that it is greater, we can benefit more by choosing a once in s and then follow π, rather than following π constantly. One obvious progression is to keep in mind all possible choices and the corresponding actions but choose the best fitting action given $q_\pi(s,a)$. Hence, we appraise a greedy policy π' in the case of

$$\pi'(s) = \text{argmax}_a\ [q_\pi\ (s,a)]$$

$$\Rightarrow \pi'(s) = \text{argmax}_a \left[\sum_{r\in R}\sum_{s'\in S} p|(s',r|s,a)[r + v_\pi\,(s')]\right]$$

here $argmax_a$ is value a where the next expression is maximized. In the case when the new greedy policy π' is as good as, however not better than the old policy π, in which case $v_\pi = v_{\pi'}$, then for all $s \in S$:

$$v_{\pi'}(s) = \max_a\ [\mathbb{E}\,[R_{t+1} + \Upsilon v_{\pi'}(S_{t+1})|S_t = s, A_t = a]]$$

$$\Rightarrow v_{\pi'}(s) = \max_a\ [\textstyle\sum_{r\in R}\sum_{s'\in S} p|(s',r|s,a)[r + v_\pi\,(s)]]$$

Because the above is similar to Bellman optimality equation, v_π, must be v_*, and both π and π' should be optimal policies. As a rule, when it comes to policy improvement it means that the new policy must be better than the old one (Engel).

One can obtain a series of upgraded policies and value functions if policy π is improved and gives us a better π', hence we can determine $v_{\pi'}$ and find a even better π'' and so on. This way we can eventually find the most optimal policy π.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

Here, $\xrightarrow{E}$ stands for policy evaluation and $\xrightarrow{I}$ is for policy improvement. Going from the theory that MDP has a finite number of policies, and that each individual policy promises to be better than the

previous one, the whole process must reach eventually to an optimal policy and optimal value functions, a process called policy iteration.

Algorithm 1: Policy Iteration:

Inputs: θ

Initialize Policy function π(s) with random values

Initialize value function V(s, a) with random values

Policy Evaluation:

While Δ > θ **do**

Δ← 0

for *each s* **do**

$\boldsymbol{\nu} \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} P\big(s', r \big| s, \pi(s)\big)[r + \gamma V(s')]$

Δ ← max (Δ| $\nu - V$ (s))

end

Policy Improvement:

Policy stable ← *True*

for *each s* **do**

old_action ← $\pi(s)$

$\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} P\big(s', r \big| s, \pi(s)\big)[r + \gamma V(s')]$

Δ ← max (Δ| $\nu - V$ (s))

If $\pi(s) \neq$ old_action then

Policy_stable← *False*

end

end

3.3 Temporal-Difference Methods

The temporal difference methods are a mix between Monte Carlo and Dynamic Programming methods. With the difference that this mix acquired the ability to learn straight from the unpolished

material and without the need for knowledge of the environment's dynamics. As well as the TD methods' capability to make updates without having to wait for the final outcome, but rather using learned estimates.

MC method lacks the ability to make updates on their own, but rather it needs the knowledge of $V(S_t)$ to be an estimation of $v(s)$ where we have the expected return G_t be the target.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

3.4 Exploration vs. Exploitation

The policy evaluation problem for action values is to estimate q (s, a), that is the expected return in state s, taking action a, and following policy p. In an episode we consider visiting a state action pair (s,a) when the state s is visited and action a has been taken. However, not all pairs will be visited, leaving some of the pairs unexplored. In the case of policy evaluation, it is important to facilitate perpetual exploration (Auer).

Hence, we arrive at a point where we call the situation in general terms exploration vs. exploitation trade-off. In this situation we can prove the need to balance the exploitation of those behaviors which give the most outcome while exploring new behaviors that are not yet known to the agent. One common technique to achieve the encounter of all state-action pairs is the use of the policies which are stochastic and have a nonzero possibility of choosing all actions in every state (Singh). With that stated, the policy will select the best-known action while exploiting. However, there will be scenarios in which the policy will choose an action that is random while exploring. The two ways to ensure this are: On policy methods and Off policy methods.

On policy methods are the policies that attempt to evaluate and improve the policy used to make decisions. In the case of an on-policy methods the policy is the one that is soft, or where $\pi(a|s) > 0$ for all $s \in S, a \in A$, but slowly moved closer to create an optimal policy. One such example is the ϵ- greedy policies, which theoretically they would prefer an action that have maximum estimated action values, but with a probability ε they rather chose an action at random. With that said, all nongreedy actions will have a minimal probability of choosing $\epsilon|A|$. The remaining portion of the probability $1 - \varepsilon + \varepsilon/|A|$ will be given to the greedy action. We can be certain that we will develop a better policy by using the concept of a greedy policy for ε-soft policies.

However, we need to be aware that one can only find the best policy among the ε-soft policies not the best policy.

Off policy methods are the ones that improve or evaluate a policy the policy that is different from the policy used to acquire the data. The on-policy approach is a compromise due to the ability to learn the action values for the near-optimal policy rather than for the optimal policy.

The best technique is to rather use a combination of the above two mentioned policies. In that case we have a policy for generating behavior and becomes the behavior policy and another one for which the method learns, hence it becomes the optimal or the target policy. In a classic scenario the target policy will be a deterministic greedy policy, but the action value function will be given by the behavior policy.

3.4.1 On-Policy Methods

The simplest scenario where one can see the application of the TD idea as an on-policy method is the SARSA algorithm (Rummery). The update rule in this scenario is executed after every alteration of the nonterminal state S_t.

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

In the scenario that S_{t+1} is the terminal state, then Q (S_t+a…) would be set to 0. Hence, the earlier mentioned rule uses every element of the five-folded equation ($S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$)known to be the conversion of one state-action pair to another. This five-folded equation is the reason behind the name of the algorithm, SARSA (Zou). The perpetual estimation of q_π for a policy π is done simultaneously with π's change regarding greediness when considering for q_π.

Hence, we arrive at the following algorithm:

Algorithm 2: SARSA

Inputs: M, ε,α

Initialize action value function $Q\ (s, a)$with random values

for $episode = 1 : M$ **do**

Initialize s_t

for $t = 1{:}\,T$ **do**

With probability ε select random a_t, otherwise $a_t = \text{argmax}_a\ Q\ (s, a)$

Execute action a_t and observe reward r_{t+1} and s_{t+1}

With probability ε select a random a_{t+1} , otherwise, $a_{t+1} = \text{argmax}_a\ Q\ (s, a)$

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \Upsilon Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$

$s_t \leftarrow s_{t+1}$

$\alpha_t \leftarrow \alpha_{t+1}$

end

end

3.4.2 Off-Policy Methods

Alternatively, an off-policy method can also be created by modifying the update rule

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

Here the Q is approximating q*. In this case q* is the optimal action-values function which lacks dependence from the policy that follows it, therefore it is an off-policy algorithm (Wulfmeier). Nevertheless, the effects of the policy are still important as it establishes the state-action pairs to be visited and therefore updated. We show the details in the following algorithm.

