POLITECNICO DI TORINO

Department of Automation and Computer Science - College of Computer Engineering, Cinema and Mechatronics

Master's degree programme in

DATA SCIENCE AND ENGINEERING

Master’s Thesis

Financial data analysis by means of Reinforcement Learning techniques

Supervisor

prof. Luca Cagliero

Co-Supervisor

Jacopo Fior

Candidate

Francesco Mangia

Marzo 2022

1. Introduction

In academia and the financial industry, quantitative trading has been a popular topic since the late 1960s. In general, it refers to the use of statistical models and data-driven methodologies in financial market analysis, following two main approaches. The classical approach is related to the development of economic theories in order to interpret the financial market behaviour. A few representative examples include the well-known Capital Asset Pricing Model and Markowitz Portfolio Theory.

Computer scientists apply data-driven techniques to analyze financial data. In particular, machine learning and deep learning have been critical components generating a high interest in the finance industry in the last period. The advancements of these techniques, technologies, and skills have allowed the financial industry to grow at a frenetic pace over the years due to its exceptional performance as well as the appealing property of learning meaningful representations from scratch. Most of these adaptive systems rely on supervised learning, which involves the training of a forecasting model on historical data to project market products trend direction. However, despite their popularity, these methods usually have a number of flaws that result in suboptimal results linked to the fact that financial assets trading is not only a process aiming to estimate future prices: it also requires many other elements that should be taken into account, such as the risk involved as well as exogenous constrictions like transaction fees and market liquidity. On the contrary, the supervised model aims to minimise prediction error (profit maximization) regardless of risk, which is not in the investor's best interests. Furthermore, as financial market data is extremely noisy, the performance of supervised machine learning algorithms, such as Neural Networks, can be suboptimal.

Due to its outstanding ability to solve complex sequential decision-making problems, reinforcement learning (RL) has attracted significant interest in many automation domains over the last decade, including robotics and video games. RL is able to fully use large amounts of financial data with fewer model assumptions and to improve decisions in complex action and space states training an agent to solve dynamic optimization problems. In the Reinforcement Learning context, the algorithm is not given any explicit supervision: it relies on the rewarding function to guide the agent and evaluate how it is behaving in the given environment. The agent in this context is a trader, the actions are the amounts of shares to be traded, and the environment is the stock's price movement.

Multistock Trading and Portfolio Allocation are problems that can be easily framed into a RL application. In the thesis we explore the performance of different RL algorithms in a variety of contexts with different parameterizations, training the agents with historical prices of the Dow Jones' stocks. The Reward Function, in particular, plays a key role in Reinforcement Learning applications. It is an incentive technique that utilizes a reward and punishment system to make the agent understand how it should behave in order to maximize its reward. As a consequence, the natural approach to train a RL agent to solve a quantitative trading task is to reward its action with a value proportional to the gained profit (Multistock Trading problem), or to the new Portfolio Value (Portfolio Allocation problem).

State-of-the-art RL techniques have already demonstrated to be profitable, being efficient in different market conditions and beating the benchmarks. Despite the results, the classic

rewarding methodology brings to model instability, overfitting and ineffectiveness in taking into account medium and long-term trends, and difficulties in diversifying the portfolio exposure to the market, while not being interpretable enough to be inspected before any deployment. Solely providing the agent with historical price data information is often not enough to enable it to learn comprehensive and reliable strategies. Especially when dealing with multi stock trading problems the complexity of the process increases, due to the fluctuating nature of the market and the demand to manage a structured portfolio, as it is necessary for the agent to take into account several factors such as distinguishing how much to allocate on each asset, which positions to open in what direction, and what is the best condition to close them in order to accumulate the maximum reward in an unknown environment. In these circumstances, the choice of the reward function turns out to be a fundamental aspect, and must be tailored to the trading context allowing the model to learn how to act efficiently.

This thesis extends the state-of-the-art RL algorithms by also analyzing a series of financial data containing, for each trading day, the information about prices' moving averages, historical volumes, trend momentum indicators, volatility-related information and a correlation matrix between the stocks considered. In particular, the thesis research aims at taking advantage of the historical technical analysis data. It also incorporates in the reward function the metrics associated to the historical returns and the asset allocation thus allowing the trader to configure his risk-aversion profile.

The experiments reported in this thesis work were run on the prices and exchange volume data of the Dow Jones stocks in the period from January 1st, 2001 to October 31th, 2021. The analysis of the results shows that our approach can accurately interfere with the behaviour of the agent with a high interpretability level and can be applicable in a real-world working application, with an integration in the workflow of the risk-management office, empowering the operators with a tool tailored on their risk-aversion profile.

In the future, this approach could be extended to work on other types of financial products or markets, different RL algorithms, data and preprocessing pipelines.

2. Financial Engineering

The stock market definition refers to the collection of exchanges and other venues, where shares of publicly held companies can be bought, sold, and issued. A corporation's shares are units of equity ownership. Shares exist as a financial asset for some companies, enabling for an equal distribution of any residual profits, if any, in the form of dividends. Shareholders of a stock that does not pay a dividend will not receive profit distribution but they expect to benefit from rising stock prices as the company's income rises.

2.1 Financial Terms and Concepts

This chapter aims to give a complete overview about the financial terms and concepts we made use of in the thesis starting from the basic concepts about positions handling up to the main portfolio allocation strategies.

2.1.1 Long & Short

From the point of view of investments in the stock market, there are essentially two strategies: the first one is based on the adoption of "long positions", by specifically betting on stock rising; the second one is based on the adoption of "short positions", through which a trader bets on the fall of a stock.

The long position assumes the purchase of an asset to later resell it at a higher price while a “short” sale is instead the sale of a financial product that an investor does not own or a sale which is consummated by the delivery of a stock borrowed by the investor. The investor later closes out the position by returning the borrowed security to the stock lender, typically by purchasing securities on the open market

Investors who sell stock short typically believe the price of the stock will fall and hope to buy the stock at the lower price in order to make a profit.

If the asset pays cash dividends while the short position is held, the dividends are paid to the short sale buyer. The lender must also be compensated by the short seller, who must match the cash dividends with his own funds.

In other words, the short seller is equally responsible for paying the lender’s cash dividends on the borrowed asset.

2.1.2 Transaction Fees

Transaction fees are expenses incurred when buying or selling a good or service representing the labour required to bring a good or service to the market, giving rise to an entire industry dedicated to facilitating exchanges. In a financial sense, transaction costs include brokers' commissions and spreads, which are the differences in terms of price paid from the dealer and from the buyer.

2.2 Financial Time Series

Prices change throughout time due to the dynamic behaviour of the economy and the non-static supply and demand balance enable market dynamics to be treated as time series and to be analysed and modelled using quantitative methods and instruments.

The theory and practice of asset valuation across time are the focus of financial time series analysis. Although it is a very empirical subject, theory serves as a framework for making inferences, just as it does in other scientific fields.

As a result of the uncertainty existing in both financial theories and its empirical time series, financial time series analysis is set apart from other types of time series analysis.

There are numerous definitions of asset volatility, for example, and the volatility of a stock return series is not clearly visible. This is why statistical theory and methods play an essential part in financial time series analysis [3, 4].

2.2.1 Prices

Let pt belong to $\mathbb{R}$ and be the price of an asset at discrete time t. The sequence $p1, p2, ..., pT$ is a univariate time-series. We use pi,t to represent the price of an asset i at time t and $p_{\text{asset } i,t}$ to represent the prices of different assets.

Given the price time series of an asset i, the column vector $\vec{p}_{i,1:T}$ is:

$$\vec{p}_{i,\,1:T} = \left[p_{i,\,1}\,,\, p_{i,\,2}\,,\, \ldots\,,\, p_{i,\,T}\right]^T \in R_+^T$$

In order to make portfolio analysis easier, we define the price vector p_t as follows:

$$p_t = \left[p_{1,\,t}\,,\, p_{2,\,t}\,,\, \ldots\,,\, p_{M,\,t}\right] \in R_+^M$$

By stacking T column-wise – samples price time-series of the M assets in the portfolio, we build the asset price matrix $P_{1:T}$ and extend the single-asset time-series notation to the multivariate scenario:

$$\vec{P}_{1:T} = \left[\vec{p}_{1,1:T'}, \quad \vec{p}_{2,1:T'}, \cdots, \vec{p}_{M,1:T}\right] = \begin{bmatrix} p_{1,1} & p_{2,1} & \cdots & p_{M,1} \\ p_{1,2} & p_{2,2} & \cdots & p_{M,2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{1,T} & p_{2,T} & \cdots & p_{M,T} \end{bmatrix} \in \mathbb{R}_{+}^{T \times M}$$

2.2.2 Volume

The trading volume of a financial asset is a measure of how much it has been traded in a certain period of time. It is measured by the number of shares traded in stocks, and in the number of contracts traded in futures and options.

Looking at volume trends over time might allow traders and quantitative investors to understand the strength or confidence behind certain stocks and market rises and losses. Volume, in fact, plays a vital part in quantitative analysis and is one of the most important technical indicators and its movement prediction is the key in a variety of financial applications [5, 6, 7].

2.2.3 Candlestick Chart

A candlestick chart is a type of financial chart that is used to represent price fluctuations over time. Using financial terms, It is also called the “Japanese candlestick chart”, as it was devised by Japanese rice trader Munehisa Hooma [8, 9].

A daily candlestick chart describes the market's movement for that day. The "*real body*" of the candlestick is the central section that represents the price range between the open and close of that day's trade. When the close price is lower than the open, the true body is filled in or black. On the other hand if the real body is empty it means that the trading day closed positively.

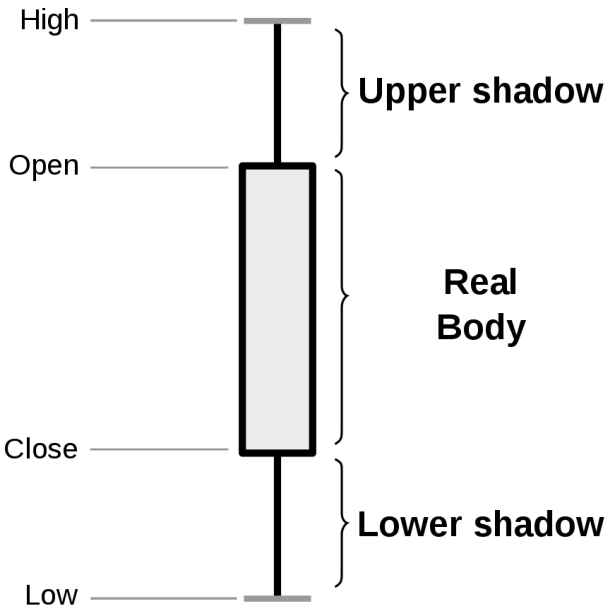

Image 2.2 Candlestick format detailed description. The image shows how and which information are described by a candle in a candlestick chart

Source: https://commons.wikimedia.org/wiki/File:Candlestick_chart_scheme_01-en.svg

The shadows are the upper and lower lines and show the highest and lowest price ranges within a given time period.

The candlestick chart is a visual aid for making stock market decisions giving traders the chance to better grasp the link between the high and low, as well as the open and close values.

2.3 Technical Indicators

Technical indicators are the most common analysis tools used to determine the market trend of an asset using historical price and volume information.

A large number of technical indicators and associated trading rules have been developed over the years with mixed success. All of them showed different performances under different market conditions and different human operators, which is why traders often used multiple indicators to confirm the same signal.

The same indicator can be analysed on different time frames to assess local and global trends for that particular asset [10].

2.3.1 SMA

A Simple Moving Average (SMA) computes the average of a specified range of prices, typically closing prices, divided by the number of periods in that range.

$$SMA = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

Where:

$$A_n = the\ price\ of\ an\ asset\ at\ period\ n$$

$$n = the\ number\ of\ total\ periods$$

2.3.2 MACD

The Moving Average Convergence Divergence indicator, namely MACD, is a measure that shows the relationship between two price moving averages. In particular the MACD is estimated as the difference between the Exponential Moving Average (EMA) of 26-periods and the EMA 12-periods.

$$MACD = 12 - Period\ EMA - 26 - Period\ EMA$$

The crossover between the two EMA, and consequently the MACD cross with the 0 value, is usually interpreted as a buy/sell signal.

2.3.3 Bollinger Bands

Bollinger Bands are a type of price envelope developed by John Bollinger. Bollinger Bands define upper and lower price range levels at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.

$$BOLU = MA\,(TP, n) + m * \sigma\,[TP, n]$$

$$BOLD = MA\,(TP, n) - m * \sigma\,[TP, n]$$

Where:

$$BOLU = Upper\ Bollinger\ Band$$

$$BOLD = Lower\ Bollinger\ Band$$

$$MA = Moving\ average$$

$$TP\ (typical\ price) = (High + Low + Close) \div 3$$

$$n = Number\ of\ days\ in\ smoothing\ period\ (typically\ 20)$$

$$m = Number\ of\ standard\ deviations\ (typically\ 2)$$

$$\sigma\ [TP,\ n] = Standard\ Deviation\ over\ last\ n\ periods\ of\ TP$$

2.3.4 Relative Strength Index

The relative strength index (RSI) is a technical indicator used that measures whether a stock or an asset is overbought or oversold based on the most recent market movements.

The RSI has traditionally been interpreted such that values greater than 70 indicate that a security is becoming overbought or overvalued and may be primed for a trend reversal or corrective price pullback. An RSI value less than 30 instead, indicates that the market could be potentially undervalued.

RSI is calculated as:

$$RSI_{step\ two} = 100 - \left[\frac{100}{1+\frac{(Previous\ Average\ Gain\ x\ 13) + Current\ Gain}{((Previous\ Average\ Loss\ x\ 13) + Current\ Loss)}}\right]$$

2.3.5 Commodity Channel Index

Donald Lambert, a technical analyst, created the CCI, or Commodity Channel Index, which was first published in Commodities magazine (now Futures) in 1980. The CCI, despite its name, can be used in any market and is not limited to commodities [12].

The CCI was originally designed to detect long-term trend changes, but traders have adapted its use on all markets and timeframes. Active traders benefit from multiple buy and sell signals when trading across multiple timeframes.

Traders frequently use the CCI to establish the dominant trend on the longer-term chart and to isolate pullbacks and generate trade signals on the shorter-term chart.

$$\frac{(Typical\ Price - Simple\ Moving\ Average}{(0.0.15\ x\ Mean\ Deviation}$$

2.3.6 Directional Movement Index

The directional movement index (DMI) is a technical indicator developed that is used to determine the strength and direction of a price movement and is intended to help the trader filter possible false signals.This is accomplished by comparing previous highs and lows and drawing a positive directional movement line (DI^{+}) and a negative one (DI^{-}).

Crossovers between the directional movement lines are often used as buy or sell signals by traders [13]. The larger the spread between the two primary lines, the stronger the price trend. If +DI is way above -DI the price trend is strongly up and vice versa.
DMI is calculated as:

$$+\ DI\ = \left(\frac{Smoothed + DM}{ATR}\right)\times 100$$

$$\text{-}DI\ = \left(\frac{Smoothed\ \text{-}DM}{ATR}\right)\times 100$$

$$DX\ = \left(\frac{|+DI-\ \text{-}DI|}{|+DI+\ \text{-}DI|}\right)\times 100$$

where**:**

$$+\ DM(Directional\ Movement)\ =\ Current\ High\ -\ PH$$

$$PH\ =\ Previous\ high$$

$$\text{-}DM\ =\ Previous\ Low\ -\ Current\ Low$$

$$Smoothed \pm DM\ =\ \sum_{t=1}^{14} DM\ -\left(\frac{\sum_{t=1}^{14} DM}{14}\right)+\ CDM$$

$$CDM\ =\ Current\ DM$$

$$ATR\ =\ Average\ True\ Range$$

2.3.7 GICS: The Global Industry Classification Standard

The Global Industry Classification System (GICS) is a four-tiered, hierarchical industry classification system based on quantitative and qualitative analysis developed in 1999 by Morgan Stanley Capital International and S&P Dow Jones Indices.

GICS systematically identifies every company by sector, industry group, industry, and sub-industry and is used by investors and analysts to identify, compare, and contrast a firm's competitors [14].

2.4 Asset Returns

The majority of financial research focuses on asset returns rather than asset prices.

Returns are used for two main reasons, according to Campbell, Lo, and MacKinlay (1997). Firstly, the return on an asset provides a comprehensive and scale-free assessment of the investment potential. Secondly, as return series have more appealing statistical features than price series, they are easier to handle.

An asset return, on the other hand, can be defined in a variety of ways [15].

2.4.1 One-Period Single Return

The *gross rate* of return on an investment is the overall rate of return before any fees, commissions, or expenses are deducted. It is expressed as a percentage over a given time period, such as a month, quarter, or year.

$$1 + R_t = \frac{P_t}{P_{t-1}} \; or \; P_t = P_{t-1}(1 + R_t)$$

The *net rate* of return, on the other hand, takes fees and costs out of the equation to give a more accurate picture of return.

$$R_t = \frac{P_t}{P_{t-1}} - 1 = \frac{P_t - P_{t-1}}{P_{t-1}}$$

2.4.2 Multi-Period Single Return

A *k-period simple gross return* is obtained by holding the asset for k periods between dates t - k and t.

$$1 + R_t[k] = \frac{P_t}{P_{t-k}} = \frac{P_t}{P_{t-1}} \, x \, \frac{P_{t-1}}{P_{t-2}} \, x \ldots \, x \, \frac{P_{t-k+1}}{P_{t-k}}$$

$$= (1 + R_t)(1 + R_{t-1}) \ldots (1 + R_{t-k+1})$$

$$= \prod_{j=0}^{k-1} (1 + R_{t-j})$$

2.4.3 Portfolio Return

A *portfolio's simple net return* is a weighted average of the simple net returns of the assets in the portfolio, where the weight of each asset equals the proportion of the portfolio's value allocated on that product.

Let p represent a portfolio with a weighting of wi on asset i. Then, at time t, the simple return of p is

$$\sum_{i=1}^{N} w_i R_{it}$$

2.4.4 Log Return

The *log return* is obtained computing the log of the gross return. Although Simple Return is a more straightforward method of calculating returns, it is asymmetric.

Log returns are a symmetric approach of calculating an asset's future value.

$$\rho_t \triangleq ln\left(\frac{p_t}{p_{t-1}}\right) ln\left(R_t\right) \epsilon R$$

2.5 Evaluation Metrics

To evaluate the performance of an algorithm in the financial field, we often rely on a set of evaluation metrics that allow us to quantify how the model has performed, both in terms of profitability and risk.

This chapter aims to analyze evaluation metrics used in this research, providing a comprehensive definition and the empirical interpretations.

2.5.1 Sharpe Ratio

The Sharpe ratio is a risk-adjusted evaluation metric developed by William F. Sharpe in 1977 [16] and is used to understand the return of an investment strategy compared to the underlying risk. It is calculated as a ratio between the average return earned in excess of the risk-free rate per unit of volatility. It can be formulated as:

$$Sharpe\ Ratio\ = \frac{R_p - R_f}{\sigma_p}$$

Where:

$$R_p = return\ of\ portfolio$$

$$R_f = risk\text{-}free\ rate$$

$$\sigma_p = standard\ deviation\ of\ the\ portfolio's\ excess\ return$$

The R_f could be a U.S. Treasury rate , such as the one-year or two-year Treasury yield [17]. In general, the risk-free rate is commonly thought to be the yield paid by a 3-month government Treasury bill, which is statistically the safest operation for an investor.

2.5.2 Calmar Ratio

The Calmar ratio [19,18] is a metric created by Terry W. Young that is used to give a quantitative idea about the performance of a trading strategy or fund. As shown in the formula below it is calculated as the average compounded annual rate of return versus the maximum drawdown encountered. It takes the name of its creator company press release: **CAL**ifornia **M**anaged **A**ccounts **R**eports.

$$CalmarRatio = \frac{R_p - R_f}{Maximum\ Drawdown}$$

Where:

$$R_p = portfolio\ return$$

$$R_f = risk\text{-}free\ rate$$

$$R_p - R_f = annual\ rate\ of\ return$$

It aims to demonstrate the amount of risk required to obtain a return.

