Abstract

The primary goal of this master's thesis is to present and analyze deep reinforcement learning techniques in predicting financial market moves. The thesis presents a trading bot that uses Q-Learning techniques to predict future stock prices, outperform markets, and automate trading. Different stock sectors (tech stocks, crypto) were analyzed using the trading bot to show how deep reinforcement learning techniques behave in different market types. The paper also pre-analyzes how these trading bots behave during unexpected events that cannot be predicted using only market data (e.g., the 2008 real-estate crash and the Dot-com bubble). In today's world, financial institutions widely use advanced machine learning or other AI techniques to improve predictability in the stock markets. This area in the financial world has been rapidly growing over the past years. Deep Reinforcement Learning can be defined as goal-directed learning, where an agent achieves long-term goals at a time-based scale. The trial-and-error learning approach optimizes the agent's behavior, so each action has consequences and rewards. It makes DRL methods perfect candidates for adoption in trading agents (buying a stock at a time can cause a reward when the stock price goes up or the opposite if down).

Keywords

Computer Science, Artificial Intelligence, Machine Learning, Econometrics, Deep Learning, Reinforcement Learning, Trading Algorithms, Financial Technology (FinTech).

1. Introduction

This master's thesis evaluates the performance of automated stock trading techniques with Reinforcement Learning algorithms. The thesis focuses on Q-Learning algorithms and investigates the potential usefulness of these techniques in a natural stock market environment. The stock market is a very non-deterministic place with many different types of participants. The main driving force of the stock market is often the aggregate psychology of individual market participants which manifests itself in the form of fear during stock crashes and greed when the stock reaches All-Time High (ATH) prices. In theory, the main driving force of the company's stock price is financial fundamentals like the stock's Price to Earnings ratio (P/E ratio), company debt, future company forecasts, inflation, unemployment, and economic growth. History shows that this is not always the case. Real-life models used in, e.g., automation can be described using mathematical formulas (state-space model) and controlled using various controllers (e.g., PID). Even if the object cannot be fully described using mathematical formulas, there are ways to approximate some parts of the model (linearization, parameter estimation). Unfortunately, for the stock market, there is no mathematical equation that can predict future moves. However, sometimes some techniques can estimate future market moves with a certain probability. One technique that stock traders widely use is Technical Analysis, which tries to forecast the direction of stock price based on past market price and volume (the total number of shares that are traded during the trading day or specified fixed time).

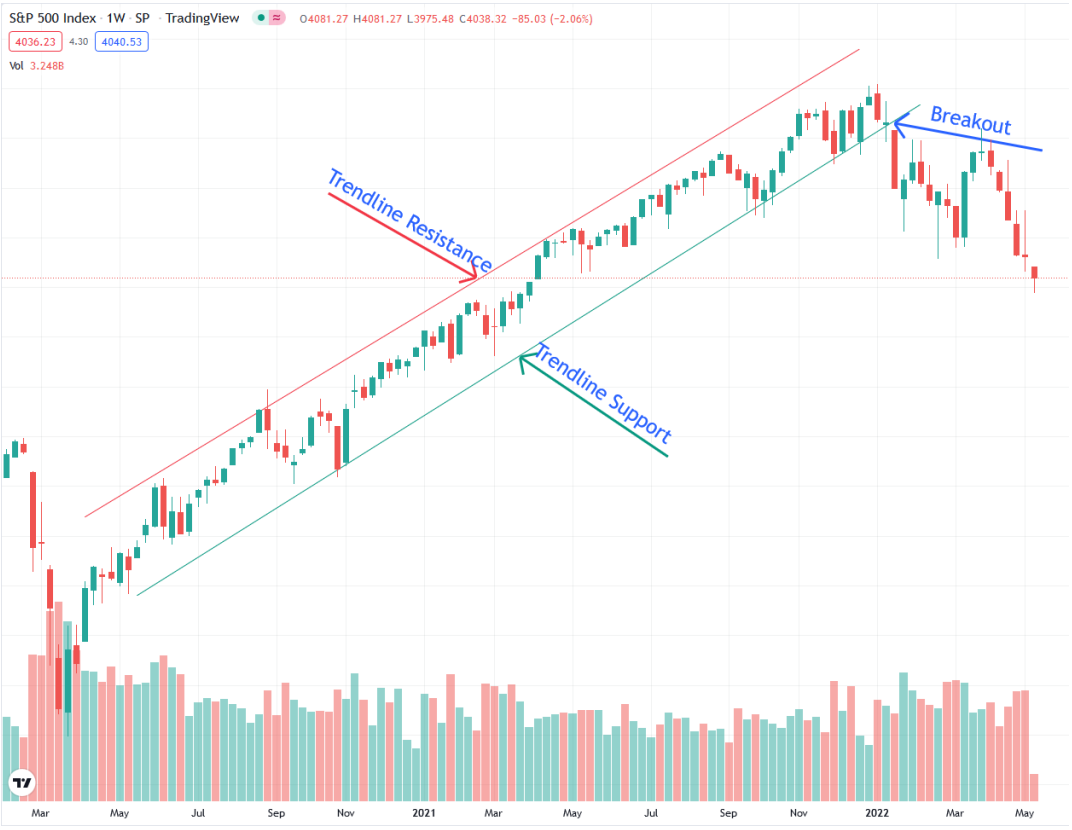


Fig. 1. S&P 500 index price during the last two years. The chart shows examples of Technical Analysis usage. Chart from https://www.tradingview.com.