Algorithm 3: Q-learning

Inputs: M, ε, α

Initialize action value function $Q\ (s, a)$with random values

for $episode = 1 : M$ **do**

Initialize s_t

for $t = 1{:}\,T$ **do**

With probability ε select random a_t, otherwise $a_t = \text{argmax}_a\ Q\ (s, a)$

Execute action a_t and observe reward r_{t+1} and s_{t+1}

With probability ε select a random a_{t+1} , otherwise, $a_{t+1} = \text{argmax}_a\ Q\ (s, a)$

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \Upsilon max_a Q(s_{t+1}, \alpha) - Q(s_t, a_t)$

$s_t \leftarrow s_{t+1}$

end

end

This algorithm's limitation is because of the implication of the maximization when the target policies are defined, it may lead to a great bias for the Q-values and therefore negatively impacting the algorithm and increasing the likelihood of creating a suboptimal policy (Icarte). We can run into this problem because we use the sample for both determination of the maximum action and for the estimation of the value. To solve the mentioned issue, we can refer to the idea of Double Q-learning. By using this approach, we effectively divide the samples in two subgroups. Following we use these subgroups to learn two independent estimates Q1(a) and Q2(a) respectively. Each of these are estimates of the true value q(a) for all $a \in A$. Afterwards, we can establish the maximizing action $A^* = argmax_a Q_1(a)$ and Q_2 correspondingly to come up with an estimate of the value $Q_2(A*) = Q_2(argmax_a Q_1$ (a)). This estimate will therefore be unbiased meaning

$$\mathbb{E}[Q_2(A^*)] = q(A^*).$$

This process can be replicated by reversing the estimates so that we get a second unbiased estimate $Q_1(A^*) = Q_1(argmax_a Q_2(a))$ in this way the two approximate value functions will be treated in a symmetrical way therefore removing the bias.

Algorithm 4: Double Q-Learning

Inputs: M,ε,α

Initialize action value function $Q_1\ (s, a)$ with random values

Initialize action value function $Q_2\ (s, a)$ with random values

for $episode = 1 : M$ **do**

Initialize s_t

for $t = 1{:}\,T$ **do**

With probability ε select random a_t, otherwise $a_t = \text{argmax}_a\ Q\ (s, a)$

Execute action a_t and observe reward r_{t+1} and s_{t+1}

If $t\ \%\ 2 = 0$ **then**

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha[r_{t+1} + \Upsilon Q_2(s_{t+1}, argmax_a Q_1(s_{t+1}, \alpha)) - Q_1(s_t, a_t)]$$

else

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha[r_{t+1} + \Upsilon Q_1(s_{t+1}, argmax_a Q_2(s_{t+1}, \alpha)) - Q_2(s_t, a_t)]$$

end

end

3.5 State-Of-The-Art

What all the methods previously discussed have in common is that the data is arranged in a diagrammatic way, allowing us to apply it to problems that have a much larger scale. Nevertheless, numerous scenarios where we can apply Reinforcement Learning the state space is a mixture of different data setups, creating a place that is difficult to separate creating difficulties for reaching an optimal policy. Other problems that we could potentially encounter with large state spaces, is the possibility of finding states that have not been previously explored. Hence, we can elaborate a new function that is closely related to some of the previously already explored states, this is achieved through an occurrence called function approximation. The function approximation basically attempts to design a new function based on the ideas already explored from the previous functions (Freidman). Therefore, we believe function approximation to be a form of supervised learning.

A value-based algorithm estimates a state-action value function that guides the optimal policy. Q-learning approximates a Q value (expected return) by iteratively updating a Q-table, which works for problems with small discrete state spaces and action spaces. An actor-critic based algorithm combines the advantages of value based and policy-based algorithms. It updates two neural networks, namely, an actor network updates the policy (probability distribution) while a critic network estimates the state-action value function. (Liu). The state-of-art actor-critic based algorithms are deep deterministic policy gradient (DDPG), proximal policy optimization (PPO), asynchronous advantage actor critic (A3C), advantage actor critic (A2C), soft actor-critic (SAC), multi-agent DDPG, and twin-delayed DDPG (TD3) (Liu).

3.5.1 Deep Q-learning

The capacity to learn complex data and nonlinear functions makes Neural Networks a perfect candidate for becoming the go-to method when it comes to function approximation. The Q-learning algorithm's simplicity and optimality together with the ability to learn complex functions have created the perfect ecosystem for the evolution of a novel algorithm, Deep Q-learning (DQL) (Du). DQL focuses on the creation of a function $Q(s, a; \theta)$ constitute as a parametrized functional form with weights θ that approximates the action-value function q (s; a). To establish and improve the approximate solutions given by the weights θ we need to define a cost function $L(\theta)$. Here is where the mean-square loss definition comes in handy.

$$(L(\theta) = \mathbb{E}_{s,a\ \rho(.)}[(y_i - Q(s, a; \theta))^2]$$

Here $\rho(s, a)$ are the probability distribution of the state action pairs. We can then get the approximate solutions and by using the stochastic gradient descent we can calculate the optimal weights θ. Furthermore, by updating the target Y_t for the before mentioned weights, we can use the general gradient rule, given the weights approximate $q(S_t, A_t)$ this includes SARSA or Q-learning updates. Therefore, we can apply the general gradient-descent update rule.

$$\theta_{t+1} = \theta_t + \alpha[Y_t - Q(S_t, A_t; \theta_t]\nabla_{\theta_t} Q(S_t, A_t; \theta_t)$$

Despite the multitude of options for creating a Q-network to be able to approximate a q function, the ability to have a unique output representation for each possible action. Nevertheless, only the state representation become the input to the Neural Networks. In this method, the outputs we align

with the input values predicted by Q-values which have only one forward pass through the network. The ability to train the Q-network values allows us to reduce the sequence of loss functions $L_t\ (\theta_t)$ that otherwise would change at every iteration t. In this case $y_t = \mathbb{E}_{s'\epsilon\mathcal{E}}[r + \gamma max_{a'} Q(s', a'; \theta_{t-1})|s, a]$ is the target for iteration *t*, and ε is the emulator of the environment used to train the network. By applying the update rule of Q-learning, $Y_t = R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a;\ \theta_t)$ as well as, applying the back-propagation technique we arrive at the optimal utilization of the DQ Learning algorithm (Mnih). The algorithm learns the greedy policy where $a = argmax_a Q(a, a; \theta)$ essentially making it an off-policy method, however, it still follows a behavior distribution that allows adequate exploration. Following is the algorithm

Algorithm 5: Deep Q- Learning (DQL)

Inputs: N, M, ε, C

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

for $episode = 1 : M$ **do**

for $t = 1{:}\,T$ **do**

With probability ε select a random a_t, otherwise $a_t = \text{argmax}_a\ Q\ (s_t, a; \theta)$

Execute action a_t in emulator and reward r_{t+1} and s_{t+1}

Store transition $(s_t, a_{t,} r_{t+1,} s_{t+1}\ )$ in D

Store transition $(s_j, a_{j,} r_{j+1,} s_{j+1}\ )$ in D

Set $y_j =$

$$\begin{cases} r_{t+1} & for\ terminal\ s_{t+1} \\ r_{t+1} + \Upsilon \max_a Q\ (s_t, a; \theta) & for\ non\ terminal\ s_{t+1} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(s_{t,}, a; \theta\right)\right)$ ^2

end

end

3.5.1.1 Experience Replay

For the effectiveness of the training of the Q-networks training, we must consider the concept of Experience Replay. When it comes to tabular Reinforcement Learning, the updates are executed online (Yin). The updates are done in a fashion that when each new transition done the value function is updated. The requirement for immense amounts of data makes for this method to be optimal and functional policy makes this method less desirable. If we look deeper at this method, we will notice that essentially, the weight adjustment for one pair will create a cascade effect on other pairs in the state-action space. In this technique the agent will store the experience for every time step $e_t = (s_t, a_t, r_t, s_{t+1})$ which therefore help us construct the dataset also known as replay memory $D = \{e_{0,}, e_1, \ldots\}$. We will use the samples randomly picked for the training portion of the learning step, rather than utilizing the current state, because this approach gives us the opportunity to a more efficient data since the experience can be used in other updates as well. This approach

gives samples the freedom to avoid being correlated and more random, hence the variance of the updates will inevitably be reduced.