2.5.3 Omega Ratio

The *Omega ratio* or reward-to-variability ratio, conceived by Con Keating and William F. Shadwick, is used as a risk-return performance measure of an investment strategy or fund. It is formulated as the probability weighted ratio of gains on losses for a given threshold returns target [20].

This indicator is calculated by a partition of the cumulative return distribution to create an area of losses and an area for gains relative to the given threshold:

$$\Omega(r) \triangleq \frac{\int_r^{\infty} (1 - F(x))\, dx}{\int_{-\infty}^{r} F(x)\, dx}$$

Where:

$F = Cumulative\ probability\ distribution\ function\ of\ the\ returns$;

$r = The\ target\ return\ threshold.\ It\ is\ used\ to\ define\ what\ is\ considered\ a\ gain\ vs\ a\ loss$;

An investor should prefer a strategy with larger value since it suggests that the asset provides more gains relative to losses for some threshold θ [21].

2.5.4 Sortino Ratio

The Sortino ratio measures the risk-adjusted return of an investment strategy[22].

It is a variation of the Sharpe ratio but it penalises only those returns falling below a user-specified threshold or required rate of return, while the Sharpe ratio penalises both downside and upside volatility equally [23].

$$S = \frac{R - T}{DR}$$

Where R is the strategy average realised return, T is the target rate of return for the investment strategy under consideration (called the minimum acceptable return *MAR*), and DR is the target semi-deviation (the square root of target semivariance), also referred to as downside deviation. The yearly standard deviation of returns below the target is a simple way to think about downside risk.

2.5.5 Tail ratio

The tail ratio is an indicator used to evaluate the ratio between the 95% right and 5% left tail of a distribution of returns. For example, a ratio of 0.5 means that losses are two times greater than profits.

2.5.6 Stability

Stability is an indicator used to determine the R-squared of a linear fit to the cumulative log returns. It is calculated computing an ordinary least squares linear fit, and returns R-squared.

2.5.7 Skewness

In statistics the skewness of a distribution indicates the asymmetry of the values around the mean. As a normal distribution is symmetric around the mean, skewness can be used to evaluate how non-normally the distribution of the returns of a strategy are.

For instance, if a portfolio's returns distribution is right-skewed, it means that there are a lot of small negative returns compared to few big ones. On the other hand, if they are negatively skewed, it can be interpreted as small positive returns and frequent large negative returns.

For a trader having a negatively skewed distribution of returns means that the portfolio is at risk of rare but massive losses.

$$Skew = \frac{\sum_{t=1}^{n} \left(x_i - \underline{x}\right)^3 / n}{\left(\sum_{t=1}^{n} \left(x_i - \underline{x}\right)^2 / n\right)^{3/2}}$$

2.5.8 Kurtosis

The Kurtosis is a measure of the proportion of portfolio returns that occurs in the tails of a distribution. A normal distribution is defined as having a kurtosis value equal to 3, which results from the tails of a normal distribution containing some of its mass.

A distribution with a kurtosis value greater than 3 has a larger tail than a normal distribution, while one with a kurtosis less than 3 has fewer returns in its tails than a normal distribution.

This is important to be analysed in a strategy tuning since the higher the kurtosis value, the higher the risk of a rare but massive downside of the portfolio value.

In other words a higher kurtosis value corresponds to more frequent outliers, as opposed to frequent modestly sized deviations. Kurtosis can be calculated as:

$$Kurtosis = \frac{\sum_{t=1}^{n} \left(x_i - \underline{x}\right)^4 / n}{\left(\sum_{t=1}^{n} \left(x_i - \underline{x}\right)^2 / n\right)^2}$$

2.6 Portfolio Management Strategies

Portfolio management is the science of selecting and supervising a group of investments based in order to target long-term financial objectives given a risk tolerance. In order to correctly manage a portfolio of financial assets a portfolio manager or a trader necessitates to have the ability to evaluate opportunities and risks across the entire investment spectrum. In order to build the optimal portfolio there are multiple trade-offs to consider starting from the financial product choice such as debt against equity, domestic versus foreign, and growth versus safety [24].

2.6.1 Risk Management

In finance, risk is defined in ISO 31000 as *the effect of uncertainty on objectives*. A fundamental aspect for a fund manager or a trader is the portfolio risk-management, namely the process of analysis, identification and mitigation of uncertainty in investment decisions. Risk management is fundamental to quantify the potential losses and to take the appropriate action to achieve the fund's investment objectives and do not exceed the risk tolerance.

Every investment operation exposes the fund or the trader to some degree of risk, which is quantifiable in absolute and in relative terms. It is generally considered equal to zero in the case of the U.S. T-bill or very high in emerging-market equities or real estate. A proper understanding of risk in its various forms can aid investors in better comprehending the opportunities, costs, and trade-offs associated with different investment strategies.

2.6.2 Benchmark

A benchmark is a measure of a security, mutual fund or investment manager's performance.

The most common stock and bond indexes used for this scope are broad market and market-segment indexes.

2.6.3 Passive Management

Passive management or index fund management is a set-it-and-forget-it long-term investing strategy. This strategy is usually designed to closely match the returns of a specific market index or benchmark.

2.6.4 Active Management

Active management involves attempting to beat the performance of an index (S&P, DOW, etc.) by actively buying and selling individual stocks and other financial products modifying the portfolio allocation based on a given strategy or the trader analysis. [26]

- Follow The winner: follow-the-Winner is a strategy that involves raising the relative weights of more successful experts/stocks.

- Follow The Loser: rather than following the winners, the Follow-the-Loser strategy involves transferring wealth from winners to losers. The basic premise of this technique is mean reversion, which states that good (poor)-performing assets will perform poorly (well) in the future.

- Pattern-Matching-based: these approaches are based on the assumption that market sequences with similar preceding market appearances tend to re-appear. Thus, the common behaviour of these approaches is to firstly identify similar market sequences

that are deemed similar to the coming sequence, and then obtain a portfolio that maximises the expected return based on these similar sequences.

3. Reinforcement Learning: Fundamentals

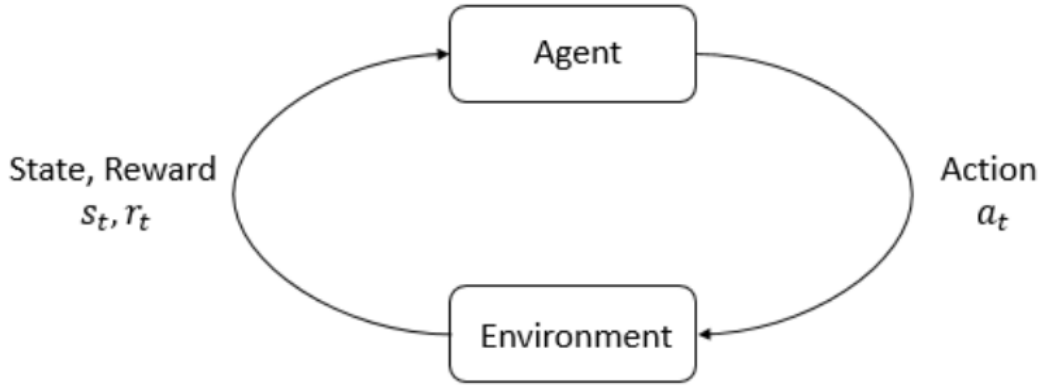

Image 3.1 Reinforcement Learning overview. The image points out how an agent acts in an environment in order to gain a reward based on his action choice.

Reinforcement Learning (RL) is a field of machine learning that investigates how an agent should operate in a given environment in order to maximize a cumulative reward based on the task. Thanks to its adaptability, RL is broadly applied to many different disciplines such as game theory, statistics and optimization research in a variety of fields. [27]

One of the fundamental contrasts between RL and supervised learning is that in RL, the agent's outcomes may modify the environment. This way RL can solve Markov Decision Processes even if the transition probabilities are not explicitly specified.

The objective of a Reinforcement Learning agent is to maximize the total Expected Reward for all the incoming actions in the next state.

$$E\left[R\left(s_0, a_0\right) + \gamma R\left(s_1, a_1\right) + \ldots + \gamma^n R\left(s_n, a_n\right)\right] = E\left[\sum_n \gamma^n R\left(s_n, a_n\right)\right]$$

Not every action immediately results in a reward/feedback; in certain cases, extended sequences of activities are required to receive a reward, yet future actions have not been completed. This means that the algorithm will have a tougher time learning if the rewards are more sparse.

3.1 Markov Decision Process (MDP)

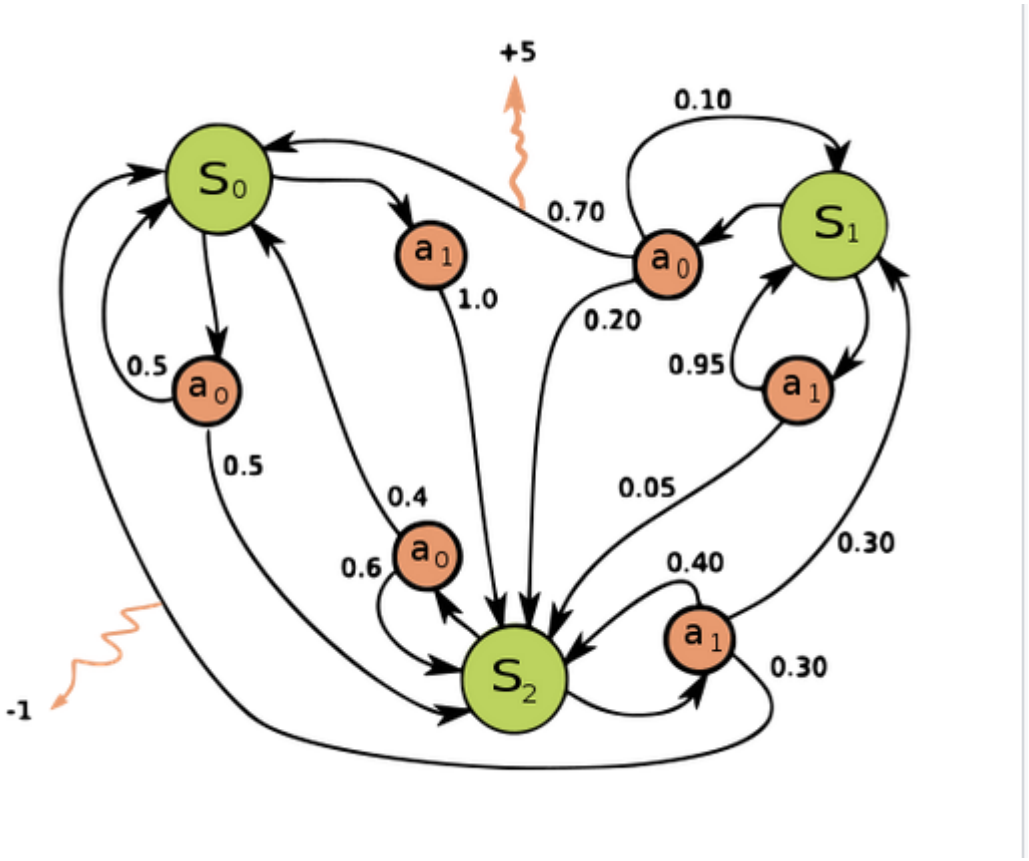

Image 3.2 Example of a simple MDP network.
Source: https://en.wikipedia.org/wiki/Markov_decision_process

A Markov decision process (MDP) is a discrete-time stochastic control process in mathematics that provides a strong mathematical framework to model decision-making in environments where outcomes are distributed partly random and partly produced by a decision maker. The essence of this mathematical model is that the state of the agent environment affects the reward obtained and the probabilities of future state transitions.

It is made of four tuples, S, A, P_a, P_r:

- S: the state space
- A: the action space. It is the set of potential action starting from the state S
- $P_a(s, s')$: the probability that taking the action a in the current state s, will result, at time $t + 1$, in state s'
- R(s, s'): is the expected immediate reward for taking the action a in the state s

The process is completely based on the Markov property, namely the memoryless property of a stochastic process [25]. Transitions from a state to the next one only depend on the current state and action, and no prior history [26].

3.2 Policy

A policy is a set of rules that an agent uses to determine which actions to choose in a given state [17, 18].

It can be deterministic, which is generally indicated by μ:

$$a_t = \mu(s_t)$$

or stochastic, which is usually denoted by π:

$$a_t \sim \pi(\cdot|s_t)$$

3.3 Value Function

Almost all Reinforcement Learning algorithms use estimating value functions, which are state functions that estimate how good it is for the agent, in terms of reward, to perform a given action in that particular state.

We can distinguish four different types of Value Function that will be discussed in the next chapters.

3.3.1 On-Policy Value Function

Vπ(s), which represents the expected return if one starts in state s and always follows the policy π.

$$V^{\pi}(S) = E_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s\right]$$

3.3.2 Optimal Value Function

V*(s) is the expected return if one starts in state s and always operates in accordance with the best environmental policy.

$$V^{*}(s) = max_{\pi} E_{\tau\sim\pi}\left[R(\tau)|s_0 = s\right]$$

3.3.3 On-Policy Action-Value Function

Qπ(s,a), which gives the expected return if one starts in state s, take an arbitrary action a, and then act according to policy for the rest of the time.

$$Q^{\pi}(s,\ a)\ = E_{\tau\sim\pi}\left[R\ (\tau)|s_0\ ,\ a_0\ =\ a\right]$$

3.3.4 Optimal Action-Value Function

Q*(s,a), which calculates the expected return if one starts in state s, takes an arbitrary action a, and then operates in the environment according to the best policy.

$$Q^{*}(s,\ a)\ = max_{\pi}\ E_{\tau\sim\pi}\left[R\ (\tau)|s_0\ =\ s\ ,\ a_0\ =\ a\right]$$

There are two important relationships between the value function and the action-value function that are frequently discussed. In particular:

$$V^{\pi}(s)\ =\ E_{a\sim\pi}\left[Q^{\pi}(s,\ a)\right]$$

and

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

The optimal action-value function, Q*(s,a), and the action chosen under the optimal policy have a strong relationship.

Q*(s,a), infact, is the expected return for starting in state s, executing an arbitrary action a, and then operating according to the optimal policy for the rest of the time. In s, the optimal policy will choose the action that maximises the expected return starting from s.

As a result, if we have Q, we can get the best action, a*(s), directly as:

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

3.4 Returns

A RL agent's goal is to select a policy that maximises the future expected cumulative rewards.

The cumulative return, Gt, is defined as:

$$G_t = r_t + r_{t+1} + r_{t+2} + \ \ldots\ \ + \ r_{N-1}$$

where t is the time index and N marks the end of an episode. For continuous action-spaces, the discounted return can be formulated as:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where γ is the discount factor and $\gamma \in [0, 1)$. The meaning and interpretations of the discounted returns will be explored in chapter 5.3.

3.5 Bellman Equation

The Bellman Equation is the core of many Reinforcement Algorithms. In summary, it decompresses the value function into two parts: the current reward and the discounted future values.

Rather than summing over successive time steps, Bellman Equation simplifies the evaluation of the value function, allowing us to identify the best solution to a complex problem by breaking it down into smaller recursive subproblems and finding their optimal solutions.

$$q_*(s, a) = E\,[R_{t+1} + \gamma \max_{a'} q_*(s', a')$$

The Bellman Equation is formulated such as at time t, the expected return from the initial state s, performing the action a and then following the optimal policy will be equal to the expected reward R_{t+1} we could achieve choosing an action in state s, plus the maximum of the expected discounted return that is feasible of any (s′, a′) where (s′, a′) is a potential succeeding state-action set.

If the agent follows an optimal policy, it is expected that the final state s′ is the one in which the best possible next action a′ at time t_{+1} can be taken.

The Bellman Equation evaluated for on-policy value-functions can be expressed as following:

$$V^{\pi}(s) = E_{a \sim \pi_{s' \sim P}}\left[r(s, a) + \gamma V^{\pi}(s')\right]$$

$$Q^{\pi}(s, a) = E_{s' \sim P}\left[r(s, a) + \gamma E_{a' \sim \pi}\left[Q^{\pi}(s', a')\right]\right]$$

(where $s' \sim P$ is a shorthand for $s' \sim P(.|s,a)$, indicating that the next state s' is sampled from the environment's transition rules; $a \sim \pi$ is a shorthand for $a \sim \pi(.|\, s)$; and $a' \sim \pi$ is a shorthand for $a' \sim \pi(.|\, s')$.)

Reformulating the Bellman Equations for the Optimal Policy Functions, it can be formulated as:

$$V^{*}(s) = max_{a} E_{s' \sim P}\left[r(s,a) + \gamma V^{*}(s')\right]$$

$$Q^{*}(s,a) = E_{s' \sim P}\left[r(s,a) + \gamma max_{a'} Q^{*}(s',a')\right]$$

The absence or presence of the max operator over actions is the essential distinction between the Bellman equations for on-policy value functions and the optimal value functions.

Its inclusion highlights the notion that whenever the agent is involved in a choice of an action, it must choose the action that produces the maximum value in order to operate optimally.

3.6 Model - Free vs Model - Based

The classification of RL algorithms is based on whether the agent has access to a function that forecasts rewards and state transitions. In particular, model-free RL agents are trained in environments with a massive number of possible states.

3.6.1 Model Free Method

Model-free methods do not require an environment model therefore can be applied to any reinforcement learning problem without distinction.

Most model-free approaches either attempt to learn a value function and infer an optimal policy from it (Value Function-based methods) or directly search in the space of policy parameters to find an Optimal Policy (Policy Search methods). These algorithms make no attempt to learn the underlying dynamics that govern how an agent interacts with its environment.

Model-free approaches are also classified as:

- On-policy methods: they choose the actions based on the current policy and use them to update the current policy.

- Off-policy methods: they articulate their actions based on a different exploratory policy than the policy being updated.

3.6.2 Value Function Based Methods

The concept of Value-Based learning is central in reinforcement learning literature. It aims to calculate how good it is to reach certain states or perform certain actions in a given environment.

Even if value-learning alone may not be sufficient to solve complex problems, it is an important building block for many Reinforcement Learning methods. Some important Value-Based methods are:

- *Monte Carlo Methods*: Monte Carlo methods are based on the concept of generalised policy iteration (GPI), an iterative scheme consisting of two steps. The first step, known as the Policy Evaluation Step, attempts to build an approximation of the value function based on the actual policy while, in the Policy Improvement Step, the policy is improved with respect to the current value function.

- *Temporal Difference Methods:* Temporal Difference (TD) Learning is an unsupervised technique that aims to predict the expected value of a variable in a sequence of states without major losses in terms of accuracy

- *Function Approximators*: the idea is to find an approximated function that, given the set of features, estimates the optimal action based on the previous experience gained during the training. They can be a linear combination of the features, neural networks, decision tree, etc.

3.6.3 Policy Search Methods

Policy search methods are another type of RL algorithm that employ parameterized policies π_{θ_π}, where θ_π is the parameter vector. These methods aim to find an optimal policy in the parameter space Θ, $\theta\pi \in \Theta$.

The policy is assessed by carrying out rollouts from the current policy and calculating the reward. The Gradient Descent is then used to update the policy's parameters in the direction of increasing expected return.

The policy parameters' updating rule can be formulated in terms of the expected return, J, as follows:

$$\theta_{t+1}^{\pi} = \theta_{t}^{\pi} + \alpha \bigtriangledown_{\theta^{\pi}} J \, , \, J = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^{k} r_{k} \right)$$

Policy Search methods generally provide higher convergence and are able to learn stochastic policies, which value-based techniques can not. The main disadvantage, instead, is their policy evaluation stage, which has a high variance and can therefore be slow to learn appropriate policies.