In short, Deep Reinforcement Learning is a subfield of Machine Learning techniques that combines Deep Learning (DL) and Reinforcement Learning (RL). The neural network learning curve is the difference between classical Machine Learning (ML) and RL algorithms. The Neural Network (NN) is trained before usage in classical ML methods. During the execution, the NN is trained. In RL methods, the NN continuously learns by trial and error. Depending on the implemented algorithm, the learning curve can be more or less sophisticated. In this paper, the Q-Learning algorithms, which are a subfield of RL, will be investigated. In short, these algorithms make decisions based on the input (e.g., stock price). The decisions that are made after that are evaluated for learning purposes. In this paper, Q-Learning algorithms will be used to determine what position on a particular day should a day-trader (a person who executes short or long trades to profit from intraday market price actions) take.

1.1 Research Objectives

Main purpose of this paper was to answer the central question: Can Deep Reinforcement Learning (especially Q-Learning algorithms) help predict the stock market? This paper will describe and evaluate several RL algorithms that belong to the Deep Q-learning (DQN) family (Deep Q-Learning, Double Deep Q-Learning, and Deep Q-Learning with noisy NN). DQN algorithms showed great performance results in several Atari games, where in some cases, RL agents achieved scores better than humans [5]. Trading can also be defined as a game where the trader tries to outperform the market. RL methods use pixels (in Atari games) as the input necessary to determine which decision to take. Instead of pixels, the RL can take market signals (volume, stock price, etc.) as the input needed to determine which action to take (e.g., long, short position).

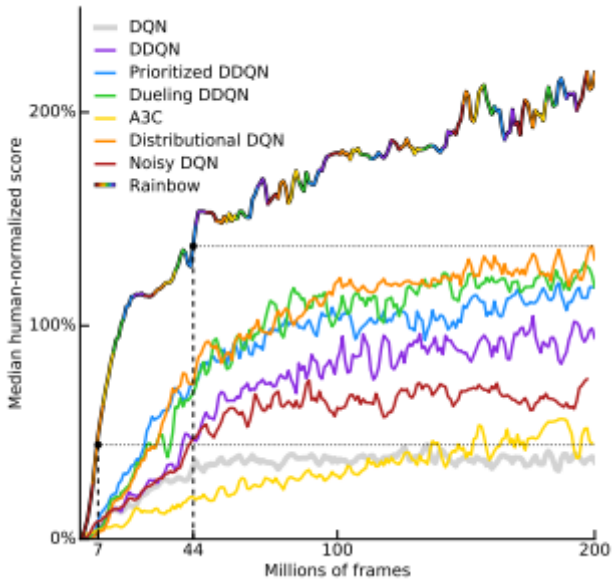


Fig. 2. Median human-normalized performance of DQN algorithms across 57 Atari games as a function of time. Author: Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver, DeepMind [5].

This paper will determine whether the stock market is too unpredictable for these algorithms or whether the DQN techniques can achieve good performance and market returns successfully. Using average market returns, this paper will determine the performance of agents. The study uses historical stock data of several assets (stock, crypto).

1.2 Thesis Outline

The thesis is organized into four main chapters. Chapter 1 introduces the topic of the usage of modern IT techniques in the stock market. Chapter 2 presents explain basic market fundamentals, indicators, and terminology. Chapter 3 describes overall reinforcement learning, illustrations of how this technique is used, and its various extensions. This chapter gets deeper into the learning process and algorithms. Chapter 4 presents the experimental part of this thesis, explains the threading bot software, and shows the results and their implications. At the end of the thesis, the conclusion offers a distilled synthesis of the research and its implications.

2. Usage of modern IT techniques in market analysis

Modern computers and improvements in computation capabilities also advanced trading. Digitalization of the stock market began with the introduction of Electronics Communication Networks (ECNs). The first ECN, Instinet, was founded in 1969, and the main idea was to make electronic transactions and trades available for small brokers and individuals. ECNs are alternative trading systems (ATS) that try to match buyers and sellers in the stock market. These systems allowed people worldwide to participate in the global stock market. Another ATS is Darks Pools, similar to ECNs but not publicly available. The participants of the dark pools can place buy/sell orders without publicly revealing their orders. It helps the big players (e.g., institutions and hedge funds) with large orders to avoid significant price fluctuations. Suppose these huge orders were publicly available, for example, selling 1 million shares of specific stock by the pension fund. In that case, this could significantly affect the stock's price (even crash it). There are also many critics of Dark Pools. These spaces give a significant advantage over public exchanges. Making trades away from the public eye makes Dark Pools more opaque than public exchanges.