By giving priority to the most suitable experiences, we can boost this method, because the in this scenario we would keep away from the expensive searches through a larger amount of memory as we can just shift our attention to a specific portion. Hence, the importance of every transition is defined by the main portion of the experience reply. In a perfect scenario this portion is to be the amount that an agent can learn, however its seldom availability, makes it difficult to calculate. Therefore, we can approximate it by using δ of TD update that gives us the distance from a value to the next step (Woergoetter). The probability of the sampling transition I is defined in the following equation:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}$$

Here p_i is the priority transition where i and a are the ones to control the amount of prioritization to be used. The two variations of this are $p_i = |\delta_i| + \varepsilon$, with ε being a small positive constant which stops the edge case of from being revisited in case the update is 0. We consider prioritizing the cases by rank such as $p_i = 1/\text{rank(i)}$, here rank(i) is the rank of transition i when the memory is sorted conform $|\delta_i|$.

3.5.1.2 Target Networks

Considering the need for further improvement of the method's stability in the Q-learning approach, we can investigate the utilization of a different network that we can use to obtain targets y_i. In other words, we will replicate the Q-network after C updates which basically gives us the ability to get to the target network Q'. Therefore, enabling us to get the Q-learning targets $y_i$for the new C updates to Q prior to the new duplication. These actions are required because they make the algorithm more robust. Otherwise, we risk having an update that increases $Q(S_t, A_t)$that leads to an increased $Q(S_{t+1}, a)$for all a. As a result, the target y_t also increases creating the risk of a discrepancy between policies. A possible solution to this problem that would ensure these discrepancies don't happen would be to add targets that have older sets of parameters which eventually would retain the period between a Q update and its subsequent effects on y_i. Bellow, we show the algorithm

Algorithm 6: Deep Q- Learning with target Network (DQL)

Inputs: N, M, ε, C

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-values function Q with random weights $\theta' = \theta$

for $episode = 1 : M$ **do**

for $t = 1{:}\,T$ **do**

With probability ε select a random a_t, otherwise $a_t = \text{argmax}_a\ Q\ (s_t, a; \theta)$

Execute action a_t in emulator and observe r_t and s_{t+1}

Store transition $(s_t, a_{t,} r_{t+1,} s_{t+1}\ )$ in D

Sample random minibatch $(s_j, a_{j,} r_{j+1,} s_{j+1}\ )$ from D

Set $y_j =$

$$\begin{cases} r_{t+1} & for\ terminal\ s_{t+1} \\ r_{t+1} + \Upsilon \max_a Q\ (s_t, a; \theta) & for\ non\ terminal\ s_{t+1} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(s_{t,}, a_t; \theta\right)\right)$ ^2

Every C step align networks with $\theta' = \theta$

end

end

With the help of Double-Q-Learning we could achieve even better results. Q-learning by itself is not a perfect method due to the likelihood of overestimating the q-values. If we are to separate the maximization operation into action selection and action evaluation (Jang). Hence, we propose to assess the greedy policy considering the online network Q, all while utilizing the target network Q' to estimate its value. Doing so allows us to decrease the chances of overestimation.

$$Y_t = R_{t+1} + \gamma Q'(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta'_t)$$

Here, by changing between the weights we can upgrade θ and θ'.

Above, we display in detail the algorithm.

3.5.2 Related Work

There are many existing works on Deep Reinforcement Learning in quantitative financial tasks. The work of Moody & Saffell implemented a policy search for stock trading. Deng et al. showed that DRL can obtain more profits than conventional methods. Nan *et al,* Vadori *et al,* Yang *et al,* Zhang *et al*, talk about more applications that include stock trading because of the lack of labeling data, they use traditional time series stock price data and combines it with news headline sentiments, while leveraging knowledge graphs for exploiting news about implicit relationships. Zhang *et al,* discuss futures contracts in their paper, Koratamaddi *et al* discuss alternative data (news sentiments). The algorithm designs trading strategies for continuous futures contracts. Both discrete and continuous action spaces are considered, and volatility scaling, Ganesh *et al* discuss high frequency trading, they propose very-long short term memory networks, or VLSTMs, to deal with such extreme length sequences. Bao *et al* discuss liquidation strategy analysis, their work builds the foundation for future multi-agent environment trading analysis. Secondly, they analyze the cooperative and competitive behaviors between agents by adjusting the reward functions for each agent, which overcomes the limitation of single-agent reinforcement learning algorithms. Finally, they simulate trading and develop an optimal trading strategy with practical constraints by using a reinforcement learning method, which shows the capabilities of reinforcement learning methods in solving realistic liquidation problems. and Buehler *et al* discusses a new application of reinforcement learning: to the problem of hedging a portfolio of “over-the-counter” derivatives under market frictions such as trading costs and liquidity constraints.

Chapter 4 Experimental Design

For this part we will use the industry notions and description we have previously elaborated to present a technique to solve or help in stock market guidance using methods previously described. To do this we have used the Jupyter Notebook. We have run the code using different agents to determine the best agent among them. The data we used has been downloaded from GitHub and it encompasses 6 years' worth of data between the years 2013 and 2019. We chose 20 stocks to run our code on, those are:

- AAON, AAP, AMS, COOP, ETO, FANG, FCCY, FIX, GOGL, HCCI, LULU, LUNA, NQP, PSA, REED, SUMR, SUPN, TREE, VET, ZYXI

The actions one can do according to our findings are: Buy, Sell and Hold. We are going to explain each of them. An investor would Buy a share at the specified price and would calculate the profit from it. Sell if the share price is above the point it was bought at. Hold or wait for the next iteration before deciding to buy or sell. Since there are only a certain number of actions possible to execute, we are going to classify them accordingly and decide based on their logic.

Flat- one can keep the stock to remain Flat, or buy to become Long, or if choose to sell it becomes **Short.**

Short- one can hold to stay Short or buy if they want for it to become Flat, if sell to become Short.

The way we encourage the algorithm to perform according to our intentions in the stock market is by giving the agent a reward according to its performance defined by the returns. The reason behind the returns being a better measurement than the prices, it is because it is a more powerful tool that gives one the freedom to generalize between the stocks and compare them adequately.
We compute the positions we take in the following way:

If the agent doesn't have a position to take, the position taken is considered Flat, the reward at that time step is computed as:

$$R_t = 0$$

The agent will take a Long position if it will buy at time-step t_{entry} at price P_{entry} because of the expectancy of an increase in price.

$$R_t = \frac{P_t - P_{entry}}{P_{entry}}$$

If the expectation is that price will fall the agent will sell at time-step t_{entry} at price P_{entry}. This is considered the short position.

$$R_t = \frac{P_{entry} - P_t}{P_{entry}}$$

Actions

The actions that are possible to take are hold, buy, or sell.

Hold: the agent will do nothing, will just skip that time-step
Buy: the agent will open a buy the share and save it as *entry_price*
Sell: the agent will close the position, will sell, and calculate the profit

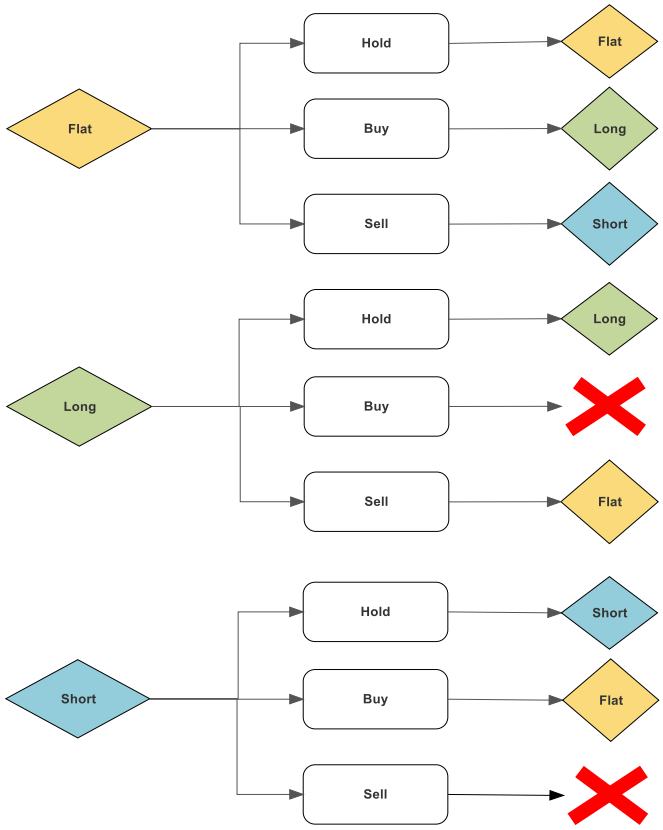

Figure 2:Schema of actions and positions, source: own

Only certain position and action combination are permissible. Hence, the above figure representation of the logic.