3.6.4 Actor Critic Methods

Actor-critic methods are temporal-difference methods that explicitly store the policy. The policy is known as the "actor" since it provides the action in a given state. The critic instead, approximates the Value Function suggesting the direction in which to update the policy.

The actor critic method is generally on-policy, despite some off-policy implementations have been introduced in the literature.

Actor critic approaches outperform prior time-difference methods in terms of convergence and simplifies the computation of the action, which was a critical point for continuous action-spaces tasks. Furthermore, Actor-Critic algorithms can be used to learn stochastic policies as well.

3.6.5 Model-based Methods

All the methods that have been discussed so far have no knowledge about the environment. Model-based strategies instead, attempt to enhance sample efficiency by utilising some environmental knowledge.

There are two techniques to implement model-based learning. The first is to create a model of the environment from basic principles and use it to learn the policy or value function. However, this approach often leads to inaccuracies when applied to new scenarios.

On the other hand, the agent can empirically deduce the structure of the environment.

3.7 Temporal Difference Learning

Usually in Reinforcement Learning algorithms, the rewards received from the environment are not immediately observable but are sent back to the terminal state. Temporal Difference (TD) Learning is an unsupervised technique that aims to predict the expected value of a variable in a sequence of states without major losses in terms of accuracy. In fact, TD, instead of trying to predict the final reward, tries to calculate the combination of rewards in all the next states.

Mathematically, the key concept on which TD is based is the discounted returns. It can formulated as:

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

3.8 Discounted Returns

The goal of a Reinforcement Learning agent is to maximize the policy π such that:

$$max\, E\left\{\sum_{n=1}^{T} \lambda^{n} R_{xi}(S(n), S(n+1))\right\}$$

where $\lambda^{n} \epsilon [0, 1]$ is the discount factor.

In reinforcement learning problems, the discount factor determines how much the agent cares about rewards in distant futures compared to more immediate ones. If $\lambda = 0$, the agent will be completely short-sighted focusing only on actions that produce an immediate reward. When instead, $\lambda = 1$, the agent will evaluate his actions as a function of the sum of all its future rewards.

The discount factor is often associated with the time horizon of the context to which the model is applied. Longer time horizons usually present greater variance as they include more irrelevant information, while shorter ones are biased only towards short-term gains.

For example, by applying reinforcement learning to trade in the stock market, common sense would suggest making profit as soon as possible in order to lock them in and keep free liquidity in order to exploit future opportunities.

As suggested in "Markov games as a framework for multi-agent reinforcement learning" by Michael Littman, 1994, the discount factor can be even thought of as the probability that the "game" will be allowed to continue after the current move.

3.9 Exploration and Exploitation

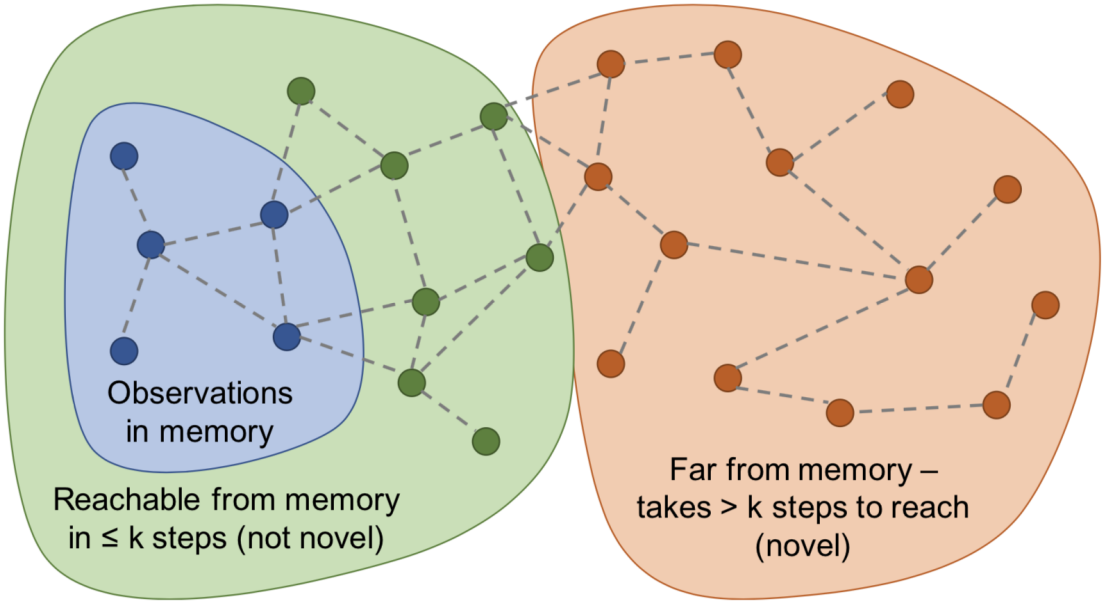

Image 5.1 The graph highlights the exploration vs exploitation trade-off in Reinforcement Learning. In the graph the nodes are the states, the edges are the potential transitions.

Image source: Savinov, et al. 2019

For every action made in a given state, a Reinforcement Learning agent has to balance the exploration vs exploitation tradeoff. It can choose between undergoing actions that have already been explored to catch higher rewards or exploring unknown actions in order to discover new states and then higher rewards.

The idea behind curiosity-driven exploration is to stimulate the agent to investigate unknown outcomes in order to potentially find the optimal solutions operating on the reward function.

Bringing the agent to have an exploratory behaviour, RL agents collect data using a stochastic policy, such as Gaussian distribution in continuous-action spaces or Boltzmann distributions in discrete ones.

Different tailored reward functions and solutions have been researched to accomplish the task at hand.

3.10 Reinforcement Learning algorithms overview

In this thesis project different algorithms have been used to train the agent on the same environment. This chapter aims to give a detailed overview of these algorithms and their peculiarities.

3.10.1 PPO: Proximal Policy Optimization

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Image 3.3: PPO algorithm pseudocode

Source: https://arxiv.org/pdf/1707.06347.pdf

One of the problems that reinforcement learning suffers from is that the generated training data is itself dependent on the current policy since our agent is generating its own training data by interacting with the environment rather than relying on a static data set as in the supervised case. This means that the data distributions of our observations and rewards are constantly changing as our agent learns which is a major cause of instability in the whole training process. Apart from that reinforcement learning also suffers from a very high sensitivity to hyper parameter tuning. To address these problems, the OpenAI team designed a new algorithm called Proximal Policy Optimization algorithm.

The core purpose behind PPO was to strike a balance between ease of implementation, sample efficiency and ease of tuning. Vanilla Policy Gradient methods define the policy gradient laws as the expectation over the log of the policy.

$$L^{PG}(\theta) = \hat{E}_t\left[log\ \pi_\theta(a_t \mid s_t)\ \hat{A}_t\right]$$

where At is an estimator of the advantage function at timestep t and πθ is a stochastic policy.

The Advantage Function basically tries to estimate the relative value of the selected action in the current state.

$$A_t = Weighted\ Sum\ of\ Rewards - Baseline$$

Where the discounted sum of rewards is a weighted sum of all the rewards the agent gained during each time step in the current episode. The baseline, or the value function, is an estimation of the discounted sum of rewards from this point onward.

PPO simplifies the TRPO implementation which tries to constrain the new updated policy into the trust-region of the old policy, defining a probability ratio between the new policy and old policy.

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta\ old}(a \mid s)}$$

PPO, in order to solve the instability and slow convergence problem brought by the TRPO algorithms, impose this ratio to be into an interval $[1 - \epsilon,\ 1 + \epsilon]$ (set to 0.2 in the original paper).

The resulting objective function is formulated as:

$$J^{CLIP}(\theta) = E\left[min\left(r(\theta)\hat{A}_{\theta old}(s,\ a),\ clip\ (r(\theta), 1 - \epsilon,\ 1 + \epsilon)\hat{A}_{\theta old}\ (s, a)\right)\right]$$

It takes the expectation of the minimum between the Normal Policy gradient objective which pushes the policy towards actions that yield a high positive advantage over the baseline and a clipped version of the first term.

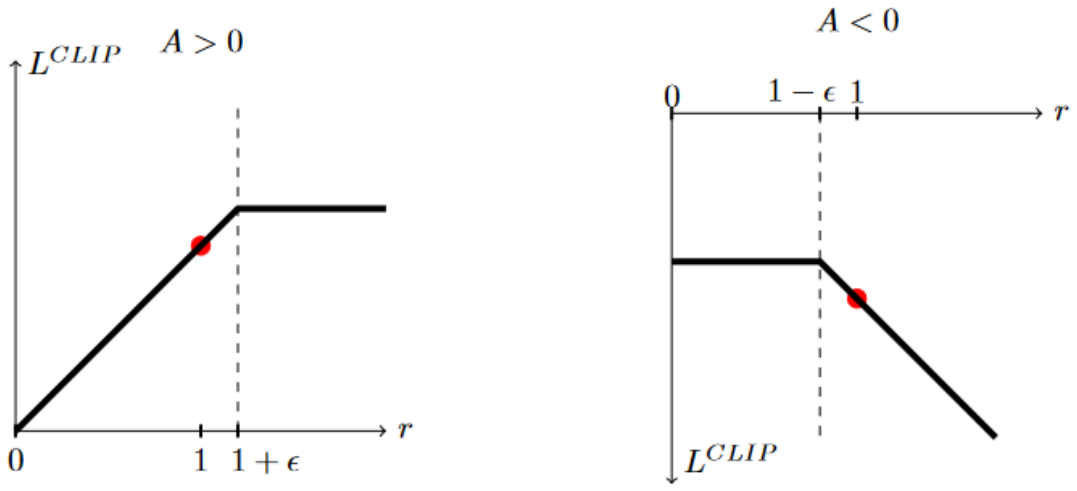

Image 3.4 L_{clip} function evaluated for A>0 and A<0.
Source: https://arxiv.org/pdf/1707.06347.pdf

In the first case, the current action in the current timestep t, yielded better than the expected return while in the second case it did not. We can notice that the Loss Function flattens out in the case of actions with excellent or really bad results so as not to change the old policy too much due to a single estimate.

3.10.2 A2C: Advantage Actor Critic

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Image 3.5 Q Actor Critic pseudocode

Source: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

A3C has been introduced by "*Asynchronous Methods for Deep Reinforcement Learning*" Deepmind article and it is essentially asynchronous parallel training, in which several workers in parallel environments update a single global value function separately in order to explore the state space more efficiently.

A2C is a single-worker variation of A3C, so without the asynchronous part. A2C produces comparable performance to A3C while being, according to empirical findings, more efficient.

In particular with A2C we combine value based methods and policy-based methods into a single algorithm training two different networks: the actor to control how the agent behaves in a given state (policy-based method) and a critic to measure how good it is performing approximating the value function (value-based method)

Using the Advantage function for the discounted cumulative award from vanilla policy gradients**,** we can formulate the Advantage Actor Critic objective function as:

$$\nabla_{\theta} J(\theta) \sim \sum_{t=0}^{T-1} \nabla_{\theta} \, log \, \pi_{\theta} \, (a_t \mid s_t) \, A \, (s_t, a_t)$$

updating both the actor, with policy gradients and advantage value, and the critic, minimising the MSE with the Bellman update equation, parameters at each timestep.

The advantage function measures how good one action is in comparison to others at a given state, whereas the value function measures how good it is to be in that state.

It has been empirically demonstrated that A2C stabilises the model by reducing the large variance of policy networks.

3.10.3 DDPG: Deep Deterministic Policy Gradient

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Image 3.6 DDPG pseudocode

Source: https://spinningup.openai.com/en/latest/algorithms/ddpg.html

Deep Deterministic Policy Gradient (DDPG) is an algorithm that concurrently learns a policy and a Q-function. In particular it uses off-policy data to learn the Q-function and then, it uses the Q-function to train its policy.

It also takes advantage of the experience-replay technique , learning from the samplings of all the agent experiences accumulated so far, and the slow learning idea from DQN.

DDPG, being based on the Q-learning approach, interleaves learning an approximator for the action-value function and an approximator of the optimal action, thus it is particularly recommended for continuous action-spaces environments.

This algorithm, during each trajectory roll-out, uses sample minibatch from the experience "buffer" to update the value and policy networks through a mean-squared Bellman Error minimization (MSBE), in order to solve the non-independence issue related to the sequentiality of data. The buffer replay must contain old experiences even if they are due to an old policy since the Bellman Equation does not take into consideration the single action transition but has to be satisfied for the aggregation of all the actions stored.

DDPG is in fact an actor-critic algorithm meaning that it trains two distinct networks at the same time. The actor decides the action to be taken in the given state in a deterministic way while the critic evaluates the state-action pairs.

The value network is updated similarly as in Q-learning but, in DDPG, the next state Q-value is calculated through the target-value network and target-policy network.

To calculate the policy loss, we take the mean of the gradients calculated for each minibatch, as:

$$\nabla_{\theta\mu} J(\theta) \approx \frac{1}{N} \sum_{i} \left[\nabla_{a} Q\left(s,\ a \mid \theta^{Q}\right)\big|_{s = s_i,\ a = \mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\big|_{s = s_i} \right]$$

Since DDPG learns a deterministic policy in an off-policy way, exploration could result in being not that efficient to find useful learning information for the agent. To solve this problem, DDPG adds a noisy parameter in the training phase. The official paper suggests using a mean-zero Gaussian Noise.

3.10.4 TD3: Twin Delayed DDPG

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

Image 3.7 TD3 pseudocode

Source: https://spinningup.openai.com/en/latest/algorithms/td3.html

Like many RL algorithms, DDPG can be unstable and often rely on finding the correct parameterization for that particular application and environment, due to the overestimation of the Q values of the Critic Network leading eventually the agent to fall in local optima.

TD3. also referred to as Clipped Double-Q Learning, tries to address these problems implementing two Q-functions instead of one and allowing then using smaller to construct the target. Underestimation does not propagate through the algorithm, unlike overestimation, so it brings to the model overall stability.

Then, in order to avoid poor-policy overestimation, in TD3 the policy is updated less frequently than the Q-function. The reference paper suggests one policy update every two Q-function updates. In this way TD3 provides noise to the target action, making it more difficult for the policy to exploit Q-function faults.

TD3, also referred to as Clipped Double-Q Learning, is an algorithm where the agent learns two Q-functions instead of one and the Bellman Error Loss functions uses the smaller to construct the targets. In TD3 the policy is updated less frequently than the Q-function. The reference paper suggests one policy update every two Q-function updates. In this way TD3 provides noise to the target action, making it more difficult for the policy to exploit Q-function faults.

In the end, TD3 applied a regularisation technique called Target Policy Smoothing to reduce the variance in the critic. In particular it adds a small clipped-random noise to the target averaging through mini-batches.

After applying the noise, the action is bounded into a valid action range. It can be formulated as:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

These techniques help to solve the DDPG problem where a Q-approximator used to develop an acute peak for an action bringing the policy to quickly exploit it learning wrong behaviours.

TD3 resulted in a significant improvement in performance over the baseline DDPG and is broadly used in multiple RL research applications.

3.10.5 SAC: Soft Actor Critic

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|\, s\right)\right),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

Image 3.8 SAC pseudocode

Source: https://spinningup.openai.com/en/latest/algorithms/sac.html

SAC (Soft Actor Critic), or off-policy actor-critic that can be seen as a maximum-entropy version of DDPG.

It has been recently created as a solution to the most common DRL critical issues: sample complexity and hyper-parameterization sensibility. It is implemented as an entropy RL framework where the agent aims to maximize both expected return and system entropy.

SAC takes the double Q-trick from the PPO (they have been published almost concurrently) but the main feature of this algorithm is the entropy regularization. It allows the agent to get, foreach timestep t, a bonus reward proportional to the policy distribution entropy.

This is closely related to the exploration-exploitation tradeoff: this way the agent is able to quickly explore the environment in order to gain useful learning signals and at the same time prevent the policy from converging to a local-optimum.

The new SAC reinforcement learning problem could be at this point formulated as the following optimization problem:

$$\pi^{*} = arg\,max_{\pi} E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\left(R\left(s_{t}, a_{t}, s_{t+1}\right) + \alpha H\left(\pi\cdot|s_{t}\right)\right)\right]$$

Where $\alpha > 0$ is a coefficient that coordinates the trade-off between the reward and the policy entropy.

The Value-Function V^{π} changes to:

$$V^{\pi}(s) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\left(R\left(s_{t}, a_{t}, s_{t+1}\right) + \alpha H\left(\pi\cdot|s_{t}\right)\right)|s_{0} = s\right]$$

as the Q^{π} which is modified to implement the entropy bonuses:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\left(R\left(s_{t}, a_{t}, s_{t+1}\right) + \alpha H\left(\pi\cdot|s_{t}\right)\right)|s_{0} = s, a_{0} = a\right]$$

Similarly to TD3 the Q-Value Functions are learned concurrently regressing to the same target using the MSBE minimization. On the other hand, TD3 takes advantage of the current policy to choose the next-state action, unlike TD3 that instead uses the target-policy. Then, since SAC learns a stochastic policy, that noise is sufficient for the exploration purposes and does not require any further noise to be added to the next-state action. During the test phase, the paper suggests using the mean action instead of a random action from the policy distribution to improve the performances over the stochastic policy.

4. Deep Learning Overview

Deep learning is a subset of a larger class of machine learning methods based on artificial neural networks and representation learning that can be categorised as supervised, semi-supervised, or unsupervised learning.

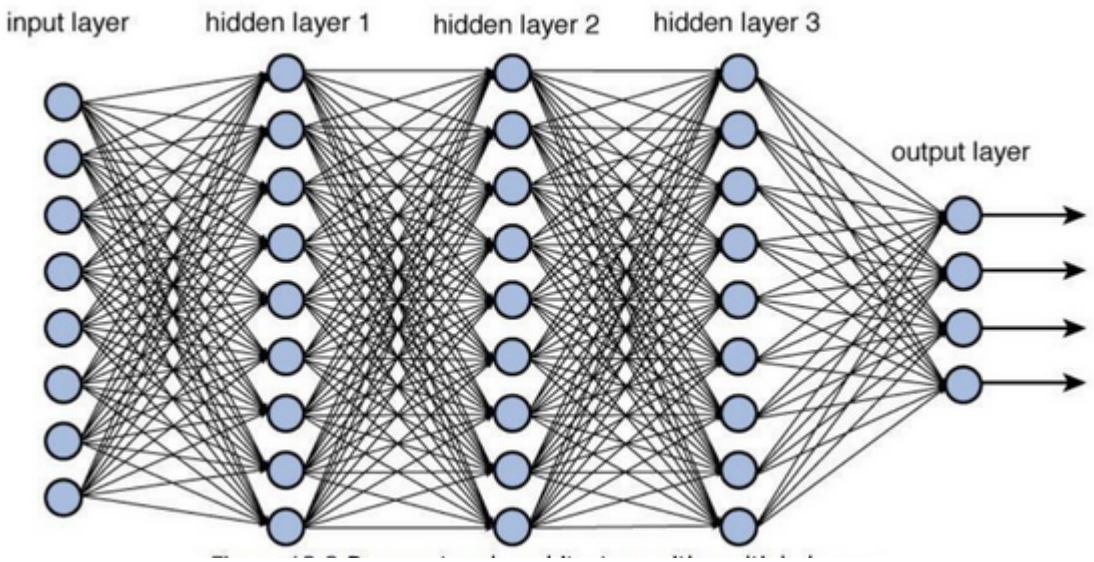

Figura 4.1 A Deep Neural Network structure made of input layers, multiple levels of input layers and an output layer.