Today, many hedge funds use various algorithms that help them make decisions, predict market moves or manage assets. Even small investors use modern techniques to manage their portfolios, calculate potential returns, and measure risks.

2.1 Non-ML Techniques

2.1.1 High-Frequency Trading

The digitalization of the stock market created an opportunity to take advantage of and exploit inefficiencies in the market players, who have faster access and lower latency in the stock exchange. High-Frequency Trading (HFT) is the automated method that trades with extremely low latency and short-lived holding positions. In short, HFT tries to take advantage of slower traders. One of the strategies tries to profit from one asset that is traded in different places.

2.1.2 Monte Carlo simulations

The Monte Carlo (MC) simulations can be used in cases where the outcome cannot be easily predicted due to random variables. This technique can help predict the probability of various outcomes and help manage risk and uncertainty in the stock market. It is also used in various fields like physics and business. Overall The Monte Carlo methods are the statistical methods that solve many mathematical or statistical problems. Simulations using MC methods are based on the principle of the Law of Large Numbers and the Central Limit Theorem [12].

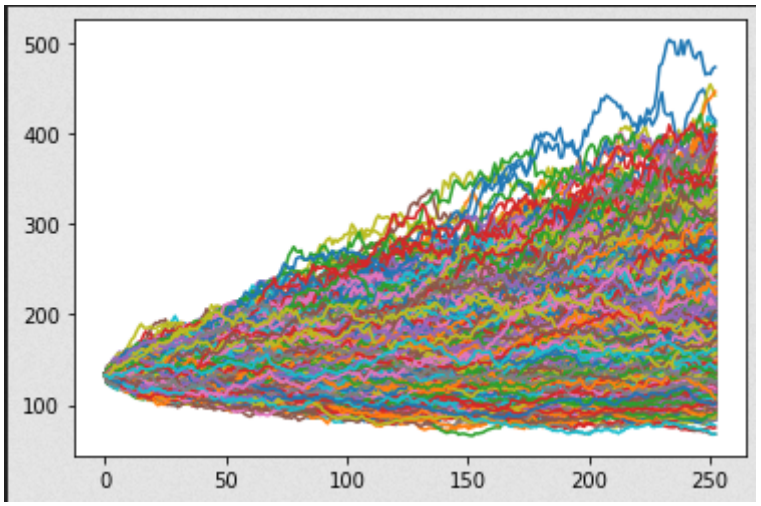


Fig. 3. Monte Carlo simulation with multiple generated series of trading returns. Different outcomes (trading returns) will form a normal distribution.

2.2 ML techniques

2.2.1 Machine Learning

Machine Learning (ML) is an automated learning process that identifies patterns from the data and can be used in the future to solve similar problems. The time and learning rate depends on the amount of training dataset and impacts the algorithm's future performance. One of the significant differences between classical algorithms and machine learning is the learning process. Classical algorithms have defined some constraints during the development cycle (e.g., conditions written by the developer). If the algorithm is tailored for a well-known problem, there is no problem for classical algorithms. Things change when the problem changes behavior in a way that requires changes in the algorithm (e.g., a new condition in code). In machine learning, the learning process ties to identifying patterns over time. In some cases, if the problem changes behavior, the ML algorithm can adjust to the new behavior by re-learning. This makes ML algorithms more flexible in many applications.

2.2.1.1 Supervised and unsupervised learning

Supervised learning is a widespread ML technique. The word “supervised“ implies that the algorithm uses some kind of guide during the learning process. This guide tells the ML algorithm the correct answer during the learning process. If the supervised ML algorithm makes an error, it adjusts inside weights based on the correct answer provided by the guide. Correct answers are provided through datasets prepared to train the ML model.