Flat- we can keep our position to stay Flat, if we buy, we become Long, and if we sell, we become short.

Long- keep to stay Long, sell to become Flat. However, an action to buy to stay Long will become a hold to stay Long.

Short- we can hold to stay Short, buy to become Flat, and if we sell to become Short it will be considered as hold to stay Short.

Agents –

After we have established all the details necessary for the environment, we need to test our data, we can move on to designing the agents. We are going to do so through the help of Deep Learning

techniques. We will look at Neural Networks which is basically a foundation composed of neurons that has been used mostly for its ability to incorporate the backpropagation algorithm.

The agents we have described in the thesis are:

Simple Moving Average Agent

Signal Rolling Agent

Policy Gradient Agent

Q-Learning Agent

Recurrent Deep Q learning Agent

We have chosen to use Recurrent Deep Q learning as it is one of the most popular baseline deep learning models in the automate trading.

To prove how the agents perform we have tested it on 20 stocks. Due to the limitation of space, we will present how the agents performed on 4 of these stocks. The stocks are: AAON, AAP, AMS and COO

Chapter 5 Results

We have used the Y axis for the price and X axis for the number of shares involved in a transaction. Most of the stocks seemed to have performed in the same way. While performing the experiments, we notice the general trend that the simpler methods compared to the more complex have less ups and downs markers. In other words, the frequency of the *buy* and *sell* signals are triggered similarly among stocks depending on the agent.

5.1 Simple Moving Average

The graphs show us that there were few events along the way. Despite the dips in the market, the graphs show a similar pattern.

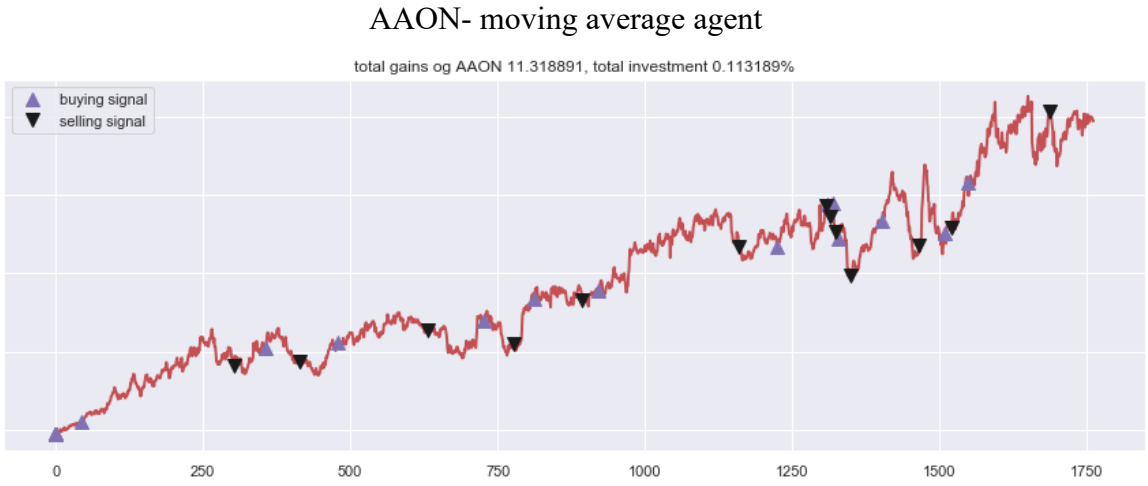

Figure 3: Representation of the AAON stock performance using moving average agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 11.3

AAP- moving average agent

Figure 4: Representation of the AAP stock performance using moving average agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -132.80

AMS- moving average agent

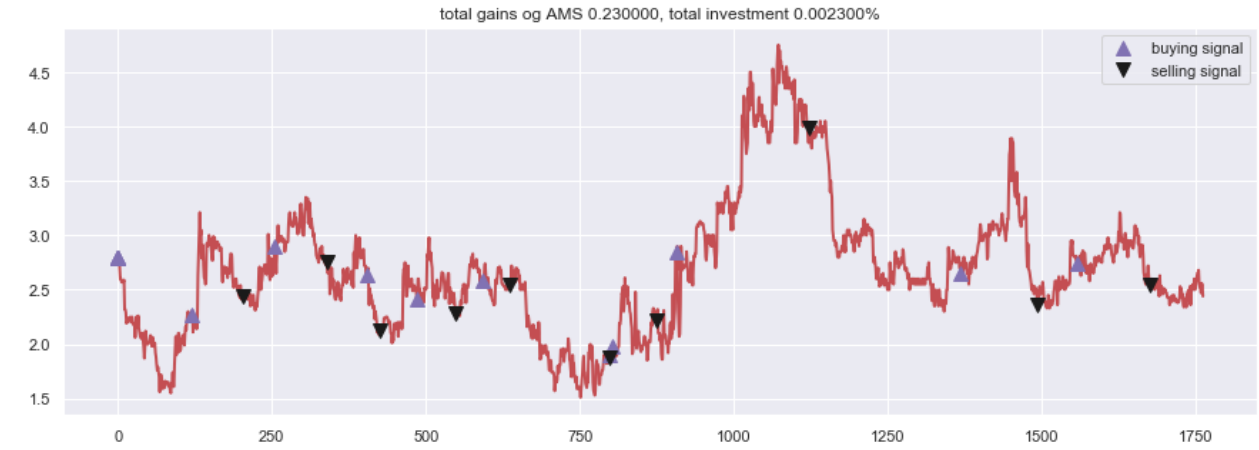

Figure 5:Representation of the AMS stock performance using moving average agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 0.23

COOP- moving average agent

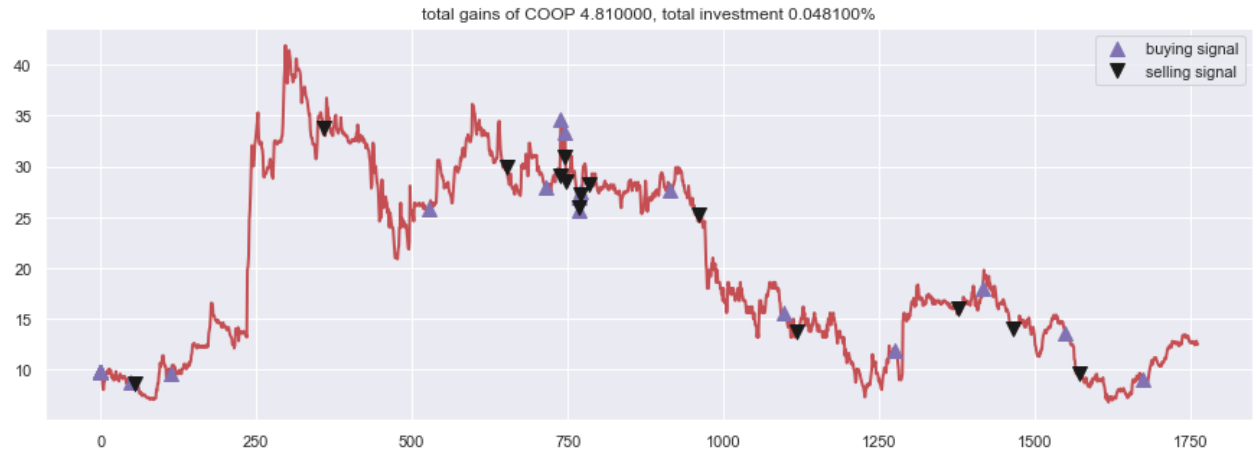

Figure 6:Representation of the COOP stock performance using moving average agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 4.81

5.2 Signal Rolling Agent

From the experiment we do notice how the agent is more sensitive to the environment and is performing much more buy/sell actions compared to previous agent.