4.1 Perceptron

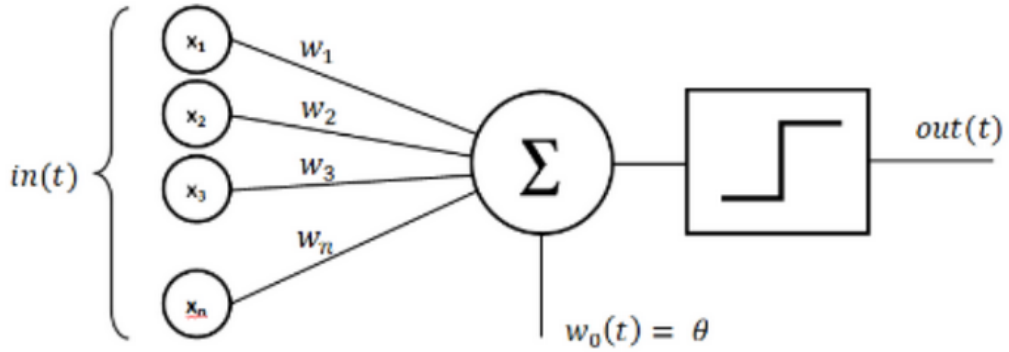

Image 4.2. A NeuralNetwork perceptron visual representation

Source: https://commons.wikimedia.org/wiki/File:Perceptron_moj.png

Frank Rosenblatt introduced the artificial perceptron idea in 1957 starting from the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers that allows neurons to learn and process elements from the training set one at a time.

The Perceptron algorithms learn the weights for the input signals in order to draw a linear decision boundary. While single layer perceptrons can learn only to distinguish linearly separable patterns, multilayer perceptrons or feedforward Neural Networks with two or more layers have a greater processing power.

4.2 Loss Function

The Loss Function (also known as the Error/Cost/Objective function) is a technique for determining how well an algorithm models a dataset. When the model predictions are completely off the mark, the loss function will return a large number while, if the predictions are correct, the loss function will return a lower value.

As changes are made to the model, the loss function is the best indicator of whether the algorithm is heading in the right direction.

In this chapter we will propose some of the more popular loss functions which tend to produce the most accurate results.

4.2.1 Mean Squared Error Loss

Mean squared error is calculated as the average of the squared differences between the predicted and actual values. The outcome is always greater than 0 and the optimal outcome is 0.0.

The squaring brings to the fact that the biggest errors result in greater Loss for the model than smaller ones, implying that the model is punished for making larger mistakes.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

4.2.2 Mean Absolute Error

Absolute Error is calculated as the absolute difference between the predicted value and the target value for each data point in a dataset.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

4.2.3 Cross Entropy Loss

Log-Loss is a popular loss function for classification problems and it is a logarithmic version of the likelihood function.

$$\mathcal{L} = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

4.3 Activation Function

The activation function in a Deep Neural Network, specifies how the weighted sum of the input is transformed into an output from a node to another. Since many activation functions are nonlinear, its choice has a large impact on the neural network's capability and performance, and different activation functions may be used in different parts of the model and tailored to the scientist task.

Although networks are designed to use the same activation function for all nodes in a layer, the activation function is used within or after the internal processing of each node in the network.

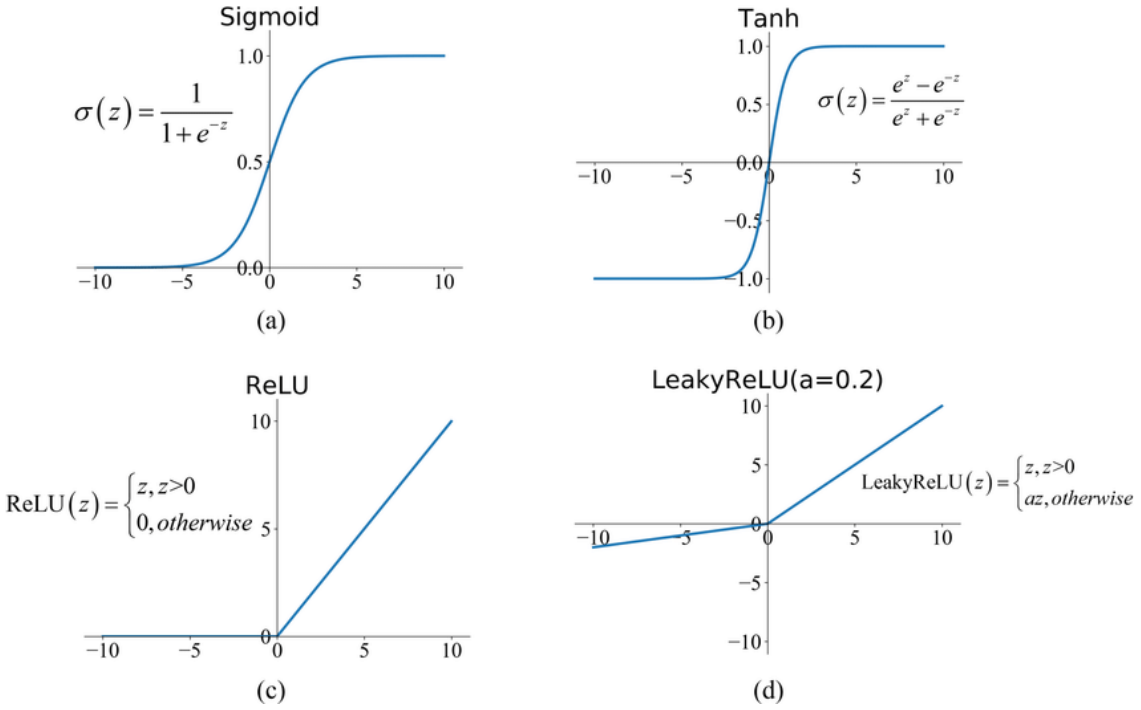

Figure 4.3 The more popular Activation Functions at the state of the art: Sigmoid, Tanh, ReLU and LeakyReLU (a)

Source: https://www.researchgate.net/figure/Commonly-used-activation-functions-a-Sigmoid-b-Tanh-c-ReLU-and-d-LReLU_fig3_335845675

4.4 Backpropagation

Backpropagation ("*backward propagation of errors*"), or reverse-mode automatic differentiation, is a way of computing the gradient for gradient descent and is a widely used machine learning algorithm for training feedforward neural networks despite it has many generalisations for other artificial neural networks.[2]

It is a method of fine-tuning the weights of a neural network, computing the gradient of a loss function with respect to all weights in the network, based on the error rate obtained in the previous iteration, namely epoch. By fine-tuning the weights, one can reduce error rates and increase the model's reliability by increasing its generalisation.

5. Deep Reinforcement Learning

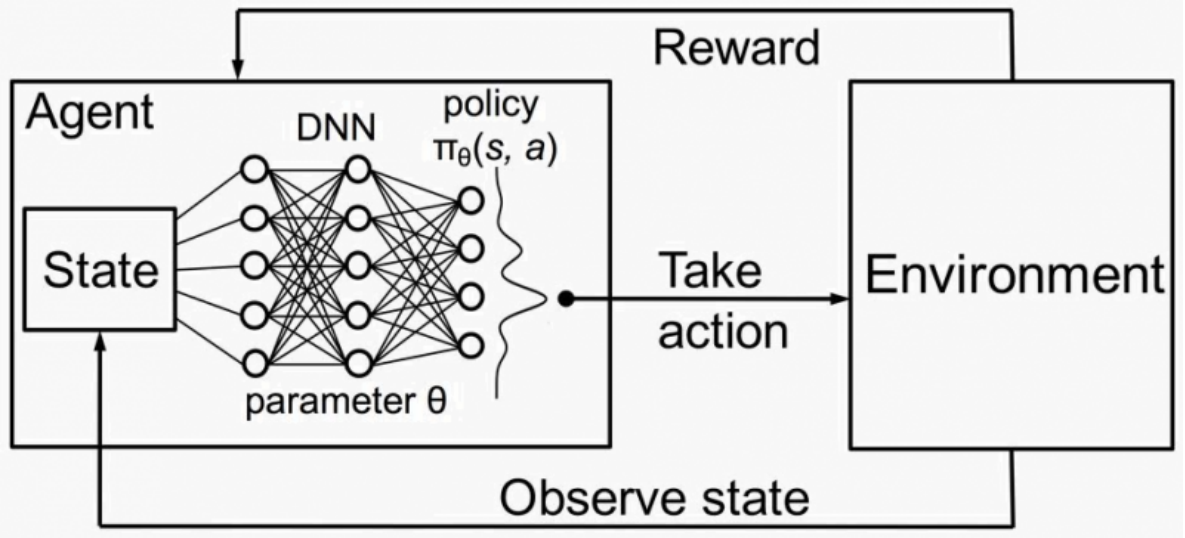

Image 5.1: Schematic view of a deep reinforcement learning agent. Compared to the Reinforcement Learning approach, in this case the policy π is evaluated through a network

Source:
https://medium.com/@vishnuvijayanpv/deep-reinforcement-learning-value-functions-dqn-actor-critic-method-backpropagation-through-83a277d8c38d

Deep Reinforcement Learning (DRL) has caught the interest of the AI community in recent years attaining human-level performance in Atari game play and in controlling 3D locomotion tasks with high dimensional state space DRL's ability to handle high-dimensional state and action space makes it ideal to be used in a financial context.

Deep Reinforcement Learning is still a really active area of research, given its great adaptability to different disciplines such as game theory, statistics and simulation-based optimization in various fields.

In many practical cases, the states of the markov decision process are multi-dimensional and cannot be solved by reinforcement learning algorithms. To solve this problem, deep learning is often used to represent policies, or other functions, through neural networks, using algorithms that can exploit them.

5.1 Deep Reinforcement Learning applied to Financial Data

In academia and the financial industry, quantitative trading has been a popular topic since the late 1960s. In general, it refers to the use of statistical models and data-driven methodologies in financial market analysis, following two main approaches.

The classical approach is related to the development of economic theories in order to interpret the financial market behaviour. A few representative examples include the well-known Capital Asset Pricing Model and Markowitz Portfolio Theory.

On the other hand, computer scientists apply data-driven techniques to analyze financial data. In particular, machine learning and deep learning have been critical components generating a high interest in the finance industry in the last period.

The advancement of these techniques, technologies, and skills has allowed the financial industry to grow at a frenetic pace over the years due to its exceptional performance as well as the appealing property of learning meaningful representations from scratch.

Electronic markets are among the main problems to which quantitative finance techniques have been applied. Electronic markets have emerged, since the 90s, in many countries as a solution for different types of financial products online.

Optimal execution is another fundamental problem in this field. It consists, in the simplest case, of a trader who wants to buy or sell a large amount of certain asset in a predefined period of time, aiming to minimise transaction fees and obtain the optimal return. The purchase of shares has a non-linear impact on the price as a function of the orderbook and has been demonstrated that can be assisted or executed effectively by machine learning and deep learning algorithms.

Assuming that the price of financial products can be derived as a function of the financial state and contracts term, it is possible to model Options Pricing problems and hedging management algorithmically. It has been shown that these methodologies outperform the classic pricing models such as Black-Scholes Model, binomial option pricing and Monte-Carlo simulations.

Most of these adaptive systems rely on supervised learning, which involves the training of a forecasting model on historical data to project market products trend direction. However, despite their popularity, these methods usually have a number of flaws that result in suboptimal results linked to the fact that financial assets trading is not only a process aiming to estimate future prices: it also requires many other elements that should be taken into account, such as the risk involved as well as exogenous constrictions like transaction fees and market liquidity. On the contrary, the supervised model aims to minimise prediction error (profit maximisation) regardless of risk, which is not in the investor's best interests.

Furthermore, as financial market data is extremely noisy, the use of an algorithm with a large learning capacity, such as Neural Networks, in this context will most likely result in overfitting.

In the Reinforcement Learning context, conversely, the algorithm is not given any explicit supervision: it relies on the rewarding function to guide the agent and evaluate how it is behaving in the given environment.

The agent in this context is a trader, the actions are the amounts of shares to be traded, and the environment is the stock's price movement.

5.2 DRL Libraries for Finance

Getting hands-on experience is appealing to beginner traders since deep reinforcement learning (*DRL*) has become one of the main research topics in quantitative finance.

Below some of best recognized open-source DRL libraries available:

- OpenAI’s Gym: is a package that allows the creation of custom reinforcement learning agents. It comes with quite a few pre-built environments in different contexts.

- Google Dopamine: GD is a research framework for fast prototyping of reinforcement learning algorithms. It features pluggability and reusability.

- RLlib: RLlib is an open-source reinforcement learning library that supports APIs for a wide range of industry applications, from video games to financial trading.

- Horizon: Horizon is Facebook's open source applied reinforcement learning platform. It is designed to solve industry applied RL problems with large datasets and slow feedback loops.

6. FinRL

FinRL is a deep reinforcement learning (DRL) library by AI4Finance-LLC that allows users to build quantitative financial analysis tools and develop their own custom stock trading strategies with different fine-tuned DRL algorithms.

Their three primary principles listed in its official research paper are:

1. Completeness: This library completely covers all the major DRL framework.
2. Reproducibility: FinRL ensures transparency on different implementation levels to provide users reliability and reproducibility.
3. Hands-on tutorial: detailed tutorial and examples are provided within the library

FINRL is a structure consisting of three layers. At the lowest level there is the environment which simulates markets by working with different types of data starting with the price and its derivatives such as technical indicators (DQN, DDPG, Adaptive DDPG, Multi-Agent DDPG, PPO, SAC, A2C and TD3). In the middle we find the trading agent which runs several fine-tuned algorithms. The last layer consists of the high-level application which trades automatically.

The three trading use-cases provided by FinRL jointly with their environment are:

- *Single stock trading*: the agent is trained to trade a single ticker

- *Multiple stock trading*: the agent is trained on a collection of different stocks at the same time. The agent will be able to buy and sell from that stocks pool independently

- *Portfolio allocation*: the agent is trained to reallocate the entire portfolio at each timestep t

6.1 Architecture

FinRL library consists of a three layer architecture:

- The stock market environment built on OpenAI gym that describes the trading problem
- Deep Reinforcement Learning trading agent, trained and tested on the environment
- Stock trading application to backtest what the agent learned and trade the live strategy on the Stock Market

The agent layer interacts with the environment layer in an exploration/exploitation manner in order to be able to make an already evaluated decision or to explore a new potential action and state and potentially gain higher reward.

Each lower layer provides the APIs higher ones, which makes the library an highly customizable framework.

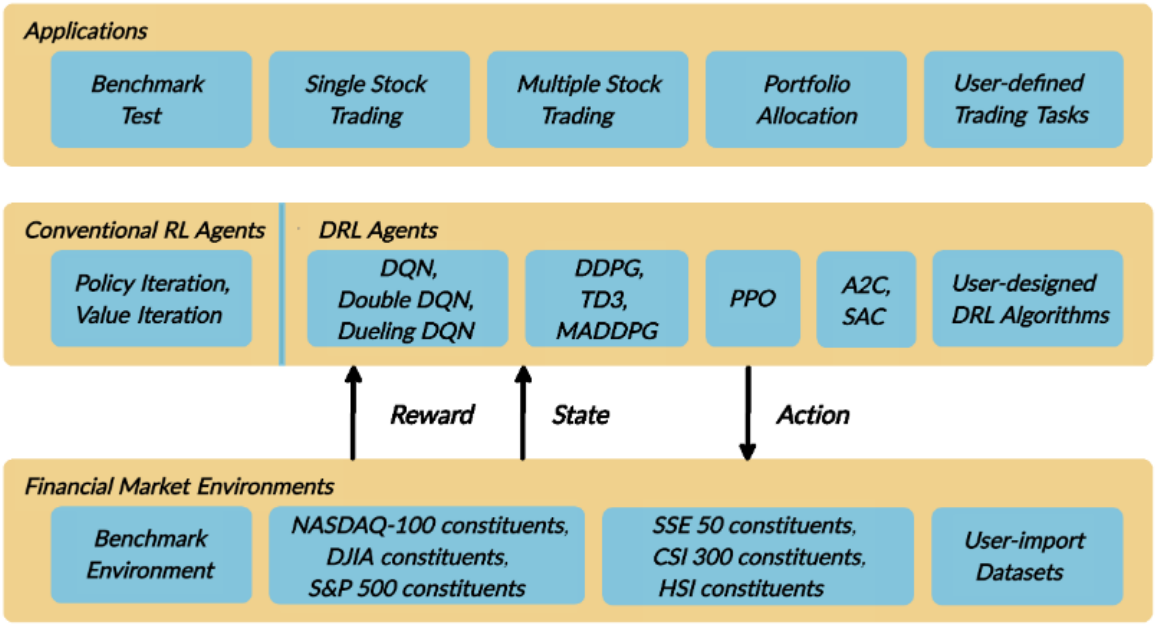

Figure 6.1 FinRLApplications, Agent and Environment layers overview. The environment layer exposes API to the RL agents.

Source:https://arxiv.org/pdf/2011.09607.pdf

Each layer contains several easily customisable modules that can be used for the trading strategy. New modules can be created and added to existing ones.

6.2 Environment: OpenAI Gym

The environment used by FinRL is based on the OpenAI gym. It is modelled as a Markov Decision Process (MDP) due to the stochastic and interactive nature of the automated trading task.

OpenAI helps to simulate the market behaviours based on the principles of time-driven simulation and working with historical price data. This means that the environment for live trading is different from the one used for training purposes. Also because of some risk aversion techniques implemented.

OpenAI also provides well-documented APIs on different levels to build custom environments and to customise the existing ones.

6.2.1 Turbulence Index

The usage of a Turbulence Index is a method to measure the market's systematic risk over time in order to avoid large drawdowns during turbulent periods.

Qualitatively, in FinRL, the financial turbulence is a market structure condition defined taking in to account:

- The amount of volatility in asset prices.
- The current correlation structure (the "decoupling of correlated assets" and "convergence of uncorrelated assets") is violated by asset price movements.

Quantitatively, it can be formulated as:

$$d_t = \left(y_t - \mu\right) \Sigma^{-1} \left(y_t - \mu\right)'$$

where

$$d_t = turbulence\ for\ a\ particular\ time\ period\ t\ (scalar)$$

$$y_t = vector\ of\ asset\ returns\ for\ period\ t\ (1 \times n\ vector)$$

$$\mu = sample\ average\ vector\ of\ historical\ returns\ (1 \times n\ vector)$$

$$\Sigma = sample\ covariance\ matrix\ of\ historical\ returns\ (n \times n\ matrix)$$

If the turbulence index at time t is higher than a predefined threshold, the agent halts and will resume when the turbulence is over.

6.3 Problem Definition: State Space, Action Space, and Reward Function

The main components of the reinforcement learning environment are described below. Starting from these concepts, one can formulate the problem of Single and Multiple Stocks Trading as a maximisation problem:

- Action: the action space is made up of all the actions that can be undertaken by the agent at time t. In the case of Portfolio Trading Problem it will be the new distribution of weights in the portfolio while for Multi Stock Trading, it consists of all the possible combinations of amounts of stocks to buy or sell. In the next chapter, the action space will be de

- State: all the information accessible by the agent at time t are stored in the state, such as the stocks' price and the technical indicators.

- Reward function: $r(s, a, s')$: it defines how the agent should be rewarded for his actions. In the financial context it is usually proportional to the Portfolio Value but can be tailored to different experiments or environments.

Algorithms	Input	Output	Type	State-action spaces support	Finance use cases support	Features and Improvements	Advantages
DQN	States	Q-value	Value based	Discrete only	Single stock trading	Target network, experience replay	Simple and easy to use
Double DQN	States	Q-value	Value based	Discrete only	Single stock trading	Use two identical neural network models to learn	Reduce overestimations
Dueling DQN	States	Q-value	Value based	Discrete only	Single stock trading	Add a specialized dueling Q head	Better differentiate actions, improves the learning
DDPG	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Being deep Q-learning for continuous action spaces	Better at handling high-dimensional continuous action spaces
A2C	State action pair	Q-value	Actor-critic based	Discrete and continuous	All use cases	Advantage function, parallel gradients updating	Stable, cost-effective, faster and works better with large batch sizes
PPO	State action pair	Q-value	Actor-critic based	Discrete and continuous	All use cases	Clipped surrogate objective function	Improve stability, less variance, simply to implement
SAC	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Entropy regularization, exploration-exploitation trade-off	Improve stability
TD3	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Clipped double Q-Learning, delayed policy update, target policy smoothing.	Improve DDPG performance
MADDPG	State action pair	Q-value	Actor-critic based	Continuous only	Multiple stock trading, portfolio allocation	Handle multi-agent RL problem	Improve stability and performance

Image 6.2: RL algorithms provided within FinRL.