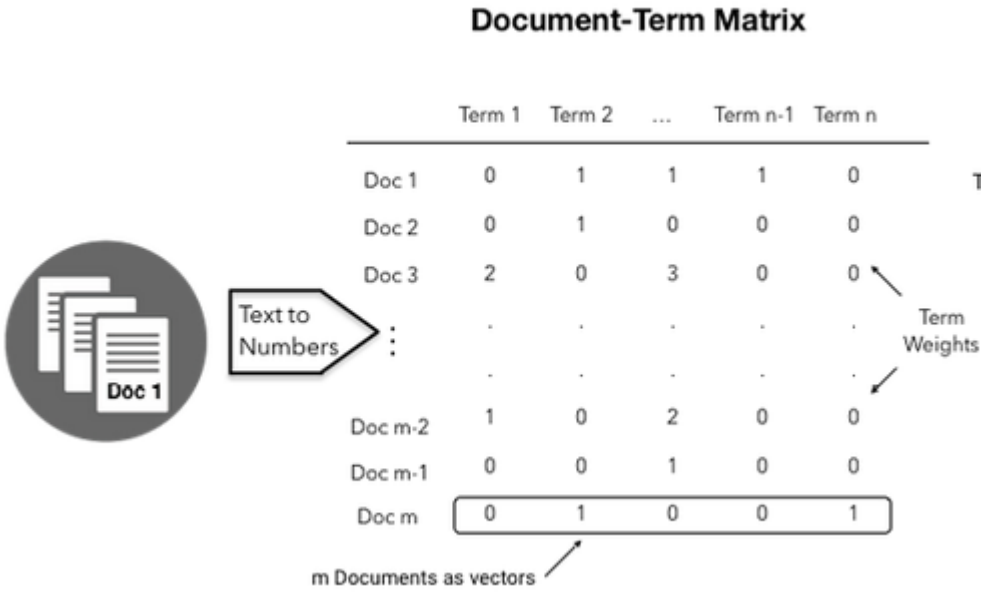
Unsupervised learning is more sophisticated because the ML algorithm can only observe and draw conclusions without guidance. The unsupervised algorithms try to identify the structure or pattern of the input datasets and classify them into a specific category (e.g., clusters).

2.2.1.2 Reinforcement Learning

Reinforcement Learning (RL) is neither supervised nor unsupervised learning. In RL, an agent interacts with the environment on their own. Actions made by the agent are interactions that result in rewards. The agent interacts with an environment in discrete steps and gets rewards as feedback (either positive or negative). RL learns using the trial-and-error method and is between supervised and unsupervised methods. For example, reward feedback has a form of supervised learning. However, the learning dataset does not need to be predefined in RL, and the agent makes their own decisions (like unsupervised learning).

2.3.2 NLP - text processing

Modern ML techniques allow analyzing and recognizing patterns in text and documents. The main idea behind the NLP techniques is to convert human-speech text to numbers readable by ML algorithms. In order to do that, the text must be converted into vectorized tokens. The token can be a single word or a couple of words. For example, the token “machine learning” is more meaningful and can help more in text analysis than two distinct tokens like “Machine” and “Learning”. There are various techniques to fetch a token list from documents automatically. One of the easiest ones is to count the occurrence of neighboring tokens (e.g., “Machine” and “Learning” tokens occur quite often (more than specific frequency) next to each other in science papers and can be merged into one token, “machine learning”). In other words, tokens are isolated semantic units. To combine tokens and specific documents, we use a bag-of-words model that uses token frequencies in vector documents. These bag-of-words models from various documents can be used to analyze similarities between documents.



Fix. 4. Document-term matrix. Each row represents a document model. Each column represents a token. Each cell represents a frequency of tokens in a specific document. The document-term matrix can be used to analyze the similarity of different documents. Author: Stefan Jansen [1].

2.3.2.1 Analyzing SEC Filings

Damir Cavar and Matthew Josefy [13] proposed Natural Language Processing (NLP) techniques to analyze fillings (e.g., 10-K) to extract meaningful knowledge from SEC filings. This method helps us learn how companies and their executives are linked to one another and how various risks are identified and handled over long periods. SEC fillings tend to be long and written in a very formalized language and sometimes are difficult to understand. Moreover, companies try to hide negative factors there. These methods can help traders understand the current company's condition.

2.3.3 Convolutional Neural Network (CNN)

Convolution Neural Networks (CNN) have the name from the operation called convolution, where one element of the matrix is summed with the neighboring elements (with corresponding weights).

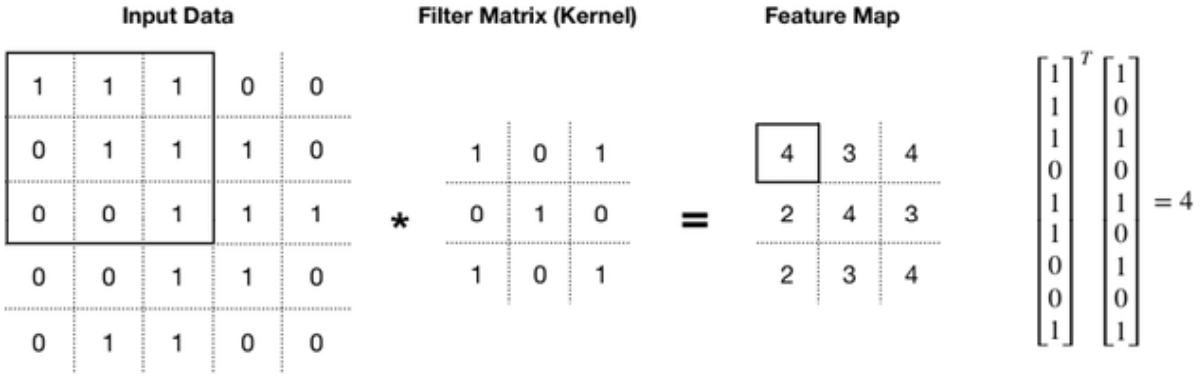


Fig. 5. Operation of matrix convolution where all neighboring elements and single matrix element are multiplied by weights (defined by Filter Matrix) and summed up. Input data can represent input images, whereas numbers can represent color intensity (e.g., 1 - high intensity). Filter Matrix (Kernel) defines weights that should be applied to each element before summation. Author: Stefan Jansen [1].