AAON- signal rolling agent

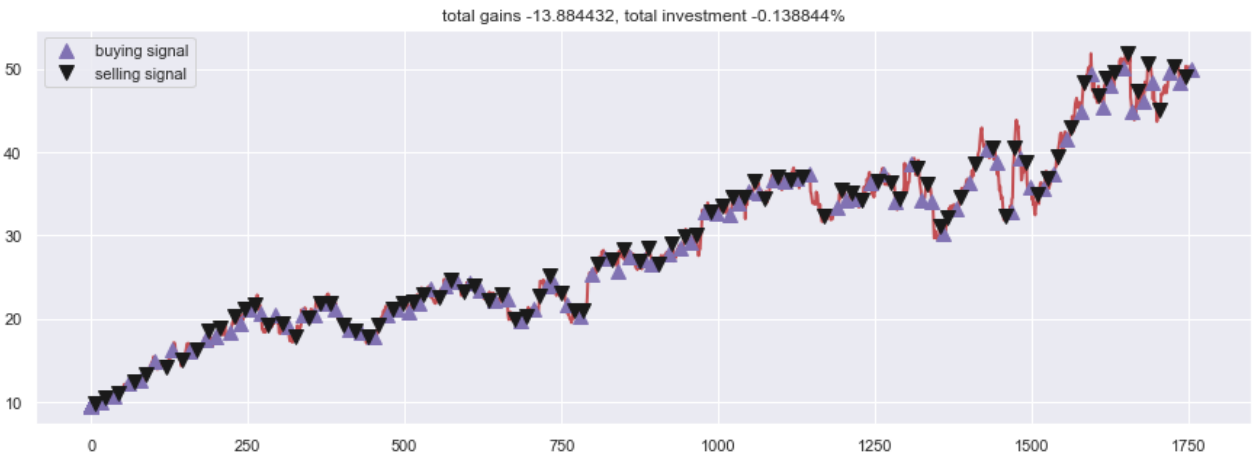

Figure 7:Representation of the AAON stock performance using signal rolling agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -13.884

AAP- signal rolling agent

Figure 8: Representation of the AAP stock performance using signal rolling agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 72.64

AMS- signal rolling agent

Figure 9:Representation of the AMS stock performance using signal rolling agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -1.52

COOP- signal rolling agent

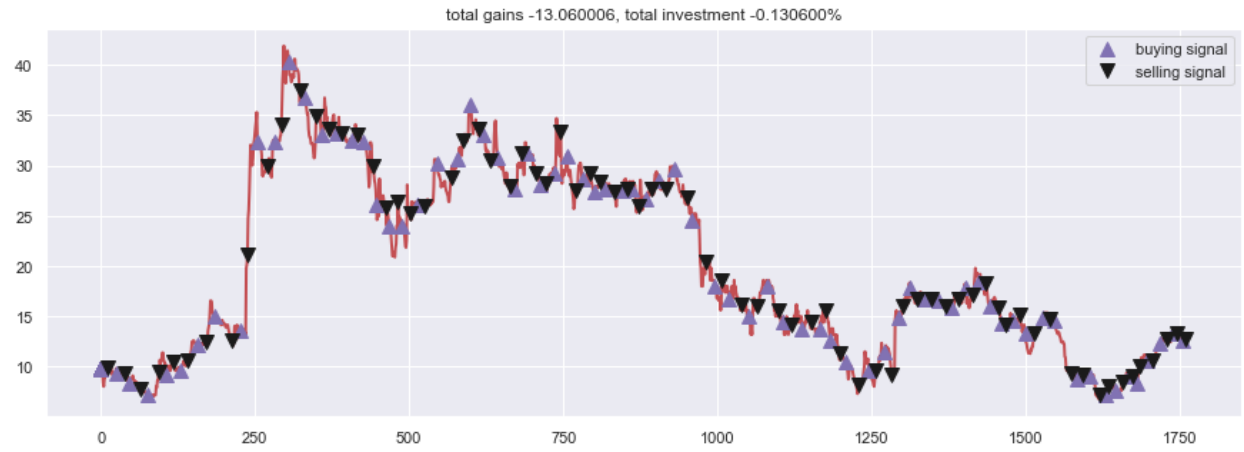

Figure 10:Representation of the COOP stock performance using signal rolling agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -13.06

5.3 Policy Gradient Agent

This agent is more sensitive to the changes of the market and is taking in to account more parameters making it more susceptible to the changes of the market. Among all stocks we noticed the sensitivity of the agent.

AAON- policy gradient agent

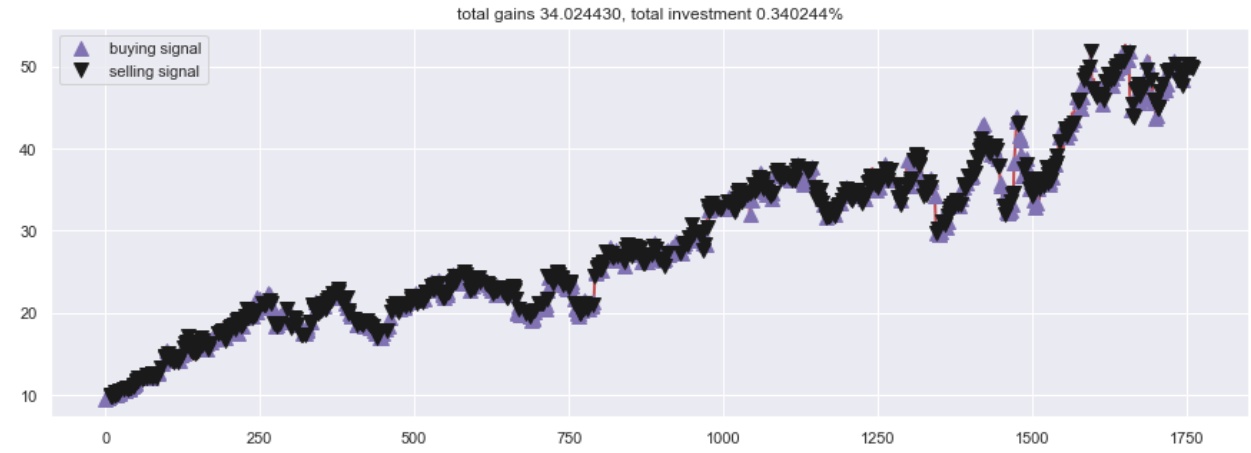

Figure 11:: Representation of the AAON stock performance using policy gradient agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 34.02

AAP- policy gradient agent

Figure 12:Representation of the AAP stock performance using policy gradient agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -2046.07