Source: https://finrl.readthedocs.io/en/latest/guide/overview.html#implemented-algorithms

7. Thesis Contribution

The research aims to take advantage of the implementation of the metrics associated to the historical returns and the asset allocation into the reward function in order to give to the model a clear view about the trader risk-aversion profile. Being rewarded not only through the results of trading operations in terms of returns, but also as a function of the results stability and, in the case of portfolio allocation, the gini index can allow the agent to gain the information necessary to behave more efficiently.

In order to achieve the result the reward function experimented was:

$$RW^{+} = \frac{w_1 SR + w_2 G + w_3 S}{\sum_{i=1}^{3} w_i}$$

$$Reward = \alpha PV + (1 - \alpha) RW^{+}$$

where:

- $w_i \in [0, 1]$ is the parametric weight given to each portfolio metric. It is bounded in the interval [0, 1] and they add up to 1.
- *SR* is the Sharpe Ratio indicator calculated on the historical returns obtained by the agent since time t
- *G* is the current portfolio Gini Index at time t
- *S* is the historical returns Stability
- $\alpha \in [0 , 1]$ is the weight given to the new Portfolio Value after each step in the *Portfolio Allocation* problem and to the Total Gains (compared to the initial balance) in the M*ultistock Trading* problem.

The idea is that different configurations of parameters can provide, during the training phase, different information to the agent who in turn can learn to behave accordingly to different risk profiles.

In this thesis we also explored the possibility to standardize or normalize the value of the metrics experimenting with both the methodologies.
We stored the metrics historical values and, for each step and metric, we used the (minValue, maxValue) and (meanValue,stdValue) couple calculated on the last 3 training months to respectively normalize or standardize the parameters.

In the test phase, in order to scale the indicators, we used the historical value from the last training quarter.

7.2 Portfolio Management

The task is to develop an automated trading solution for portfolio allocation training the agent on the DOW 30 stocks price dataset.

The stock trading process is modelled as a Markov Decision Process (MDP) so that we are able to formulate the model trading goal as an MDP maximisation problem.

For all the experiments run in this thesis, we assumed no margin, no short sale and no transaction costs. It means that we allocate all of the money in the portfolio and we trade only the selected 30 stocks. Since the assets considered are highly liquid, they can be converted into cash quickly and without potential slippage issues.

As a result, the weight of each individual stock is greater or equal than zero, and the weights of all stocks add up to one. The total portfolio value is calculated as:

$$pValue_t = \sum_i^n w_i lastClosingPrice$$

The Reinforcement Learning Environment used for the Portfolio Allocation problem is defined as:

- Action: the portfolio weight for each stock in the DOW 30. The value is bounded within [0,1]. For every timestep the model re-elaborate every coefficient.

- State: the state space shape is (34, 30) and it is made by price, technical indicators information and a correlation matrix about each stock for the day t.

- Reward function $r(s, a, s')$: it is the reward function defined in the introduction of this chapter. It is the weighted sum of the new Portfolio Value at time t and the weighted average of the portfolio metrics.

- Environment: the environment is the portfolio allocation for Dow 30 constituents at time t.

Protocol 1 Portfolio allocation using deep reinforcement learning (DRL)

Input: s, state space includes covariance matrix for stocks and technical indicators
Output: Final portfolio value
Initialize $P_0 = \$1,000,000$, $\boldsymbol{w_0} = (\frac{1}{m}, ..., \frac{1}{m})$, P_0 is the initial portfolio value, w_0 is the initial portfolio weights, m is the number of stocks in the portfolio;
for $t = 1, ..., n$ **do**
    Portfolio manager DRL observes a state s and outputs a portfolio weights vector $\boldsymbol{w_t}$;
    Normalize the weights $\boldsymbol{w_t}$ to sum to 1;
    Calculate stock returns vector $\boldsymbol{r_t} = (\frac{v_{1,t}-v_{1,t-1}}{v_{1,t-1}}, ..., \frac{v_{m,t}-v_{m,t-1}}{v_{m,t-1}})$, v is the closing price;
    Portfolio incurs period return $\boldsymbol{w_t^T r_t}$;
    Update portfolio value $P_t = P_{t-1} \times (1 + \boldsymbol{w_t^T r_t})$;
end

Image 7.1 Portfolio Allocation Pseudocode using DRL

The reward function used by FINRL's basic model for the Portfolio Allocation problem was defined as:

$$Reward = PV$$

where PV is the Portfolio Value at time t. Instead, for this type of problem, the reward function implemented and tested in this research will be formulated as:

$$RW^{+} = \frac{\frac{w_1 SR + w_2 G + w_3 S}{3}}{\sum_{i=1} w_i}$$

$$Reward = \alpha PV + (1 - \alpha) RW^{+}$$

The aim of the thesis is to demonstrate the possibility of defining different risk profiles by assigning different coefficients to the SR (*Sharpe Ratio*), S (*Stability*) and G (*Gini index*) metrics in the reward function, influencing the agent's behaviour so that act consistently with the chosen parameterization. The alpha parameter, on the other hand, will be used to define the weight of the new portfolio value with respect to the weighted average of the metrics.

7.2 Multi Stock Trading

The second problem taken into account is the Multi Stock Trading Problem. It is modelled as a MDP in order to be able to formulate the solution as a maximisation problem. As in the

Portfolio Allocation Problem case, the agent will be trained on the historical prices of the DOW 30 exploiting different DRL algorithms.

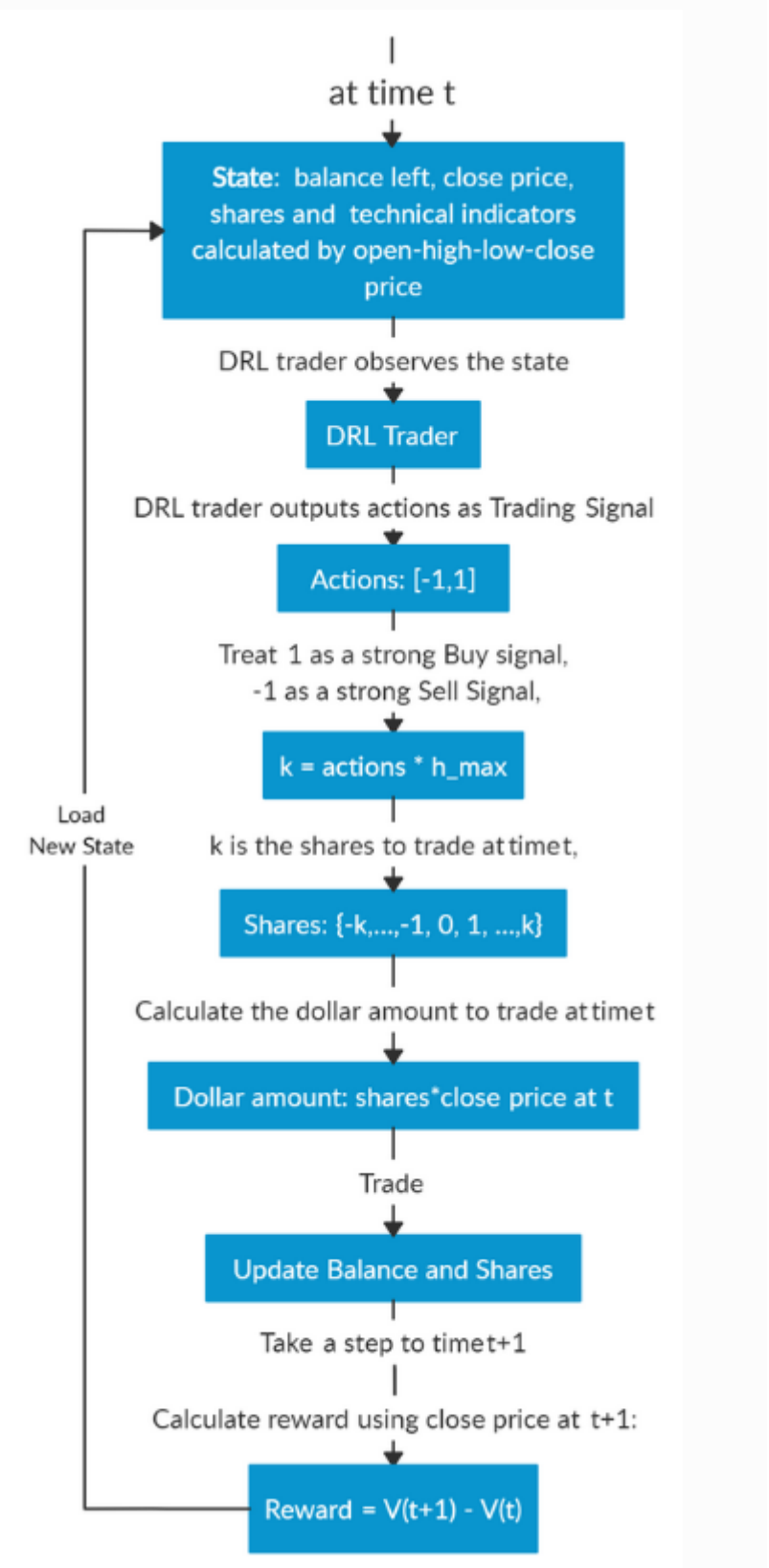

Image 7.2 this image show how the models behave procedurally for each timestep, from the state acquisition to the reward calculation

Source: https://finrl.readthedocs.io/en/latest/tutorial/MultipleStockTrading.html#

The Deep Reinforcement Learning environment components used to solve this problem can be defined as:

- Action: the action space describes the potential action performed by the model at each timestep t. In the Multistock Trading problem the action $a \in \{-k, 0, +k\}$ describe the number of shares to buy, for $k > 0$, to sell, for $k < 0$ or hold for $k = 0$. The action can be carried upon multiple stocks as well.

- Reward function $r(s, a, s')$: it is the reward function defined in the introduction of this chapter. It is the weighted sum of the trading reward at time t and the weighted average of the portfolio metrics. The trading reward is calculated as:
  $V(t) - V(t-1)$ where
  $V(t) = balance\ (t) + dollar\ amount\ of\ the\ stocks\ (t)$.

- State: the state space shape is (34, 30) and it is made by price, technical indicators information and a correlation matrix about each stock for the day t.

- Environment: Dow 30 constituents

In this case, the reward function used in the FINRL AI4Finance's Paper was defined in as the trading reward at time t:

$$Reward = Trading\ Reward$$

In this research we will instead reward the trading agent with the new reward function, defined as a weighted sum of the trading reward on the metric associated to the historical returns.

Contrary to what we saw for the Portfolio Allocation problem, the Gini Index is not considered in the reward function of the Multi Stock trading problem, as it is a metric of a portfolio that can make sense in the case of long-term allocations.

In this case, the new reward function, is thus formulated as:

$$RW^{+} = \frac{w_1 SR + w_2 S}{\sum_{i=1}^{2} w_i}$$

$$Reward = \alpha TR + (1 - \alpha) RW^{+}$$

8. Experiments & Results

The experiments used to validate the hypothesis of this research consist of a set consisting of an experiment on portfolio allocation and multi stock trading, divided by metrics scaling mode, for different algorithms and parametric configurations.

The overall experiment's hierarchy is described in Table 8.1

Table 8.1: experiments hierarchy

Reward Function Parametrization																			
Portfolio Allocation										Multi Stock Trading									
Standardization					Normalization					Standardization					Normalization				
DDPG	A2C	TD3	PPO	SAC	DDPG	A2C	TD3	PPO	SAC	DDPG	A2C	TD3	PPO	SAC	DDPG	A2C	TD3	PPO	SAC

8.1 Data Preparation

In this thesis we used DOW 30 constituents' price and volume data provided Yahoo Finance, downloaded using FINRL's high-level APIs, from January 1st, 2001 till October 31th, 2021 . For each stock the dataframe contains the Date, Open Price, High Price, Low Price, Close Price, Volume and Stock Ticker. All the prices are in Dollars, while the volume is defined in the number of shares traded for that ticker on that particular day.

FINRL uses the class *FeatureEngineer* to validate data, fill the null values and compute technical indicators (MACD, Bollinger Bands, CCI, DX,. SMA30, SMA60). We also process the Covariance matrix, a useful feature to quantify the risk associated with a particular portfolio.

The *lookback* used to calculate the covariance matrix is one trading-year or to 252 trading days.

We performed a number of experiments involving different sets of algorithms and reward functions using the Quantopian pyfolio package to backtest the trading strategies. We assumed to have 1,000,000$ starting capital by 01-07-2021 and we used the model to trade the DOW 30 constituents until 10-31-2021.

8.1.1 Dow Jones Industrial Average

The Dow Jones Industrial Average (DJIA) or simply Dow Jones is a price-weighted measurement stock market index of 30 companies listed on stock exchanges in the United States. The companies listed in the Dow Jones are described in Table 8.1.

However, the DJIA does not use a weighted arithmetic mean and it does not represent its component companies' market capitalization (unlike, for example, the S&P 500). It rather reflects the sum of the price of one share of stock for all the product, divided by the Dow Divisor, a predetermined constant that is used to determine the effect of a one-point move in any of the 30 stocks that comprise the Dow. As of Dec. 27, 2021 the Dow Divisor was 0.15172752595384 [29].

$$DJIA\ Price\ =\ SUM\ (Component\ stock\ prices)/Dow\ Divisor$$

Table 8.2. List of companies in the DOW 30

3M	MMM	1976
American Express	AXP	1982
Amgen	AMGN	2020
Apple Inc.	AAPL	2015
Boeing	BA	1987
Caterpillar	CAT	1991
Chevron	CVX	2008
Cisco Systems	CSCO	2009
The Coca-Cola Company	KO	1987
Dow Inc.	DOW	2019
Goldman Sachs	GS	2013
The Home Depot	HD	1999
Honeywell	HON	2020
IBM	IBM	1979
Intel	INTC	1999

Johnson & Johnson	JNJ	1997
JPMorgan Chase	JPM	1991
McDonald's	MCD	1985
Merck & Co.	MRK	1979
Microsoft	MSFT	1999
NIKE	NKE	2013
Procter & Gamble	PG	1932
Salesforce	CRM	2020
The Travellers Companies	TRV	2009
UnitedHealth Group	UNH	2012
Verizon	VZ	2004
Visa	V	2013
Walmart	WMT	1997
Walgreens Boots Alliance	WBA	2018
The Walt Disney Company	DIS	1991

8.1.2 Gini Index

Several measures are taken by investors to protect their portfolios from risk. Diversification, in particular, is an important way to protect a portfolio and can be achieved including a variety of securities and investments from various issuers and industries.

In this experiment we labelled each constituent of the DOW 30 using the GICS Classification System (section 2.3.7) in order to be able to calculate a global portfolio-Gini Index.

The Gini ratio is infact a measure of statistical dispersion intended to represent the inequality within a population. In our case, we allow the model to be rewarded to have a highly diversified portfolio.

To validate the thesis hypothesis we performed a large set of experiments that can be classified by problem definition, algorithm used, scaling method used to scale the portfolio metrics exploited in the reward function and reward function parameterization for each metric.

8.2 Portfolio Allocation Problem: overall results description by algorithm

Figures 8.1 ,8.2, 8.3 represent the boxplot charts calculated on the Portfolio Allocation problem's results dataset, obtained from the experiments performed using different parameterizations and scaling methods. Each boxplot contains the aggregated results of all the experiments performed for each algorithm. In the calculation of the quartiles, both data scaling methodologies (normalization and standardization) and all the tested parametric configurations were taken into consideration.

Analyzing the distributions of the results, shown in Figure 8.1, we realise that all the algorithms over-performed in terms of sharpe ratio, stability and cumulative results compared to the FinRL baseline, highlighted by the dashed lines. All the medians and upper quartiles are in fact above the baseline.

The baseline was calculated by reproducing the experiments, for each algorithm, proposed by AI4Finance using FinRL without the improvements implemented by the thesis project.

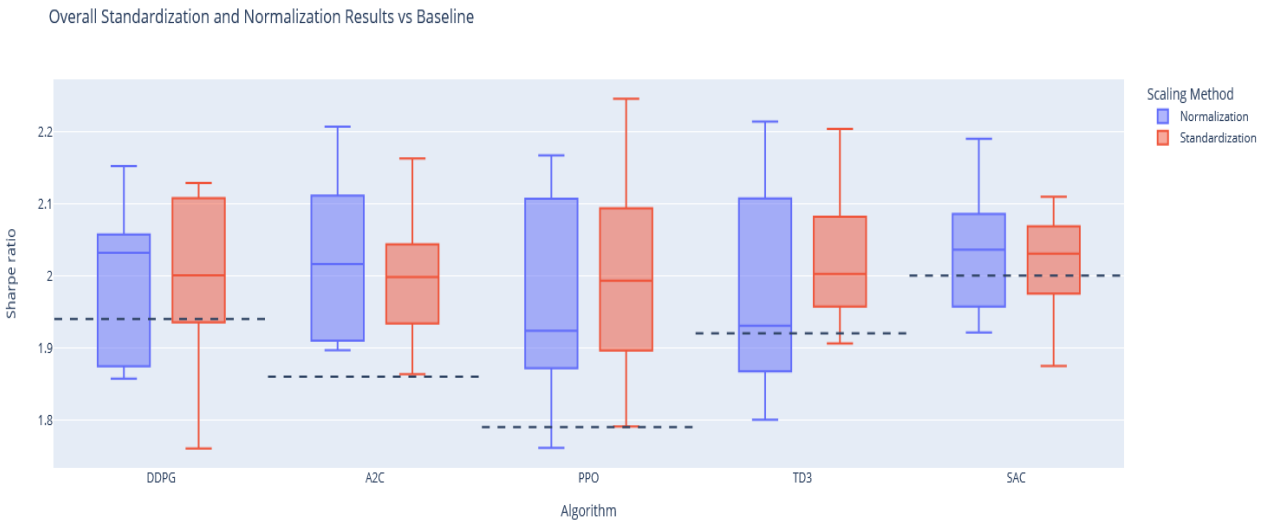

Figure 8.1: Portfolio Overall Standardization and Normalization Sharpe Results vs Baseline

Figure 8.2 describes the algorithm's results in terms of resulting stability, aggregating all the experiments tested for each DRL algorithm. As in the case of the sharpe ratio, also in terms of stability, for almost all algorithms, the medians and upper quartiles exceed the baseline.

SAC is the only one that has not improved compared to the baseline but this result is attributable to the entropic behaviour of the algorithm.