2.3.3.1 Analyzing satellite images

Analyzing satellite images can be helpful in approximating crop quality and quantity in a specific area (e.g., the Europe market) and, therefore, based on that, predicting future prices. These algorithms can work on an extensive data set to process large areas.

Fix.6. Satellite data and machine learning are used to classify types of lands and, therefore, can help with future prices of some crops or with the recognition of soil types or potential places with specific minerals (mining industry). Author: Stefan Jansen [1].

2.3.4 Generative Adversarial Networks (GANs)

In 2014 by Godfellow et al. [14], Generative Adversarial Networks (GANs) found applications in many domains a few years later. GANs consist of two networks: generator and discriminator. The generator network tries to produce images that the discriminator network cannot distinguish from real or fake. Two networks compete and try to outsmart each other. An image recognition task is an excellent example of how GANs networks work. The generator network either gives an accurate picture from the dataset or tries to generate a fake photo similar to the real one. The task of the discriminator network is to distinguish between counterfeit and genuine pictures. During this competition, the generator network learns how to generate better fake pictures, and the discriminator network learns how to detect phony images. The GANs learning process is a zero-sum game between two neural networks.

As mentioned before, GANs methods contain two neural networks. The discriminator network belongs to discriminative models algorithms that focus on learning the probability of a specific outcome for a given input data:

$$P(Y|X = x)$$

On the other hand, the generator network belongs to generative models algorithms, which are statistical models of joint probability distribution:

$$P(Y|X)$$

Today, GANs are one of the most exciting and innovative areas of machine learning due to various applications, that include:

- image recognition - recognition of gender, hair color of humans, animals
- video generation
- human pose identification

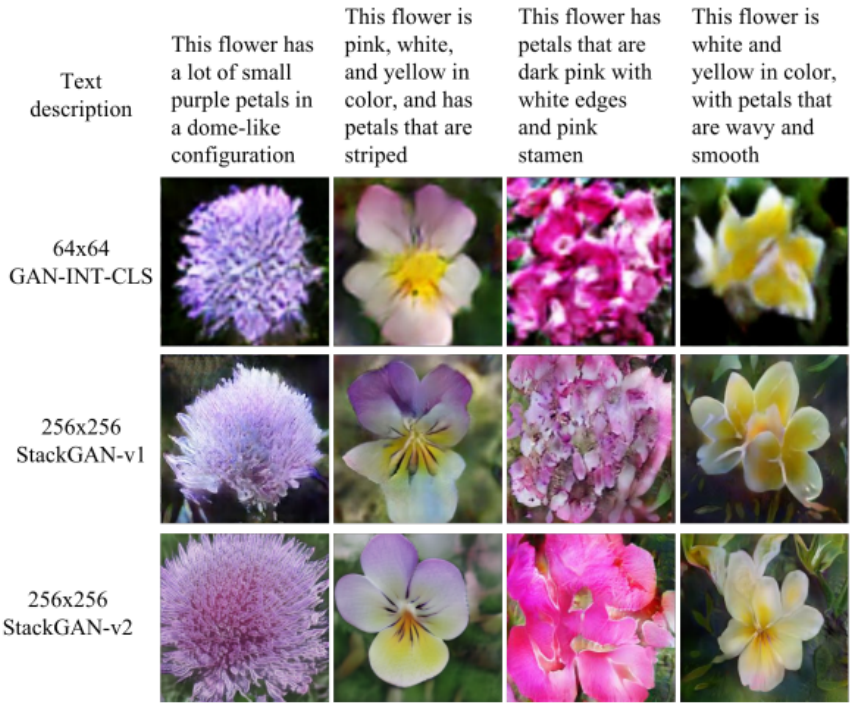


Fig. 7. Text-to-photo synthesis using StackGANs and GAN-INT-CLS networks. Top row - text description that was used to generate images (input to networks). Second/third/fourth row - different results of synthesis using different GANs networks implementation (StackGANs, GAN-INT-CLS) [15].