AMS- policy gradient agent

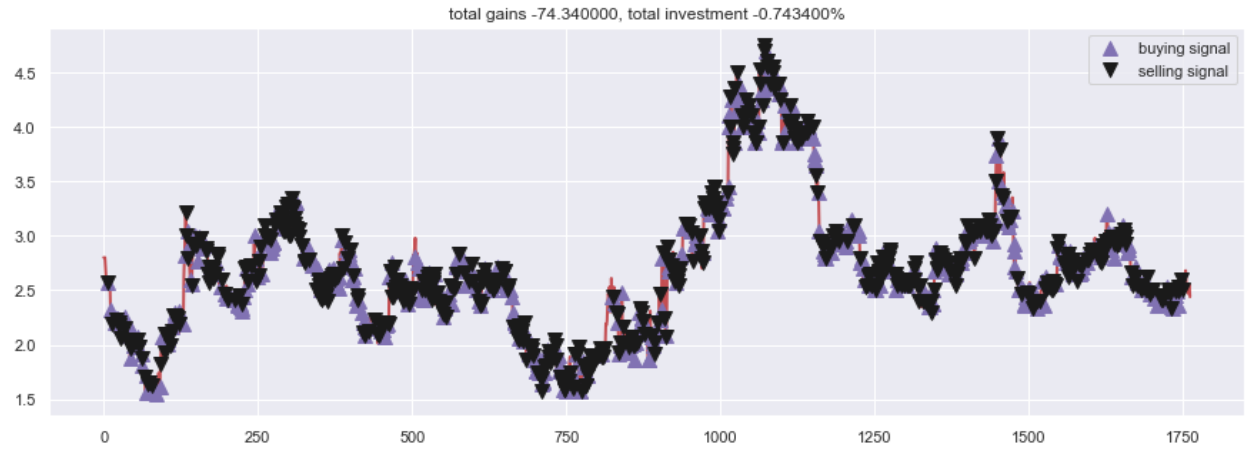

Figure 13:: Representation of the AMS stock performance using policy gradient agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -74.34

COOP- policy gradient agent

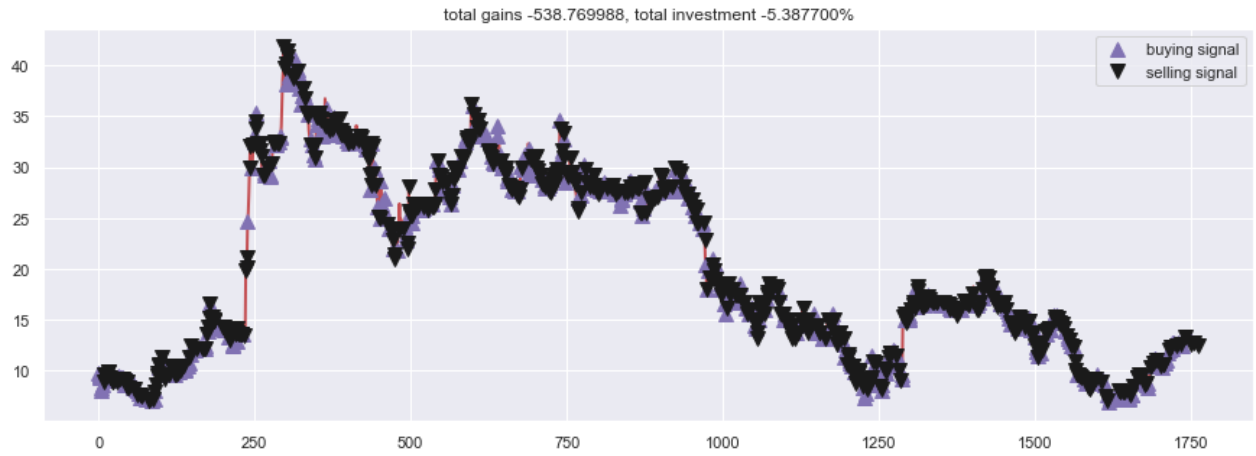

Figure 14:Representation of the COOP stock performance using policy gradient agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -538.77

5.4 Q-Learning Agent

The market changes reflect on the performance of the Q-learning agent. We can notice the trend by looking at the data plotted on the bellow figures

AAON Q-Learning Agent

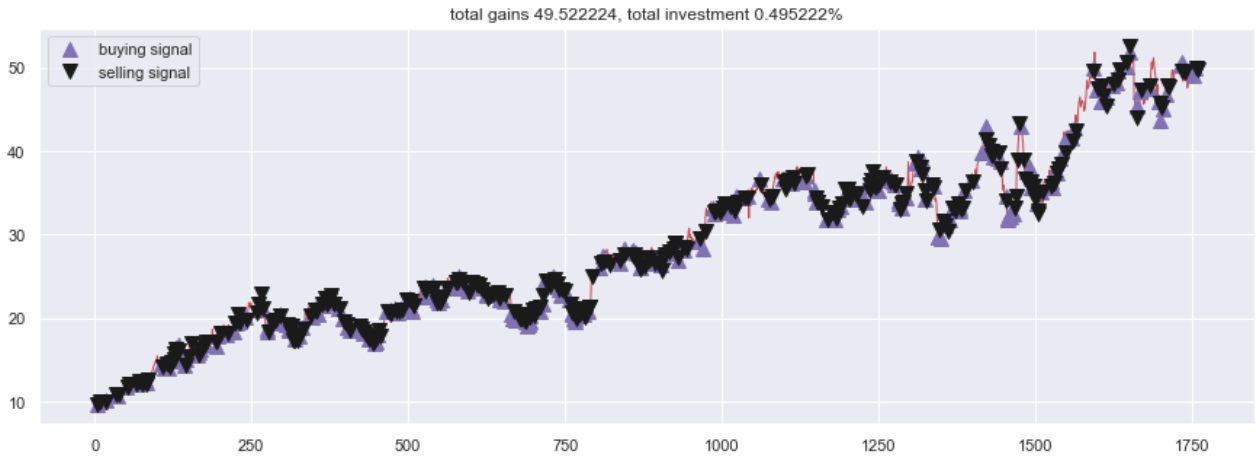

Figure 15:Representation of the AAON stock performance using Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 49.52

AAP Q-Learning Agent

Figure 16:Representation of the AAP stock performance using Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -253.33

AMS Q-Learning Agent

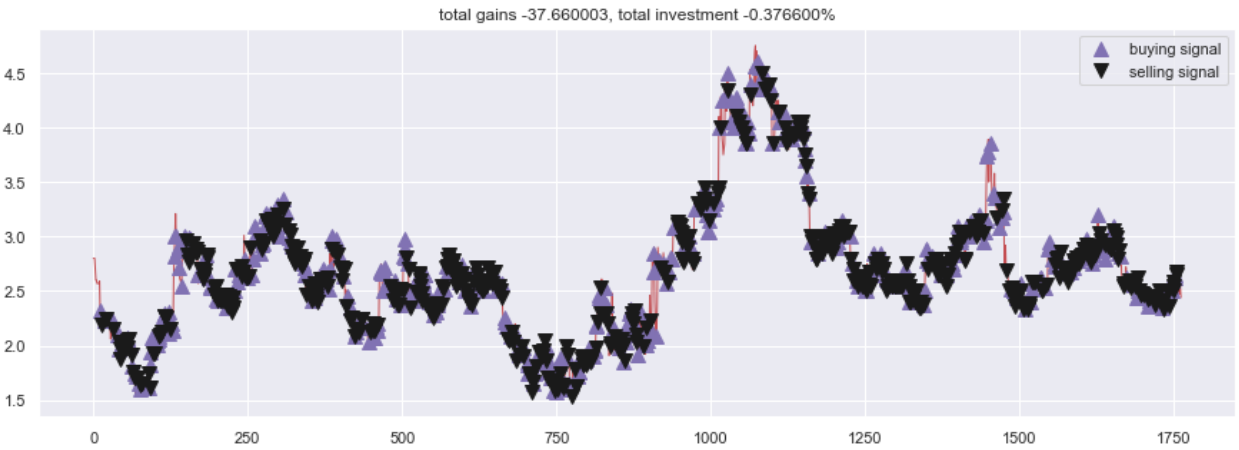

Figure 17:Representation of the AMS stock performance using Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -37.66