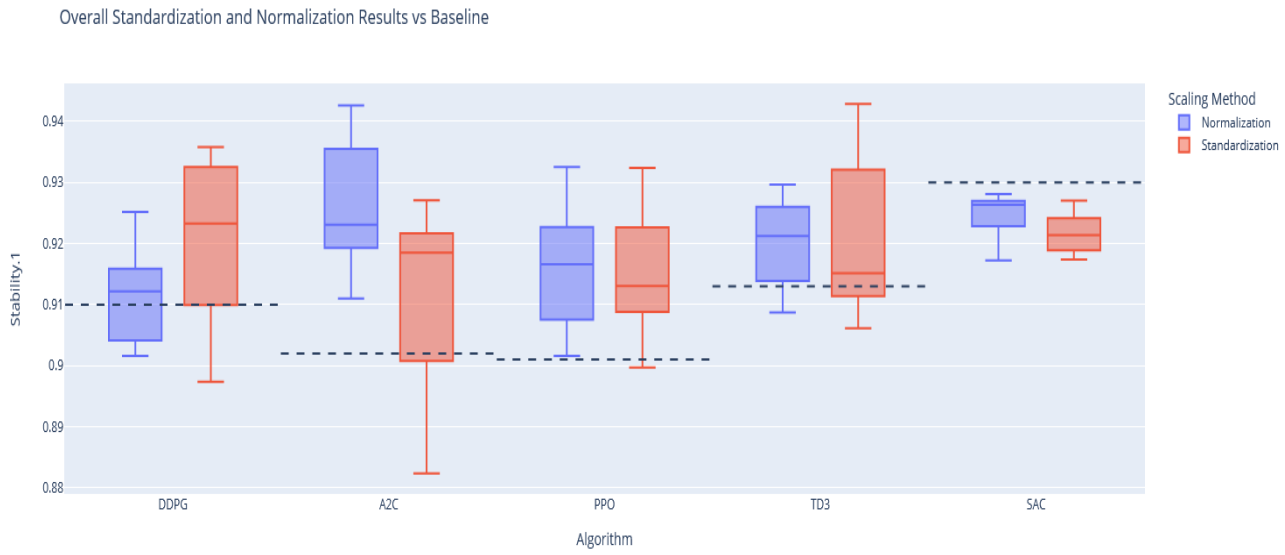

Figure 8.2: Portfolio Overall Standardization and Normalization Stability Results vs Baseline

Figure 8.3 shows how in the portfolio allocation problem, on average all the algorithms overperformed with respect to the baseline also in cumulative returns, calculated on the test set.

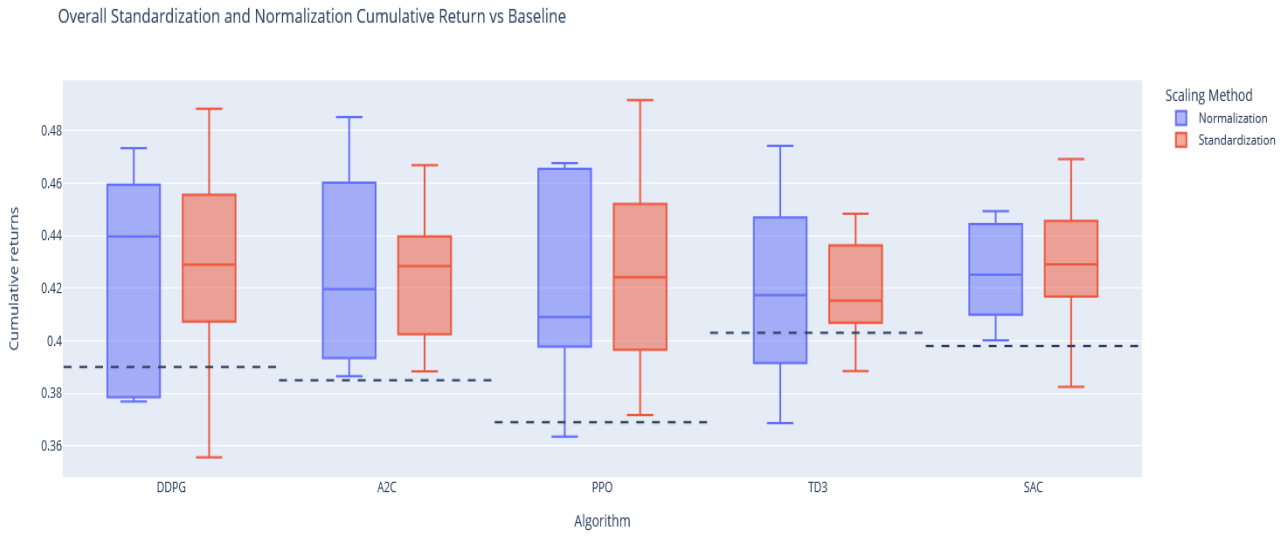

Figure 8.3: Portfolio Overall Standardization and Normalization Cumulative Returns vs Baseline

8.3 Multistock Overall Standardization and Normalization Sharpe Results vs Baseline

Figures 8.4 ,8.5, 8.6 represent the boxplot charts calculated on the Multi Stock Trading problem's results dataset, obtained from the experiments performed using different parameterizations and scaling methods. Analyzing the distributions of the results we realise that all the algorithms over-performed in terms of sharpe ratio, stability and cumulative results compared to the FinRL baseline, highlighted by the dashed lines.

Also in the case of Multistock trading the baseline was calculated by reproducing the experiments, for each algorithm, proposed by AI4Finance using FinRL without the improvements implemented by the thesis project.

From Figure 8.4 we can see that even in the case of multi stock trading, the aggregate results of all the algorithms have overperformed with respect to the FinRL baseline

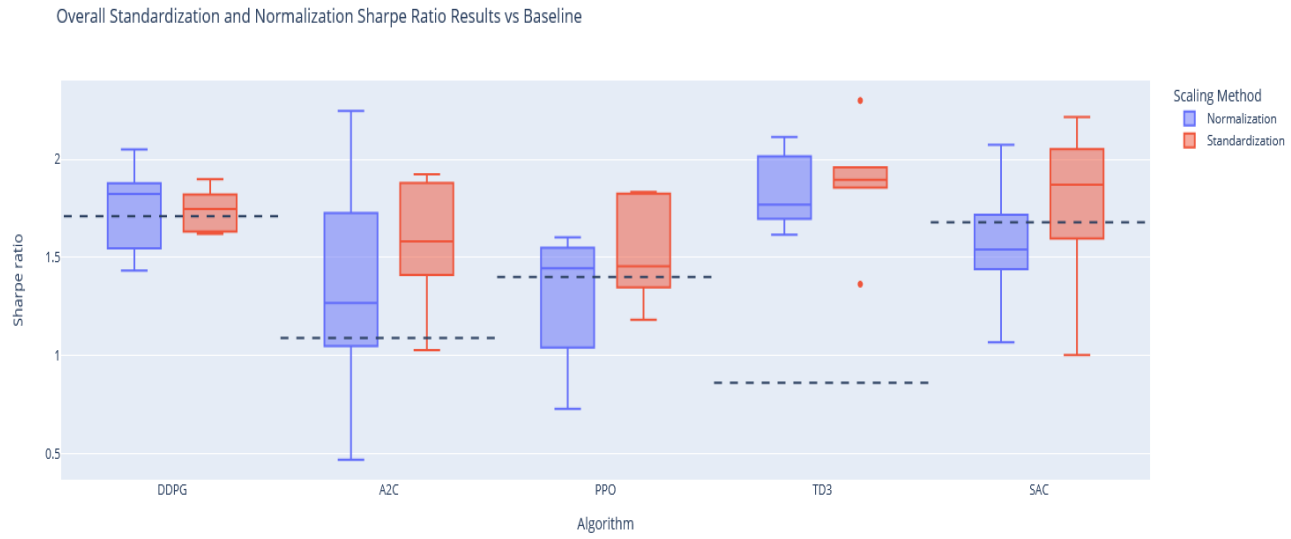

Figure 8.4 Multistock overall sharpe ratio results vs baseline

Figure 8.5 depicts the aggregate results in terms of stability. In this case, they all overperformed except the SAC and the PPO who settled on the same level.

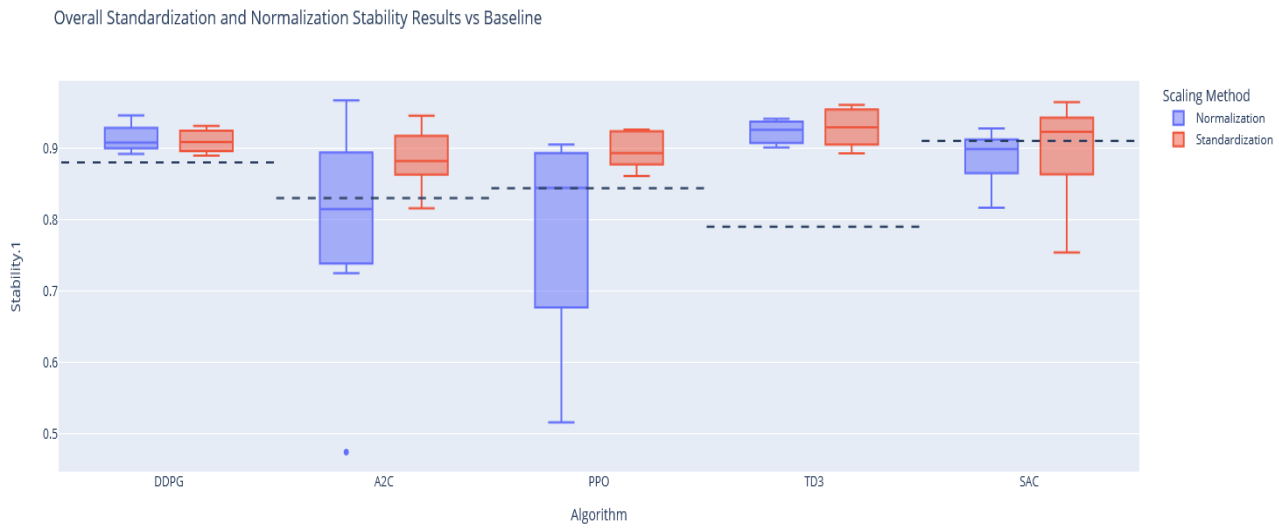

Figure 8.5 Multistock overall stability results vs baseline

From the boxplots chart in Figure 8.6 we can notice that all the algorithms overperformed in terms of cumulative returns compared to those obtained by the baseline, except PPO which is set on average about at the same level.

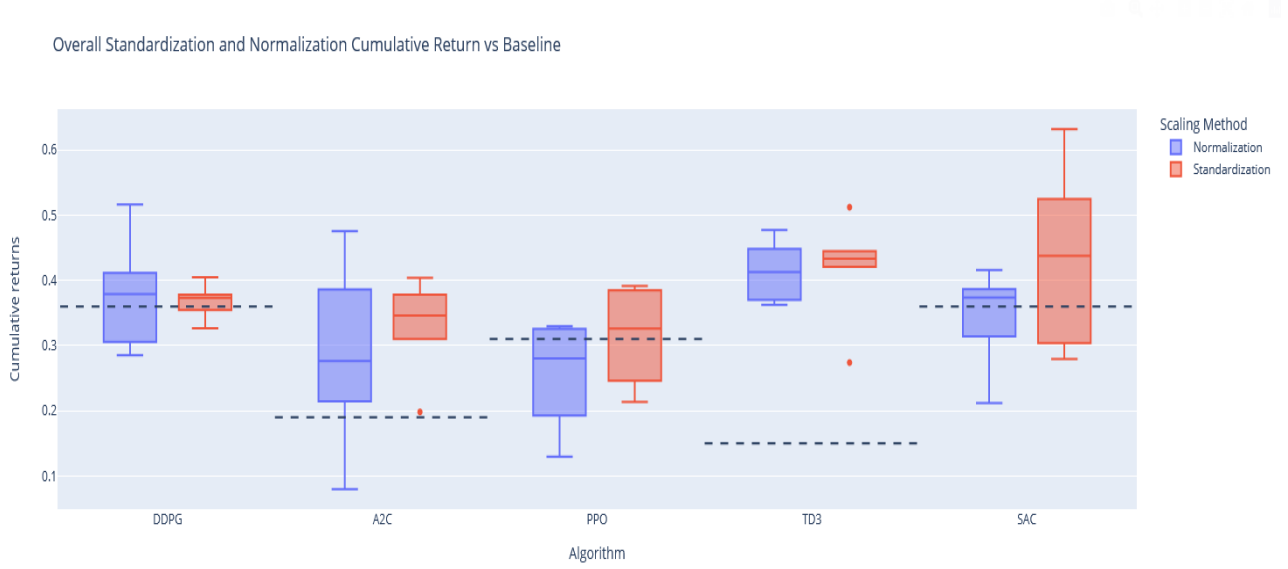

Figure 8.6 Multistock overall cumulative returns results vs baseline

8.4 Sharpe Ratio Metric Overview

Figure 8.7 and Figure 8.8 highlights the results obtained in terms of sharpe ratio for different sharpe coefficients, while alpha is fixed at 0.5. Each boxplot represents the aggregated results for different algorithms and coefficient values of the reward function, given a fixed value of alpha and the coefficient associated with the Sharpe Ratio metric.

Almost every algorithm, both in the case of Portfolio Allocation and Multi stock trading, performed consistently with the values of the coefficient associated with the Sharpe metric, showing a linear correlation between the coefficient and the results in terms of Sharpe Ratio.

It demonstrates that through the new reward function it is possible to accurately interfere with the training of the models, aiming to obtain better returns regardless of the volatility, congruently with the risk profile of the reference trader.

Figure 8.7 shows these results in the case of Portfolio allocation. In particular, the results in terms of sharpe ratio performed linearly with the coefficient associated with the metric sharpe ratio in the reward function. Only for the SAC algorithm, despite having passed the baseline, the results are not linear, due to the entropic nature of the algorithm.

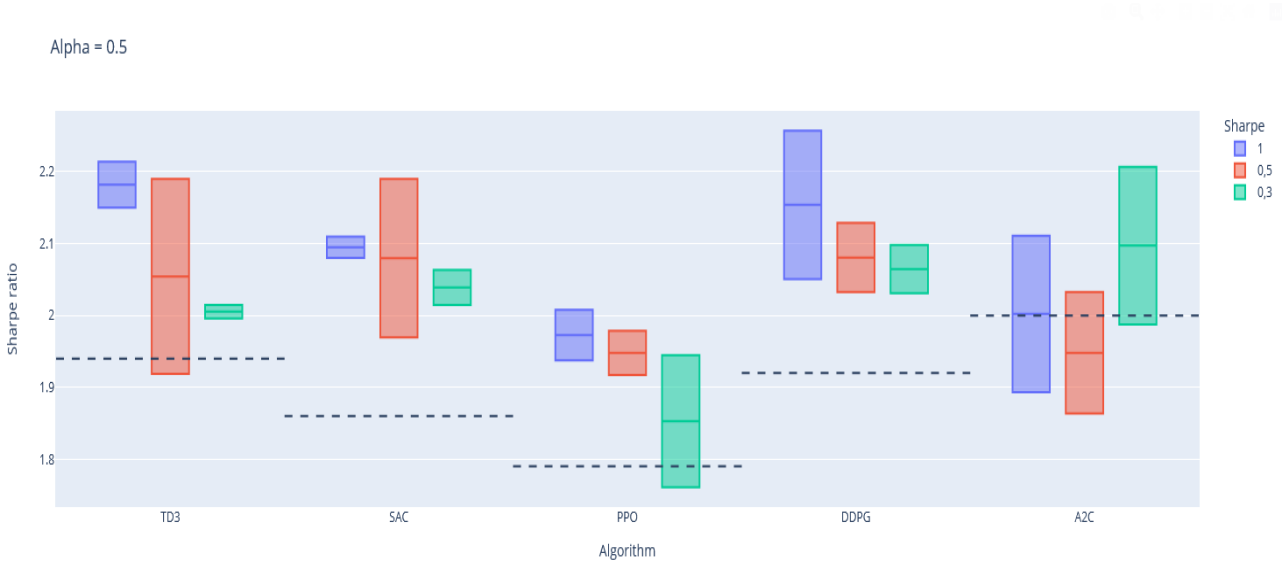

Figure 8.7 Portfolio Allocation Sharpe Ratio results with fixed alpha coefficient equal to 0,5. The representation shows the sharpe ratio distribution by algorithm / alpha coefficient.

In the case of multi stock trading, on the other hand, this linearity is not noticed, as some algorithms for which the coefficient was set to 0.5 showed Sharpe Ratio higher than the others, as we can see in Figure 8.8. We suppose this is due to the fact that the Sharpe Ratio metric is more congenial to a long-term allocation problem than to short-term trading activities.

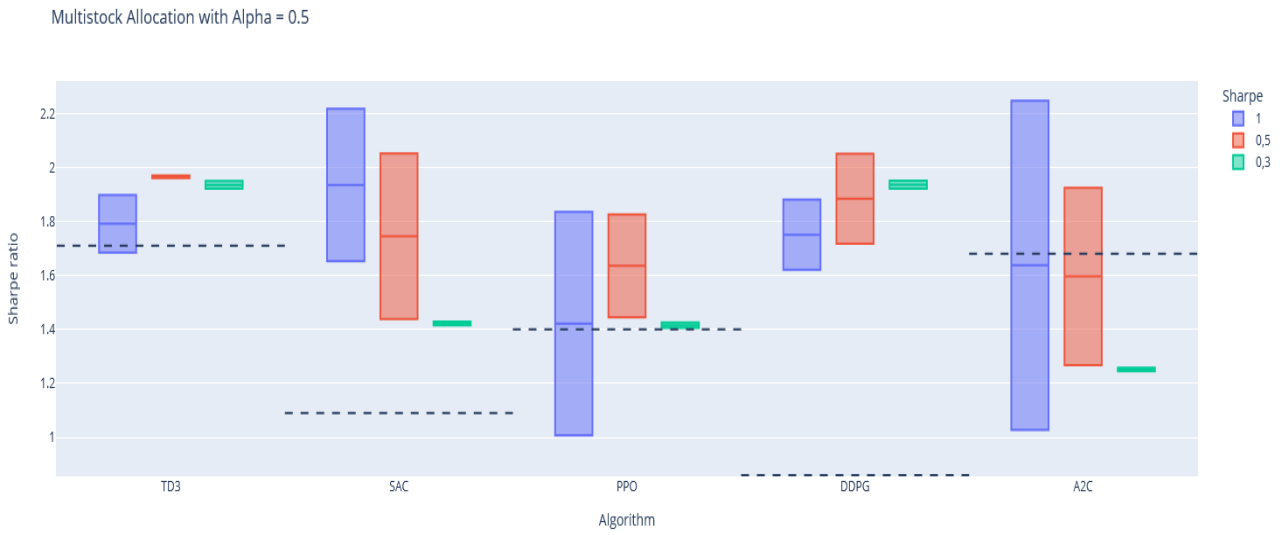

Figure 8.8 Multi Stock Trading Sharpe Ratio results with fixed alpha coefficient equal to 0,5. The representation shows the sharpe ratio distribution by algorithm / alpha coefficient.

8.5 Stability Metric Overview

Figure 8.9 and Figure 8.10 highlights the results obtained in terms of Stability for different stability coefficients, while alpha is fixed at 0.5. Almost every algorithm, both in the case of Portfolio Allocation and Multi stock trading, performed consistently with the values of the coefficient associated with the sharpe, demonstrating that through the new reward function it is possible to accurately interfere with the training of the models.
Only in the case of A2C, the relation does not seem, in this case respected, showing a great variance in its results. Figure 8.9 depicts these results for the Portfolio Allocation problem.