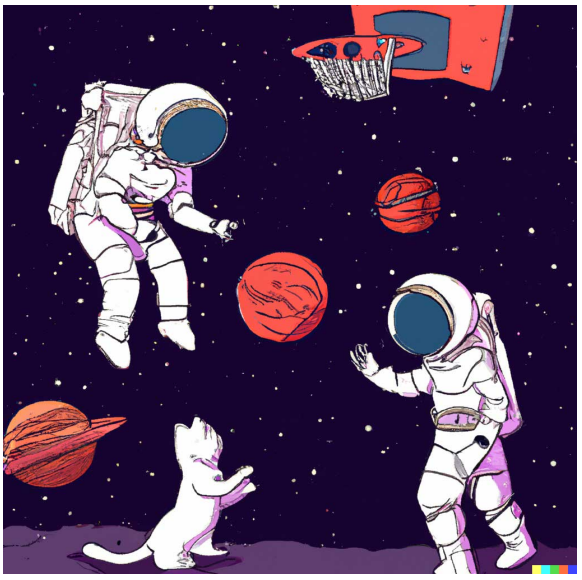


Fig. 8. Text-to-image synthesis using DALL-E 2. The text description is “An astronaut playing basketball with cats in space as a children's book illustration”. The image was generated from the text using DALL-E 2 AI system that uses i.a. GANs. Author: DALL·E 2 AI system.

2.3.4.1 Synthetic time series

Jinsung Yoon, Daniel Jarrett, Mihaela van der Schaar proposed a framework that can generate quite realistic time-series using a combination of supervised learning and GANs [14]. Time-series generative adversarial network (TimeGAN) learns how to generate synthetic financial data using time series (e.g., historical stock price). The generative network algorithm in GANs can generate fake time series or use historical time series to fool the discriminator network. During the learning process, the generative network can continuously generate better synthetic time-series data.

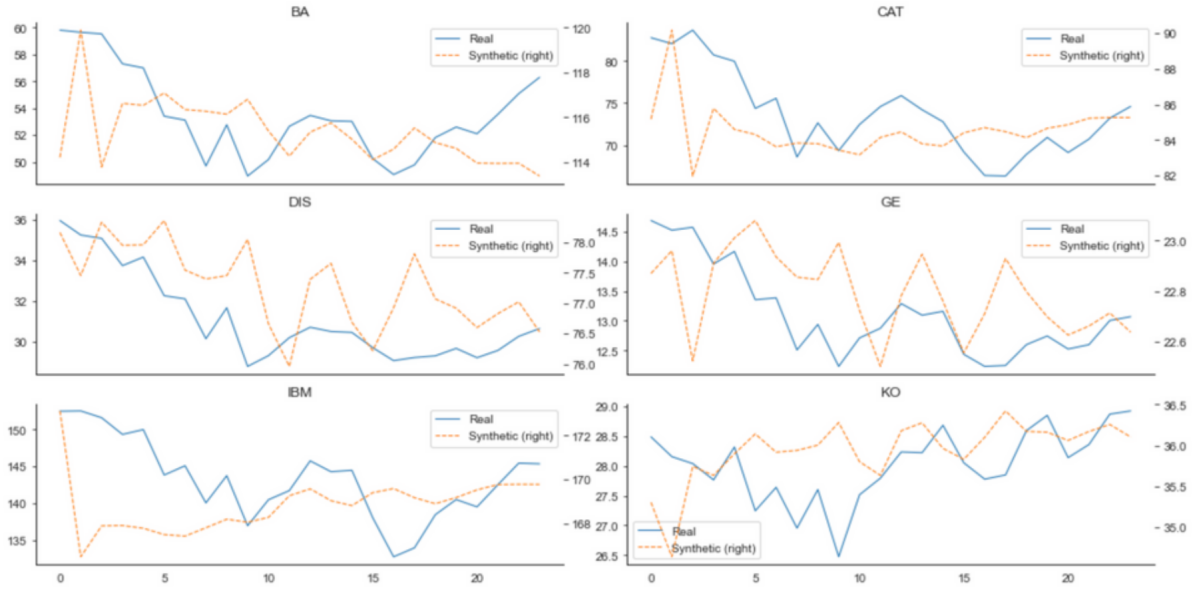


Fig. 9. Comparison of synthetic stock price time-series (generated by TimeGAN) and real-time series. Author: Stefan Jansen [1].

2.3.5 Deep Reinforcement Learning

Reinforcement learning due to flexibility can be used in various areas. The main idea behind RL is to optimize the process where an agent makes some actions and receives feedback from the environment as a reward. The main area where RL can be utilized:

- Machine Learning: RL is a subfield of Machine Learning.
- Engineering: Optimization of the process that has a specific number of actions (e.g., optimized robot movement)
- Neuroscience: RL works similarly to the human brain
- Economics: Maximization of return of investment in stock market

- Game: AlphaGo computer program that defeated a professional human Go player uses RL to improve learning and decision making.

3. Market fundamentals

3.1 Useful data to analyze asset prices

There are several methods to analyze asset prices in the stock market.