COOP Q-Learning Agent

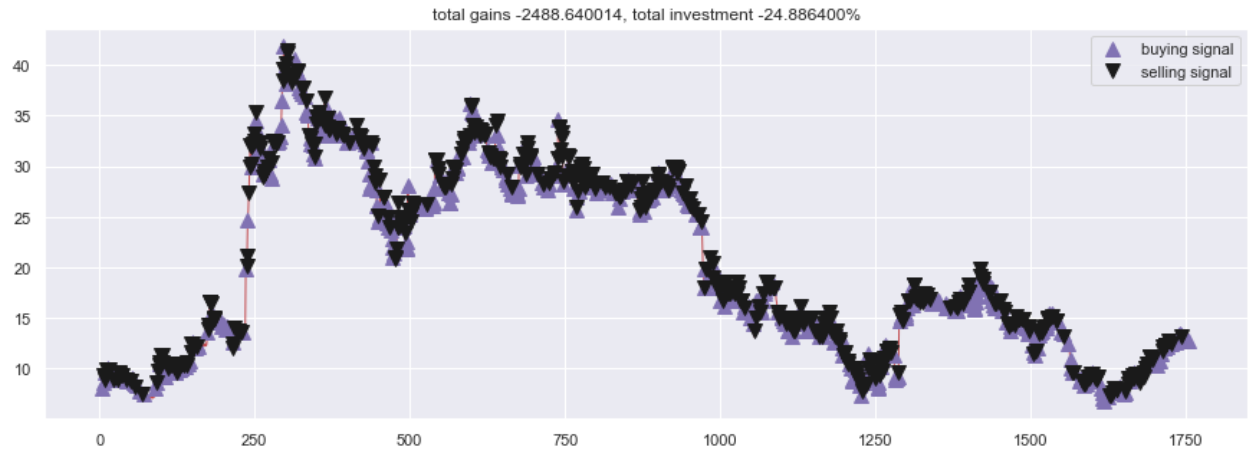

Figure 18:Representation of the COOP stock performance using Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -2488.64

5.5 Recurrent Q learning Agent

The most complex agent, however not as sensitive to market changes, we are further investigating this problem in the bellow tables.

AAON Recurrent Q-Learning Agent

Figure 19:: Representation of the COOP stock performance using Recurrent Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -17.236

AAP Recurrent Q-Learning Agent

Figure 20:Representation of the AAP stock performance using Recurrent Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -9956.99

AMS Recurrent Q-Learning Agent

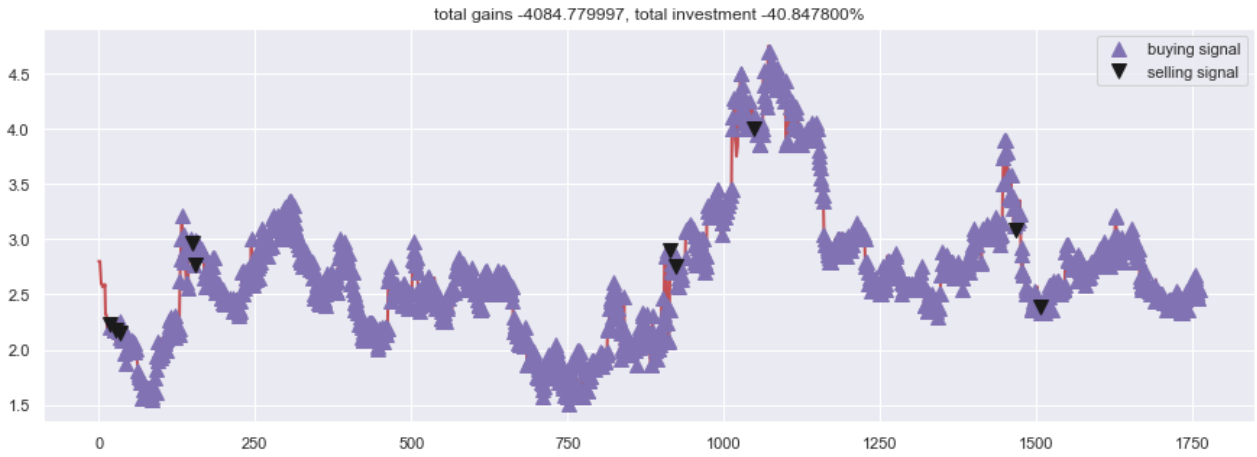

Figure 21:Representation of the AMS stock performance using Recurrent Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = -4084.80

COOP Recurrent Q-Learning Agent

Figure 22:Representation of the COOP stock performance using Recurrent Q-Learning agent. The black symbols denote selling signal, the blue symbols denote buying signal. Total Gains = 2.64

AAON			
Algorithm	**Total Balance**	**Total Gains**	**Total Investments (%)**
Moving Average Agent	10011.31	11.32	0.1132
Signal Rolling Agent	9986.11	-13.88	-0.1388
Policy Gradient Agent	10034.024	34.02	0.3402
Q-Learning Agent	10049.52	49.52	0.4952
Recurrent Q Learning Agent	9982.76	-17.23	-0.1723

Table 1: Results for AAON stock using the 5 agents

AAP			
Algorithm	**Total Balance**	**Total Gains**	**Total Investments (%)**
Moving Average Agent	9867.2	-132.79	-1.3279
Signal Rolling Agent	10072.64	72.63	0.7263
Policy Gradient Agent	7953.93	-2046.07	-20.4607
Q-Learning Agent	9746.67	-253.33	-2.5333
Recurrent Q Learning Agent	136.154	-9956.98	-99.5698

Table 2:Results for AAP stock using the 5 agents

AMS			
Algorithm	**Total Balance**	**Total Gains**	**Total Investments (%)**
Moving Average Agent	10000.23	0.23	0.0023
Signal Rolling Agent	9998.48	-1.52	-0.0152
Policy Gradient Agent	9925.66	-74.34	-0.7434
Q-Learning Agent	9962.33	-37.66	-0.3766
Recurrent Q Learning Agent	5915.22	-4084.77	-40.8477

Table 3:Results for AMS stock using the 5 agents

COOP			
Algorithm	**Total Balance**	**Total Gains**	**Total Investments (%)**
Moving Average Agent	10004.81	4.81	0.0481
Signal Rolling Agent	99986.93	-13.06	-0.1306
Policy Gradient Agent	9461.23	-538.76	-5.3876
Q-Learning Agent	7511.35	-2488.64	-24.88
Recurrent Q Learning Agent	9288.44	2.64	0.0264

Table 4: Results for COOP stock using the 5 agents

By looking at the above tables we can better understand, and prove our suspicion from the graphs section, which was that the more complex an agent is, the poorer it performs. There are multiple factors which could influence such a performance. Will describe these factors in more detail in the conclusion. The algorithms with higher frequency might imply that they are more sensitive to temporal changes in short time. However, high sensitivity might not always be advantageous due to a significantly increased transaction fee, which in real world would be considered.

Source code for the above performed experiments can be found at:

https://github.com/Magda123-blip/A-Comprehensive-Review-of-Machine-Learning-Methods-in-Stock-Market

5.6 Future Work

We can expand our analysis by incorporating Natural Language Processing. Integrating news and social media data, could give us a better sense of what the public thinks and how they feel about certain companies. Sentimental Analysis will give the investors the upper hand on knowing where to invest and data analysts will be able to design better algorithms to capture a larger portion of the market.

Chapter 6 Conclusions

Machine learning has been successfully applied to financial market prediction and investment. Several studies show that machine learning has great performance comparing with other methods. To achieve good generalization performance across a wide variety of financial products, stocks, and markets, the choice of models and training data matter. Therefore, the goal of this paper was to review the theoretical and practical factors that impact the stock market movement and price to better understand what impacts the market and how a data analyst should choose, process, and look at the data. We reviewed which auto trading algorithm performs better. We further investigate the advancement of AI applications in financial market, especially on the existing reinforcement learning and deep learning methodologies. In our study, we pre-process a fairly small-to-mid-scale stock market dataset and compare the models on the configuration commonly seen for individuals who often have very limited number of computational resources.

Among different agents that we have analyzed we can conclude that in our experiments more parameters and more complex models could lead to worse performance. One possible solution would be to have a larger set of training data, a better machine with a greater computational power. Another resolution is to train the model with a larger set of training data specifically when we choose the deep reinforcement learning models, to generate a more generalized mode

References

Abad, C., Thore, S. A., & Laffarga, J. (2004). Fundamental analysis of stocks by two-stage DEA. *Managerial and Decision Economics*, *25*(5), 231-241.