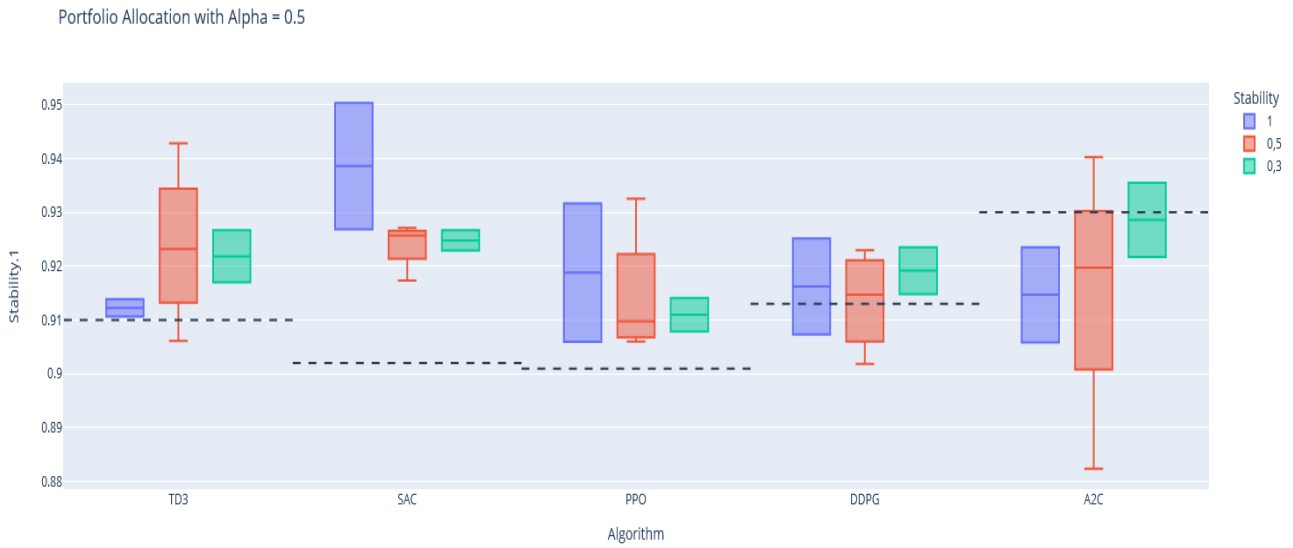

Figure 8.9 Portfolio Allocation Stability results with fixed alpha equal to 0,5. The representation shows the resultant stability distributions by algorithm / alpha coefficient.

As for the sharpe ratio, also for stability, in Figure 8.10 we can see how the results, despite exceeding the baseline in terms of stability, are more variable than what we noted in the case of portfolio allocation. In fact, in the case of PPO, DDPG and SAC, a coefficient equal to 0.7 associated with the stability metric performed better than higher values.

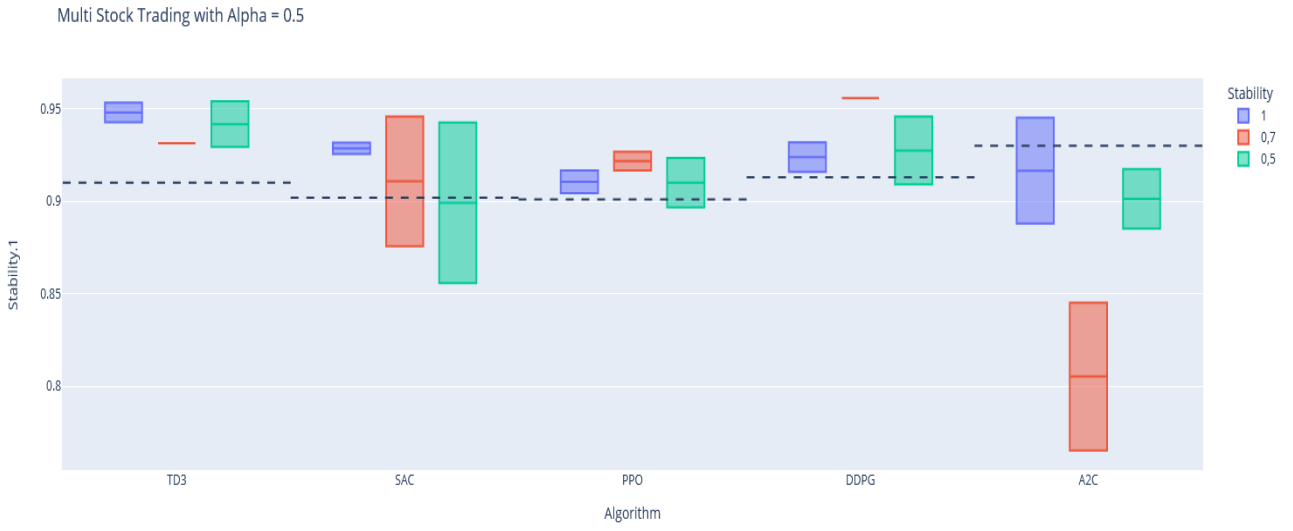

Figure 8.10 Multi Stock Stability results with fixed alpha equal to 0,5. The representation shows the resultant stability distributions by algorithm / alpha coefficient.

8.6 Coefficients Configuration Comparison

The graph shown in figure 8.11 was obtained by aggregating the results of the experiments for different configurations of the reward function, highlighting the results in terms of sharpe ratio.

It is interesting to note that the best results, in the Portfolio Allocation Problem, were obtained by giving the same weight to the new value of the portfolio (Alpha Coefficient equal to 0.5) and to the coefficient associated with the sharpe ratio metric, validating the initial hypothesis of this thesis. Rewarding the agent with this reward function means in fact rewarding him for the profits obtained and at the same time for high levels of sharpe ratios, i.e. returns with respect to risk.

It is followed by the configuration with alpha equal to 0.5 and the same weight associated with the sharpe ratio and stability metrics which, compared with the other configurations, jointly provides the greatest relevance to the sharpe ratio and the value of the portfolio in the reward function.

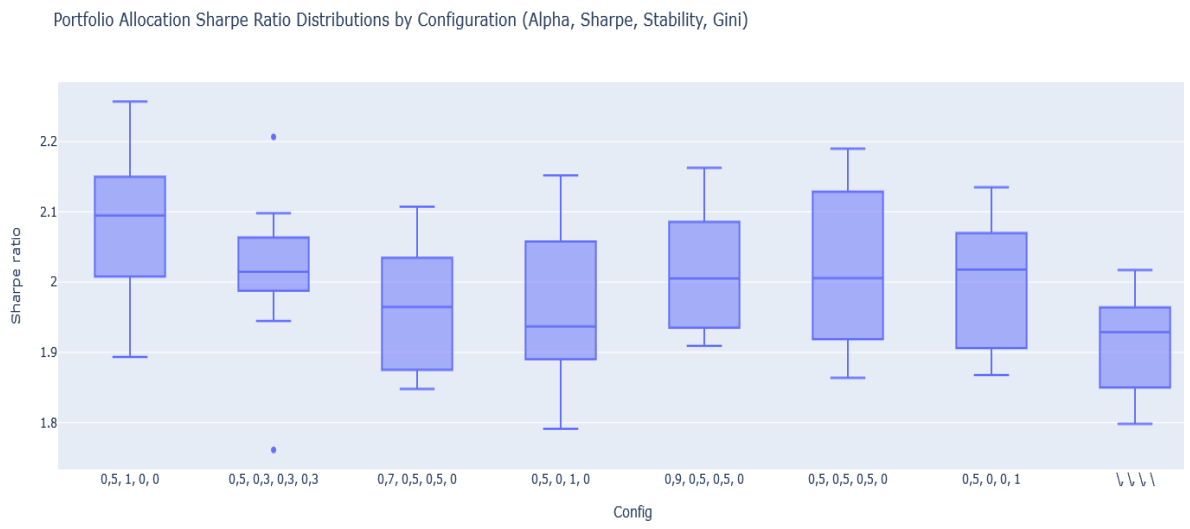

Image 8.11: The image shows a representation of the Sharpe Ratio results distribution given different metric configurations in the reward function for the Portfolio Allocation Problem

The same pattern can be noticed in Figure 8.12 that represents the aggregated results obtained from the experiments on the Multi Stock Trading problem, with the configurations (0.5, 1, 0) and (0.9, 0.5, 0.5) leading the ranking.

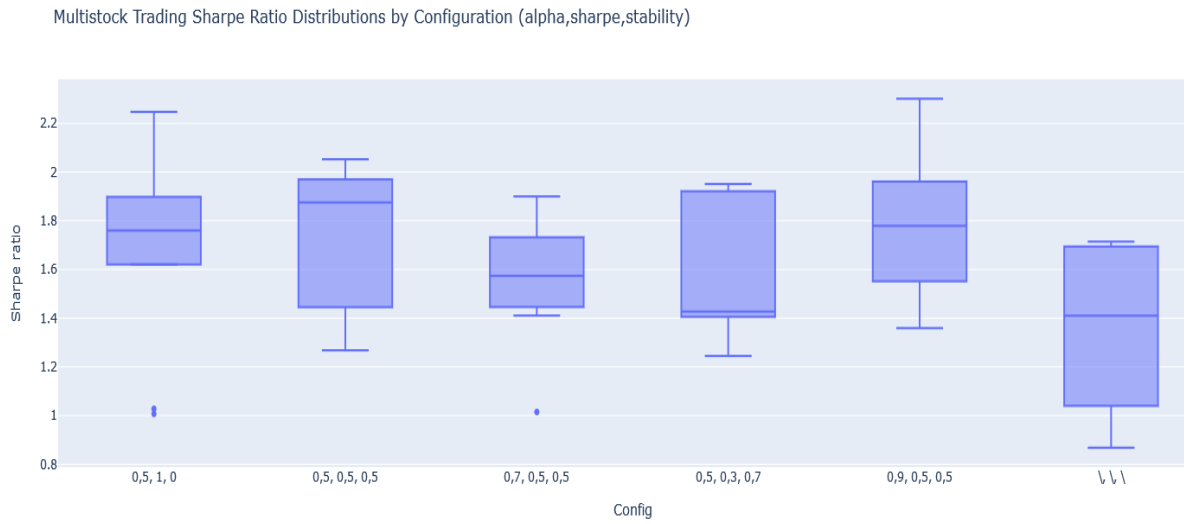

Figure 8.12: The image shows a representation of the Sharpe Ratio results distribution given different metric configurations in the reward function for the Multi Stock Trading Problem

On the other hand, by evaluating the aggregate results in terms of stability, consistently with what is expected given the hypotheses of this research, in Figure 8.13 the configuration that presents the most stable results is (0.5, 0, 0, 1). In this case, in fact, the model is rewarded in the same way according to the value of the portfolio at time t and the stability obtained.

In second place we find the configuration that jointly enhances stability and the Gini Index. Indeed, it is clear that favouring an uncorrelated portfolio, given the industry sector for each stock, can provide greater long-term stability.

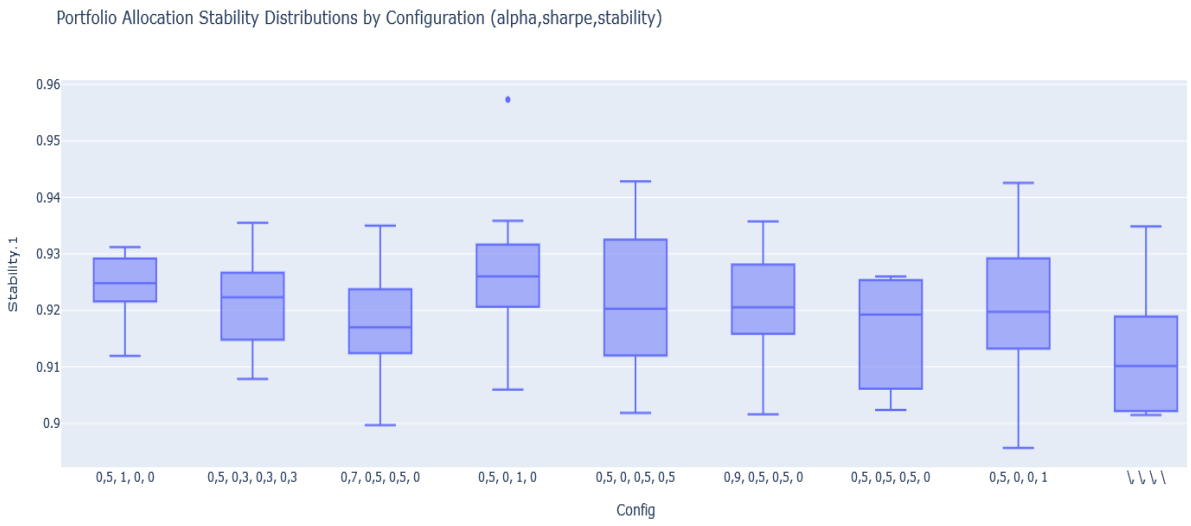

Figure 8.13: The image shows a representation of the Stability results distribution given different metric configurations in the reward function for the Portfolio Allocation Problem

Even in the case of Multi stock trading, the configurations that have performed best in terms of stability are the ones where the influence of stability in the reward function has been prioritised, such as the (0.9, 0.5, 0.5), the (0.5, 0, 1) and the (0.5, 0.3, 0,7)

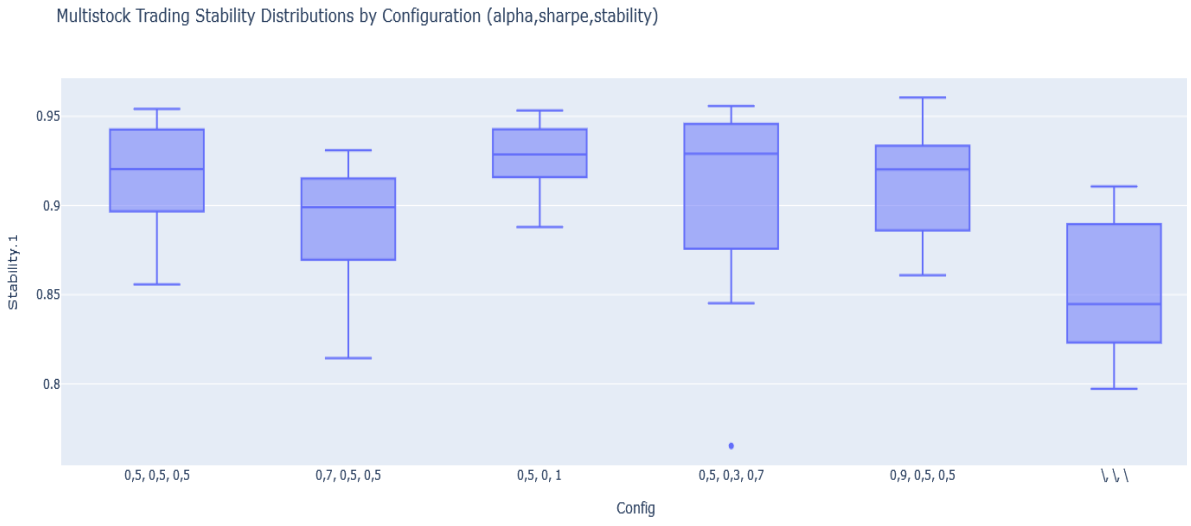

Figure 8.14: The image shows a representation of the Stability results distribution given different metric configurations in the reward function for the Multi Stock TradingProblem

Analyzing the drawdown results show in image 8.15 i.e. maximum observed loss from a peak to a trough in the returns of the portaglio, we deduce that the configuration that produced the best results was the one in which the model was uniquely rewarded for the new value of the portfolio and the value of the gini index.

The Gini Index in fact, pushes the model to avoid over-exposure to a particular market sector in order to be covered from the systematic risk of a large market correction and and therefore drawdowns.

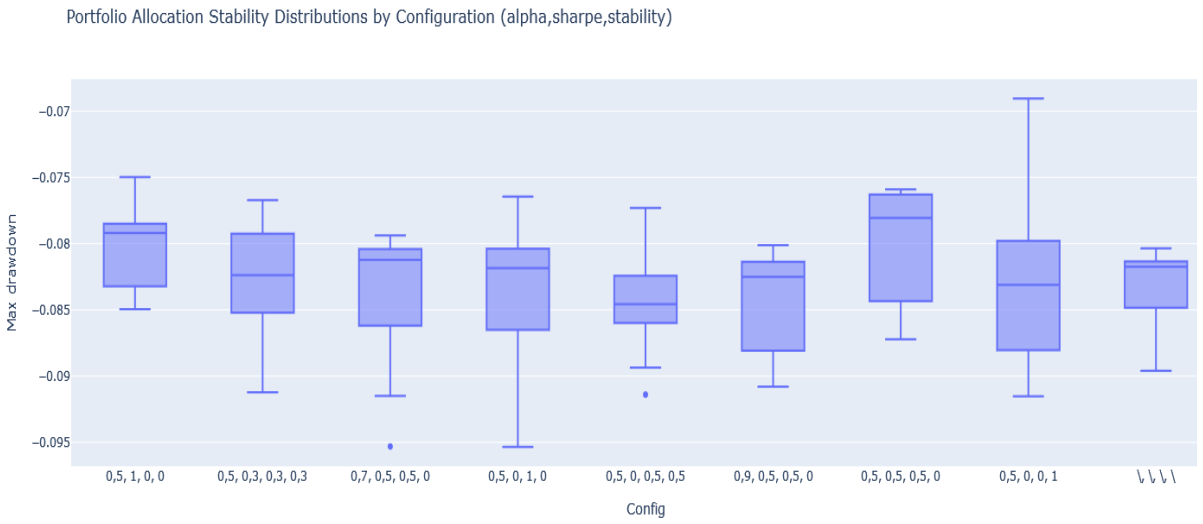

Figure 8.15: The image shows a representation of the max drawdown distribution given different metric configurations in the reward function for the Portfolio Problem

With regards to the Multi Stock trading problem, Figure 8.16 shows that no models significantly outperformed the others. The reason could be associated with the fact that this problem implies faster trades and no obligation to allocate the entire portfolio.

The model that in fact performed slightly better than the others, is the one in which the reward function gave greater weight to the sharpe ratio which pushed the agent to prefer those trades that potentially promised greater rewards given the same level of risk.

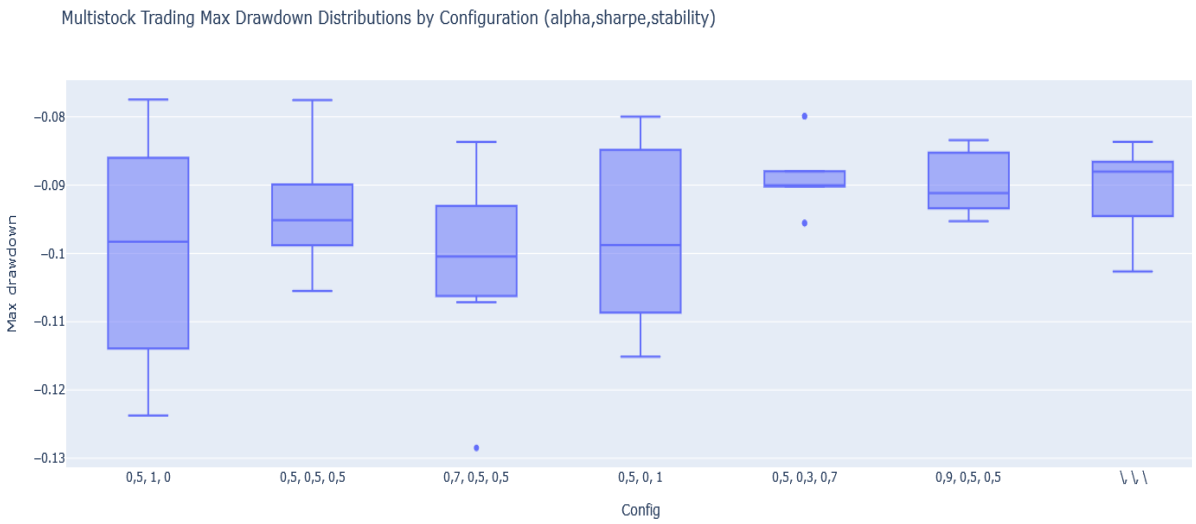

Figure 8.16: The image shows a representation of the max drawdown distribution given different metric configurations in the reward function for the Multi Stock Trading Problem

Finally, drawing the graphs to evaluate the configurations that performed best in terms of annual returns (Figure 8.17), for the portfolio allocation problem we realise that the results almost linearly follow the trend of the coefficient associated with the sharpe ratio.

In those configurations that favour the sharpe ratio metric, in fact, the agent had more room to focus on recognizing the most profitable trades without paying too much attention to the stability or composition of the portfolio at time t.