- Historical price - the price of an asset from previous days/months/years. Using statistical and technical analysis, these data can be analyzed to identify patterns and trends
- Volume data - trading volumes of the asset give information about participation and the liquidity of the asset.
- News articles and press releases - can provide insights into market sentiment and potential trends.
- Economic data - this includes indicators like GDP, inflation, unemployment, and interest rates. These indicators can affect the whole market, and asset prices can drop, even if it has strong fundamentals.
- Sentiment - that includes social media and forums sentiments. It can provide insights into market sentiment and potential trends
- Fundamental data - this includes e. g. financial statements, SEC filings, and other fundamental data like Price/Earning ratio. These indicators can provide insights into long-term asset trends.

By analyzing these types of data, the trader can gain more knowledge and understanding about the factors that drive specific assets in the stock market.

3.2 Trading Indicators

Trading indicators help traders find patterns in the asset price that can show price trends and predict future prices. Due to the uncertainty of the stock market and massive competition, there is no Holy Grail of trading indicators, and traders use multiple indicators to recognize future trends. Technical Indicators can predict the probability of specific patterns and guide the trader's decisions. In this thesis, Trading Indicators will be the inputs to the Neural Network. Based on the values of these indicators and, therefore, the outputs of the neural network, the Trading Bot will make decisions.

3.2.1 Moving Average (MA)

This indicator captures the average change of stock price over time. It is one of the most fundamental analysis tools in technical analysis. Simple Moving Average can be described as:

$$SMA = \frac{\sum_{i=1}^{n} close\_price_i}{n}$$

where:

$close\_price_i$ - asset close price on i-th day

n - number of days

SMA can indicate an overall stock trend and average changes. SMA for a more extended period (e.g., 200 days) will have a more significant price change lag than SMA for a shorter one (e.g., 20 days). A rising 200 SMA indicator can indicate that the long-term trend is positive.

Fig. 10. OHLC chart with APPL stock price and comparison of 20 SMA (red line), 50 SMA (green line), and 200 SMA (blue line) indicators. 20 SMA responds more quickly to stock price changes (compared to 50 and 200 SMA). The SMA indicator is a primary trend indicator in technical analysis. Chart from https://www.tradingview.com.

3.2.2 Exponential Moving Average (EMA)

Exponential Moving Average (EMA) is a type of moving average that focuses more on recent prices. It can also be described as an exponentially weighted moving average. It helps traders identify buy and sell signals. EMA can be described as

$$EMA_{today} = (value_{today} \bullet \frac{smoothing}{1+ days}) + (EMA_{yesterday} \bullet (1 - \frac{smoothing}{1+ days}))$$

where:

$value_{today}$ - today asset close price

$smoothing$ - smoothing factor, usually - 2

$days$ - EMA length (e.g., 10/50/200 days)

3.2.3 Momentum Indicators

3.2.3.1 Aroon Indicator (AI)

Aroon Indicator (AI) is used to identify price trend changes and the strength of the trend. This indicator has two types:

- Aaron up - measures the uptrend strength
- Aaron down - measures the downtrend strength

Crossover of both types indicates trend change. The formula is:

$$Aroon\ Up = \left[\frac{period\ specified - number\ of\ periods\ since\ the\ highest\ high}{period\ specified} \times 100\right]$$

$$Aroon\ Down = \left[\frac{period\ specified - number\ of\ periods\ since\ the\ lowest\ low}{period\ specified} \times 100\right]$$

where:

$number\ of\ periods\ since\ the\ highest\ high$ – length of days since the asset has reached a recent high

$number\ of\ periods\ since\ the\ lowest\ low$ – length of days since the asset has reached a recent low

$period\ specified$ - usually 25 days

Fig. 11. OHLC chart (top) and Aroon indicators (bottom) of S&P 500 index. The yellow line of the bottom chart represents the Aroon Up indicator which measures uptrend strength. The blue line of the bottom chart represents the Aroon Down indicator which measures downtrend strength. Crossing these lines indicates trend change. Chart from https://www.tradingview.com.

3.2.3.2 Parabolic Stop and Reverse (PSAR) Indicator

Parabolic Stop and Reverse (PSAR) Indicator can be a good trend-following indicator and detect potential reversal. PSAR is represented in dots either above or below price values. When the dots are below the asset price, there is an uptrend. Otherwise, when the dots are above the price. To calculate the PSAR value, we use two formulas:

$$PSAR_{rising} = PSAR_{previous} + \left[AF_{previous}\left(EP_{previous} - PSAR_{previous}\right)\right]$$

$$PSAR_{falling} = PSAR_{previous} - \left[AF_{previous}\left(PSAR_{previous} - EP_{previous}\right)\right]$$

where

$AF_{previous}$- Acceleration Factor which begins with a value equal to 0.02 and increment by 0.02 (up to 0.2) whenever extreme makes a new low (for $PSAR_{falling}$) or high (for $PSAR_{rising}$)

$EP_{previous}$- Extreme Point the lowest low (for $PSAR_{falling}$) ort highest high (for $PSAR_{rising}$)

To analyze charts and asset price, the trader should use only one type of PSAR indicator, either $PSAR_{rising}$ or $PSAR_{falling}$. In order to determine which one to use, there are two conditions. If the initial PSAR indicator is $PSAR_{rising}$ and the asset price has closed below, $PSAR_{rising}$ then the trend is now down, and trader should use $PSAR_{falling}$ the formula. If the price will close above $PSAR_{falling}$ the trend is up, and trader should use $PSAR_{rising}$ formula. The PSAR indicator, specifically trend change ($PSAR_{rising} \Rightarrow PSAR_{falling}$) or ($PSAR_{falling} \Rightarrow PSAR_{rising}$) can determine buy and sell signals, respectively.