Aggarwal, R. K., & Wu, G. (2006). Stock market manipulations. *The Journal of Business*, *79*(4), 1915-1953.

Auer, P., Jaksch, T., & Ortner, R. (2008). Near-optimal regret bounds for reinforcement learning. *Advances in neural information processing systems*, *21*.

Bao, W., & Liu, X. Y. (2019). Multi-agent deep reinforcement learning for liquidation strategy analysis. *arXiv preprint arXiv:1906.11046*.

Bhandari, J., & Russo, D. (2021, March). On the linear convergence of policy gradient methods for finite mdps. In *International Conference on Artificial Intelligence and Statistics* (pp. 2386-2394). PMLR.

Buehler, H., Gonon, L., Teichmann, J., Wood, B., Mohan, B., & Kochems, J. (2019). Deep hedging: hedging derivatives under generic market frictions using reinforcement learning. *Swiss Finance Institute Research Paper*, (19-80).

Choudhry, R., & Garg, K. (2008). A hybrid machine learning system for stock market forecasting. *International Journal of Computer and Information Engineering*, *2*(3), 689-692.

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, *28*(3), 653-664.

Dias, A. (2013). Market capitalization and Value-at-Risk. *Journal of Banking & Finance*, *37*(12), 5248-5260.

Ding, Z., Huang, Y., Yuan, H., & Dong, H. (2020). Introduction to reinforcement learning. In *Deep reinforcement learning* (pp. 47-123). Springer, Singapore.

Du, S. S., Lee, J. D., Mahajan, G., & Wang, R. (2020). Agnostic Q-learning with function approximation in deterministic systems: Tight bounds on approximation error and sample complexity. *arXiv preprint arXiv:2002.07125*.

Eddy, S. R. (2004). What is dynamic programming? *Nature biotechnology*, *22*(7), 909-910.

Engel, Y., Mannor, S., & Meir, R. (2005, August). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on Machine learning* (pp. 201-208).

Feinberg, E. A., & Shwartz, A. (Eds.). (2012). *Handbook of Markov decision processes: methods and applications* (Vol. 40). Springer Science & Business Media.

Friedman, J. H. (1994). An overview of predictive learning and function approximation. *From statistics to neural networks*, 1-61.

Ganesh, P., & Rakheja, P. (2018). VLSTM: Very Long Short-Term Memory Networks for High-Frequency Trading. *arXiv preprint arXiv:1809.01506*.

Hsu, M. W., Lessmann, S., Sung, M. C., Ma, T., & Johnson, J. E. (2016). Bridging the divide in financial market forecasting: machine learners vs. financial economists. Expert Systems with Applications, 61, 215-234.

Hu, J., & Wellman, M. P. (1998, June). Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML* (Vol. 98, pp. 242-250).

Icarte, R. T., Klassen, T., Valenzano, R., & McIlraith, S. (2018, July). Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning* (pp. 2107-2116). PMLR.

Ireland, P. (2010). Limited liability, shareholder rights and the problem of corporate irresponsibility. *Cambridge Journal of Economics*, *34*(5), 837-856.

Kappen, H. J. (2011). Optimal control theory and the linear Bellman equation

Koratamaddi, P., Wadhwani, K., Gupta, M., & Sanjeevi, S. G. (2021). Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation. *Engineering Science and Technology, an International Journal*, *24*(4), 848-859.

Le, D. Y. N., Maag, A., & Senthilananthan, S. (2020, November). Analysing Stock Market Trend Prediction using Machine & Deep Learning Models: A Comprehensive Review. In *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)* (pp. 1-10). IEEE.

Liu, X. Y., Yang, H., Gao, J., & Wang, C. D. (2021, November). FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the Second ACM International Conference on AI in Finance* (pp. 1-9).

Luccioni, A., & Palacios, H. (2019). Using natural language processing to analyze financial climate disclosures. In *Proceedings of the 36th International Conference on Machine Learning, Long Beach, California*.

Malkiel, B. G. (1989). Efficient market hypothesis. In *Finance* (pp. 127-134). Palgrave Macmillan, London.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, *12*(4), 875-889.

Mosavi, A., Faghan, Y., Ghamisi, P., Duan, P., Ardabili, S. F., Salwana, E., & Band, S. S. (2020). Comprehensive review of deep reinforcement learning methods and applications in economics. *Mathematics*, *8*(10), 1640.

Nan, A., Perumal, A., & Zaiane, O. R. (2020). Sentiment and knowledge based algorithmic trading with deep reinforcement learning. arXiv preprint arXiv:2001.09403.

Ni, Z., Paul, S., Zhong, X., & Wei, Q. (2017, November). A reinforcement learning approach for sequential decision-making process of attacks in smart grid. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1-8). IEEE.

Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S. S., ... & Gandomi, A. H. (2020). Data science in economics: comprehensive review of advanced machine learning and deep learning methods. *Mathematics*, *8*(10), 1799

Nuti, G., Mirghaemi, M., Treleaven, P., & Yingsaeree, C. (2011). Algorithmic trading. *Computer*, *44*(11), 61-69.

Otterlo, M. V., & Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement learning* (pp. 3-42). Springer, Berlin, Heidelberg.

Prasad, D. (1994). Is underpricing greater for mixed offerings as compared to pure primary offerings in the OTC Market. *Journal of Financial and Strategic Decisions*, *7*(1), 25-31.

Rincón-Zapatero, J. P., & Rodríguez-Palmero, C. (2003). Existence and uniqueness of solutions to the Bellman equation in the unbounded case. *Econometrica*, *71*(5), 1519-1555.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Vol. 37, p. 14). Cambridge, UK: University of Cambridge, Department of Engineering.

Sarangi, P. K., Singh, S., & Sahoo, A. K. (2022). A Study on Stock Market Forecasting and Machine Learning Models: 1970–2020. In *Soft Computing: Theories and Applications* (pp. 515-522). Springer, Singapore

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Machine Learning Proceedings 1994* (pp. 284-292). Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tola, V., Lillo, F., Gallegati, M., & Mantegna, R. N. (2008). Cluster analysis for portfolio optimization. *Journal of Economic Dynamics and Control*, *32*(1), 235-258.

Vadori, N., Ganesh, S., Reddy, P., & Veloso, M. (2020, October). Risk-sensitive reinforcement learning: A martingale approach to reward uncertainty. In *Proceedings of the First ACM International Conference on AI in Finance* (pp. 1-9).

Wang, S., & Jing, Y. (2017). Deep Reinforcement Learning with Surrogate Agent-Environment Interface. *arXiv preprint arXiv:1709.03942*.

Wen, L., Zhou, K., Li, J., & Wang, S. (2020). Modified deep learning and reinforcement learning for an incentive-based demand response model. *Energy*, *205*, 118019.

Woergoetter, F., & Porr, B. (2008). Reinforcement learning. *Scholarpedia*, *3*(3), 1448.

Woolridge, J. R., & Dickinson, A. (1994). Short selling and common stock prices. *Financial Analysts Journal*, *50*(1), 20-28.

Wulfmeier, M., Rao, D., Hafner, R., Lampe, T., Abdolmaleki, A., Hertweck, T., ... & Riedmiller, M. (2021, July). Data-efficient hindsight off-policy option learning. In *International Conference on Machine Learning* (pp. 11340-11350). PMLR.

Yang, H., Liu, X. Y., Zhong, S., & Walid, A. (2020, October). Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance* (pp. 1-8).

Yin, M., & Wang, Y. X. (2020, June). Asymptotically efficient off-policy evaluation for tabular reinforcement learning. In *International Conference on Artificial Intelligence and Statistics* (pp. 3948-3958). PMLR.

Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, *2*(2), 25-40.

Zou, S., Xu, T., & Liang, Y. (2019). Finite-sample analysis for sarsa with linear function approximation. *Advances in neural information processing systems*, *32*.