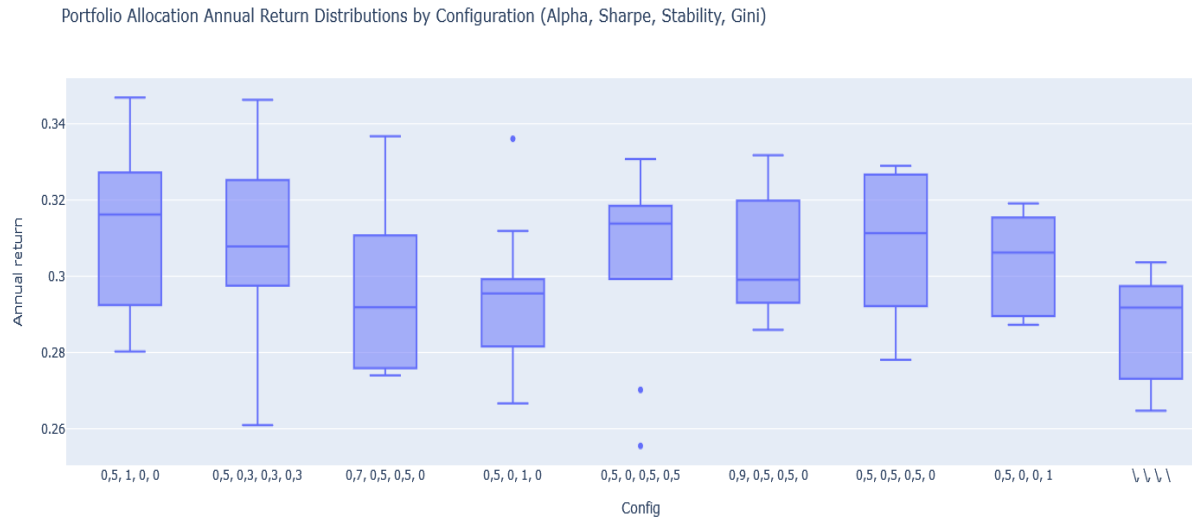

Figure 8.17: The image shows a representation of the annual returns distribution given different metric configurations in the reward function for the Portfolio Allocation Problem

Even in the case of Multi Stock Trading, the most performing configuration was the one associated with the higher sharpe ratio, namely the configuration (0.5, 1, 0). This configuration has in fact exceeded 40% annual return.

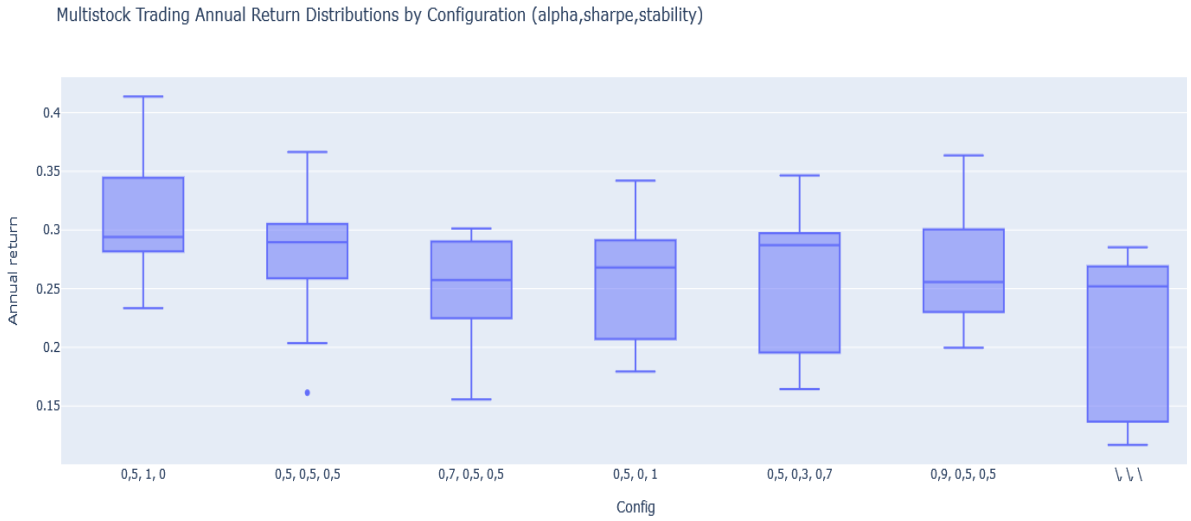

Figure 8.18: The image shows a representation of the annual returns distribution given different metric configurations in the reward function for the Multi Stock Trading Problem

8.7 Best Configuration Comparison

Table 8.3 shows the best performing algorithms results in terms of cumulative returns, both for the Portfolio Allocation and Multi Stock trading problems. Figure8.19, on the other hand, shows the equity line from July 2020 to October 2021, compared with the performance of the DJIA.

Both the represented configurations present an Alpha coefficient equal to 0.5 and Sharpe equal to 1. As shown in the previous chapters, in fact, the performances of the models in terms of cumulative returns responded linearly to the values of the sharpe ratio coefficient, increasing the cumulative returns with the same level of risk.

Table 8.3

Problem	*Multistock*	*Portfolio*
Scaling Method	*Standardization*	*Normalization*
Algorithm	*SAC*	*DDPG*
Configuration		
Alpha	*0,5*	*0,5*
Sharpe	*1*	*1*
Stability	*0*	*0*
Gini	\	*0*
Results		
Annual return	*0.413726*	*0.346904*
Cumulative returns	*0.631723*	*0.470040*
Annual volatility	*0.172428*	*0.145247*
Sharpe ratio	*2.217177*	*2.257061*
Calmar ratio	*3.894910*	*4.279437*
Stability	*0.964523*	*0.911949*
Max drawdown	*-0.113925*	*-0.078025*
Omega ratio	*1.465561*	*1.409963*
Sortino ratio	*3.506085*	*3.194643*
Skew	*-0.052183*	*-0.019835*
Kurtosis	*1.680755*	*1.233719*
Tail ratio	*1.269227*	*1.140191*
Daily value at risk	*-0.020207*	*-0.017114*

Figure 8.9 represents the equity lines of the two best algorithms, compared to the performance of the DJIA.

Figure 8.19: Equity Line comparison between the best performing Portfolio Allocation and Multi Stock Trading models vs DJIA

8.8 Equity Line Comparison

Charts in Image 8.21 and 8.22 show the backtests performed using different parameters configurations of the SAC for Multi Stock Trading and the DDPG for the Portfolio Allocation problem

The backtest was performed on the test period dataset, from July 2020 to the end of September 2021.

As anticipated in chapter 8.4, the parameterizations that prefer a higher weight for the coefficient associated with the sharpe ratio tend to perform better also in terms of Cumulative Returns, outperforming the market during bull markets and limiting drawdowns. On the other hand, parameterizations that favour stability and diversification tend to flatten the equity line, reducing the volatility of returns, protecting capital from sharp market moves.

Figure 8.20: different SAC Configurations' Equity Lines for the multiple stock trading problem

Image 8.20 shows a comparison between the equity lines of different sac parameterizations for the multi stock trading problem. The most performing configuration, as anticipated, was the one where the model was rewarded, equally, through the value of the Sharpe Ratio and the returns at time t. Following, two mixed sharpe-stability configurations that can be associated with medium risk profiles. The model that performed worst was the one awarded only for 15% of the reward function through the sharpe, while the remainder was assessed on the basis of returns and stability.

Figure 8.21: Different DDPG Configurations' Equity Lines for the Portfolio Allocation problem

The same pattern can be noticed in figure 8.21, where a comparison between different DDPG parameterizations for the portfolio allocation problem is shown. The best algorithm in terms of returns in the testing period was, as we expected, the one with alpha equal to 0.5 and the coefficient associated with the sharpe ratio equal to 1, followed by the configuration in which the weight of the reward function was distributed between Sharpe and Stability.

While in the last places, again in terms of cumulative returns, the configurations in which the coefficients of stability and gini index have been preferred. The last configuration in particular showed to be particularly resilient to market drops and stable in periods of high volatility.

9. Conclusions and future works

This thesis work has investigated the use of Reinforcement Learning techniques to address multi-stock trading and portfolio allocation. The idea was to give the reference trader or fund manager the possibility to configure the agent in such a way that it favours a certain style in terms of risk and exposure to the market.

State-of-the-art reinforcement learning algorithms developed have always adopted a market-neutral approach, without providing any contextual information to the agent on the trader's risk profile and the portfolio under management. The methodologies developed in this thesis aim at overcoming such a limitation by incorporating stock-related technical data to better contextualise past stock returns.

The contextual information provided to the Reinforcement Learning agent, to enable him to learn how to behave in a complex system and optimise his policies, was not just about the prices of each single action. In fact, to ensure that the agent could recognize the different market structures and have an overview of the context in which it was operating, we included in the training set information on prices' moving averages, historical volumes, trend momentum indicators, volatility-related information and a correlation matrix between the stocks considered.

In this thesis we have experimented the possibility of extending the decision metrics, such as the sharpe ratio or the stability, related to the portfolio composition and historical returns within the reward function of the RL method. In fact, by rewarding the agent for having obtained higher values in these metrics, it was possible to influence the learning signals and the development of trading strategies. The results of the experiments proved the effective functioning of this methodology.

The results achieved on real stock market data show that through the new, price-aware reward function implemented in this thesis, it was possible to improve the average results of all the algorithms and to guide the agent in the development of strategies to achieve specific results. In particular, by giving more weight in the reward function to the metrics of sharpe ratio, stability and gini index, it was possible to obtain more stable and at the same time profitable strategies, in line with the defined parameterization.

The different deep reinforcement learning algorithms have reacted positively to the changes implemented in this thesis, both in the case of portfolio allocation and in multi stock trading. In particular, the configurations in which the sharpe ratio metric was preferred have far outperformed compared to the FinRL baseline, reaching annual returns of 41% compared to the 30% achieved by the base model. On the other hand, in the cases where the configurations preferred the Stability and Gini Index metrics, the strategies were more stable and resilient to drawdowns. Even in the case of mixed configurations, where the agent was jointly awarded for more than one metric, the results were consistent with what was specified in the reward function, still obtaining better results than the baseline. The results show how the reinforcement learning model was able, despite the stochastic nature of the markets, to recognize the setups that would have allowed it to optimise cumulative rewards in the short and long term.

The future works will address the application of similar approaches to other markets, such as cryptocurrencies, and financial instruments. They will also investigate the interpretability of the models thus providing experts the reasons behind the generated predictions.

Bibliography

[1] Sharpe, W.F., 1964. Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, *19*(3), pp.425-442.

[2] Markowitz, H.M., 1968. *Portfolio selection*. Yale university press.

[3] Tsay, R.S., 2005. *Analysis of financial time series*. John wiley & sons.

[4] Wu, J., Xu, K., Chen, X., Li, S. and Zhao, J., 2022. Price graphs: Utilising the structural information of financial time series for stock prediction. *Information Sciences*, *588*, pp.405-424.

[5] Ma, S. and Li, P., 2021. *Predicting Daily Trading Volume via Various Hidden States* (No. 2107.07678).

[6] Zhao, L., Li, W., Bao, R., Harimoto, K. and Sun, X., 2021. *Long-term, Short-term and Sudden Event: Trading Volume Movement Prediction with Graph-based Multi-view Modeling* (No. 2108.11318).

[7] Rak, R., Drożdż, S., Kwapień, J. and Oświęcimka, P., 2013. Stock Returns Versus Trading Volume: Is the Correspondence More General?. *Acta Physica Polonica B*, *44*(10), p.2035.

[8] Morris, G.L., 2006. *Candlestick Charting Explained: Timeless Techniques for Trading stocks and Sutures*. McGraw Hill Professional.

[9] Zhao, R., Deng, Y., Dredze, M., Verma, A., Rosenberg, D. and Stent, A., 2019, May. Visual attention model for cross-sectional stock return prediction and end-to-end multimodal market representation learning. In *The Thirty-Second International Flairs Conference*.

[10] Turner, J.G. and Murphy, C.J., 2021. How Do Proteins Associate with Nanoscale Metal–Organic Framework Surfaces?. *Langmuir*, *37*(32), pp.9910-9919.

[11] Kong, A., Zhu, H. and Azencott, R., 2021. Predicting intraday jumps in stock prices using liquidity measures and technical indicators. *Journal of Forecasting*, *40*(3), pp.416-438.

[12] Bouktif, S. and Awad, M.A., 2015, November. Predicting stock market movement: An evolutionary approach. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)* (Vol. 1, pp. 159-167). IEEE.

[13] Kumar, A., Alsadoon, A., Prasad, P.W.C., Abdullah, S., Rashid, T.A., Pham, D.T.H. and Nguyen, T.Q.V., 2021. Generative adversarial network (GAN) and enhanced root mean square error (ERMSE): deep learning for stock price movement prediction. *Multimedia Tools and Applications*, pp.1-19.

[14] Singh, A. and Xu, D., 2016. Random matrix application to correlations amongst the volatility of assets. *Quantitative Finance*, *16*(1), pp.69-83.

[15] Tsay, R.S., 2005. *Analysis of financial time series*. John wiley & sons.

[16] Sharpe, W.F., 1966. Mutual fund performance. *The Journal of business*, *39*(1), pp.119-138.

[17] Sharpe, W.F., 1994. The Sharpe Ratio, the journal of Portfolio Management. *Stanfold University, Fall*.

[18] Young, T.W., 1991. Calmar ratio: A smoother tool. *Futures*, *20*(1), p.40.

[19] Bacon, C.R., 2021. *Practical risk-adjusted performance measurement*. John Wiley & Sons.

[20] Keating, C. and Shadwick, W.F., 2002. A universal performance measure. *Journal of performance measurement*, *6*(3), pp.59-84.

[21] Bernardo, A.E. and Ledoit, O., 2000. Gain, loss, and asset pricing. *Journal of political economy*, *108*(1), pp.144-172.

[22] Sortino, F.A. and Price, L.N., 1994. Performance measurement in a downside risk framework. *The Journal of Investing*, *3*(3), pp.59-64.

[23] Rollinger, T.N. and Hoffman, S.T., 2013. Sortino: a 'sharper'ratio. *Chicago, Illinois: Red Rock Capital*.

[24] Li, B. and Hoi, S.C., 2014. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, *46*(3), pp.1-36.

[25] Bellman, R., 1957. A Markovian decision process. *Journal of mathematics and mechanics*, pp.679-684.

[26] Chen, Scott, 2022. "*Active Management*", Investopedia.com.

[27] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285.

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

[29] Deisenroth, M.P., Neumann, G. and Peters, J., 2013. A survey on policy search for robotics. *Foundations and trends in Robotics*, *2*(1-2), pp.388-403.

[30] Dabérius, K., Granat, E. and Karlsson, P., 2019. Deep execution-value and policy based reinforcement learning for trading and beating market benchmarks. *Available at SSRN 3374766*.

[31] Shultz, S., 2009. Barron's Online (barrons. com). *Journal of Business & Finance Librarianship*, *15*(1), pp.25-30.

[32] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F. (2020). "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning".

[33] Ciresan, D.; Meier, U.; Schmidhuber, J. (2012). "Multi-column deep neural networks for image classification". *2012 IEEE Conference on Computer Vision and Pattern Recognition*. pp. 3642–3649.

[35] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffry (2012). "ImageNet Classification with Deep Convolutional Neural Networks"

[36] "Google's AlphaGo AI wins three-match series against the world's best Go player". *TechCrunch*. 25 May 2017. [2]Goodfellow, Bengio & Courville 2016, p. 200, "Furthermore, back-propagation is often misunderstood as being specific to multi-layer neural networks, but in principle it can compute derivatives of any function"

[37] Freund, Y.; Schapire, R. E. (1999). "Large margin classification using the perceptron algorithm" (PDF). *Machine Learning*. **37** (3): 277–296. doi:10.1023/A:1007662407062. S2CID 5885617.

[38]Deepmind's paper "Asynchronous Methods for Deep Reinforcement Learning" (Mnih et al, 2016)

[39]G. L. Morris. Candlestick Charting Explained: Timeless Techniques for Trading Stocksand Futures: Timeless Techniques for Trading stocks and Sutures. McGraw Hill Professional, 2006.

[40]Reizinger, Patrik; Szemenyei, Márton (2019-10-23). "Attention-based Curiosity-driven Exploration in Deep Reinforcement Learning". arXiv:1910.10840 [cs.LG].

[41]Törn, Aimo, and Antanas Žilinskas. Global optimization. Vol. 350. Berlin: Springer-

Verlag, 1989

Sitography

1. https://books.google.it/books?id=8oge2lAwJzsC&pg=PA8&lpg=PA8&dq=october+1980+commodities+magazine+lambert&source=bl&ots=yZ81lB8boE&sig=ACfU3U3EAjJFbRT5Oxq9ZNH2o2w0dczq5g&hl=en&ppis=_e&sa=X&redir_esc=y#v=onepage&q&f=false
2. https://www.investopedia.com/terms/m/macd.asp
3. https://proceedings.neurips.cc/paper/2017/file/facf9f743b083008a894eee7baa16469-Paper.pdf
4. https://livebook.manning.com/book/grokking-deep-reinforcement-learning/chapter-8/v-7/6
5. https://github.com/AI4Finance-Foundation/FinRL/blob/master/tutorials/tutorial_env_multistock_cashpenalty.ipynb
6. https://repository.tudelft.nl/islandora/object/uuid:0fac495f-f87a-4a61-a80f-5f901323379a/datastream/OBJ/download
7. https://github.com/AI4Finance-Foundation/FinRL/blob/master/FinRL_portfolio_allocation_NeurIPS_2020.ipynb
8. https://osf.io/gtvxp/download
9. https://arxiv.org/pdf/2011.09607.pdf
10. https://towardsdatascience.com/finrl-for-quantitative-finance-tutorial-for-multiple-stock-trading-7b00763b7530
11. https://towardsdatascience.com/measuring-financial-turbulence-and-systemic-risk-9d9688f6eec1#e8a1
12. https://www.top1000funds.com/wp-content/uploads/2010/11/FAJskulls.pdf)
13. https://arxiv.org/abs/1707.06347
14. https://en.wikipedia.org/wiki/Backpropagation#cite_note-FOOTNOTEGoodfellowBengioCourville2016[httpswwwdeeplearningbookorgcontentsmlphtmlpf33_214]-3
15. https://theaisummer.com/Actor_critics/
16. https://spinningup.openai.com/en/latest/algorithms/ddpg.html#deep-deterministic-policy-gradient
17. https://spinningup.openai.com/en/latest/algorithms/td3.html
18. https://spinningup.openai.com/en/latest/algorithms/sac.html

20. https://www.investopedia.com/terms/s/sharperatio.asp
21. https://www.investopedia.com/terms/s/sortinoratio.asp
22. https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/calmar-ratio/
23. https://www.wallstreetmojo.com/omega-ratio/
24. https://rc.library.uta.edu/uta-ir/bitstream/handle/10106/28108/KANWAR-THESIS-2019.pdf?sequence=1&isAllowed=y
25. https://en.wikipedia.org/wiki/Reinforcement_learning
26. https://amslaurea.unibo.it/16718/1/thesis.pdf
27. https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7
28. http://eprints-phd.biblio.unitn.it/2961/1/These_final02.pdf
29. https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/
30. https://towardsdatascience.com/finrl-for-quantitative-finance-tutorial-for-multiple-stock-trading-7b00763b7530
31. https://arxiv.org/pdf/2011.09607.pdf
32. https://arxiv.org/abs/1707.06347
33. https://www.analyticsvidhya.com/blog/2018/11/reinforcement-learning-introduction-monte-carlo-learning-openai-gym/
34. https://ai.stackexchange.com/questions/4456/whats-the-difference-between-model-free-and-model-based-reinforcement-learning
35. https://arxiv.org/abs/2103.04909
36. https://spinningup.openai.com/en/latest/algorithms/sac.html
37. https://arxiv.org/pdf/2006.16712.pdf
38. https://spinningup.openai.com/en/latest/algorithms/ddpg.html#deep-deterministic-policy-gradient

.