3.2.3.3 Relative Strength Index (RSI)

Momentum indicator RSI tells how much the stock is overbought or oversold. This information can be helpful in predicting if the stock is more likely to rise or fall. The below equations represent Relative Strength (RS) and Relative Strength Index (RSI) formulas.

$$RS = \frac{trading\_days\_with\_rising\_prices}{all\_trading\_days} \div \frac{trading\_days\_with\_falling\_prices}{all\_trading\_days}$$

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

where:

$trading\_days\_with\_rising\_prices$ - a number of days where the stock's close price was higher than open price (trading day closed on green).

$trading\_days\_with\_falling\_prices$ - a number of days where the stock's close price was lower than open price (trading day closed on red).

$all\_trading\_days$ - a sum of $trading\_days\_with\_rising\_prices$ and $trading\_days\_with\_falling\_prices$. Usually, it is 14 days.

RSI above 70 means the stock is overbought, and the upward trend is about to change. In other words, it tells us that this is a good time to sell. RSI below 30 indicates that the stock is oversold, and the downward price trend is about to change. This RSI value indicates a good place for a long position.

3.2.3.4 Moving Average Convergence Divergence (MACD)

Moving Average Convergence Divergence (MACD) is an indicator that helps identify trend momentum and is based on the two moving average indicators. MACD can be described as:

$$MACD = EMA_{23} - EMA_{50}$$

where:

EMA_{23}, EMA_{50} - 23 and 50-period exponential moving averages

When the value of MACD is below zero, and it indicates a bearish signal. Above zero - bullish. Often when MACD crosses the zero value, there is a signal to buy (from negative to positive value) and sell (from positive to negative value).

3.2.3.5 Stochastic Oscillator

Stochastic Oscillator is a momentum indicator that contains two variables - the fast oscillating %K and the moving average of %K, called %D. Like the RSI indicator, both variables are in the range from 0 to 100. The formula of both variables is:

$$\%K = 100 \times \frac{close\_price - lowest\_low\_price}{highest\_high\_price - lowest\_low\_price}$$

$$\%D = SMA_{\%K}$$

where:

$close\_price$ - latest close price

$lowest\_low\_price$ - lowest low from the last X period (usually 14 days)

$highest\_high\_price$ - highest high from the last X period (usually 14 days)

$SMA_{\%K}$- Simple Moving Average of $\%K$ variable (usually 3-period simple moving average). See sec. 3.4.2.

3.2.3.6 Schaff Trend Cycle (STC)

Schaff Trend Cycle (STC) is an indicator developed by Doug Schaff in 1999 that helps investors identify buy and sell signals. STC can show the trend cycle. The STC can be described as:

$$STC = 100 \times \frac{MACD - \%K(MACD)}{\%D(MACD) - \%K(MACD)}$$

where:

$MACD = EMA_{23} - EMA_{50}$ - Moving Average Convergence Divergence (subtraction of 23-period exponential moving average and 50-period moving average)

$\%K(MACD)$, $\%D(MACD)$ - 10-period Stochastic from the MACD

3.2.3.7 Ultimate Oscillator (ULTOSC)

Ultimate Oscillator was developed by Larry Williams, and it measures price momentum across different timeframes. The indicator can generate “buy” or “sell” signals. Usually, ULTOSC takes 7, 14, and 28-day periods. The 7-day period (shortest) has the biggest weight in the calculation. ULTOSC is similar to RSI indicator (sec. 3.4.3.4), meaning that the value ranges from 0 to 100. Like RSI, values below 30 mean the stock is oversold, and values above 70 mean the stock is overbought. In order to calculate ULTOSC averages for specific periods, needs to be calculated:

$$AVG_X = \frac{\sum_{p=1}^{X} BP}{\sum_{p=1}^{X} TR}$$

where:

$BP = close\_price - min(low\_price - prior\_price)$ - Buying Pressure

TR - True Rate, see sec. 3.4.4.1

With calculated AVG_7,$AVG_{14}$and AVG_{28} $ULTOSC$ can be described as:

$$ULTOSC = \left[\frac{\left(AVG_7 \times 4\right) + \left(AVG_{14} \times 2\right) + \left(AVG_{28} \times 1\right)}{4+2+1}\right] \times 100$$

3.2.4 Volatility Indicators

3.2.4.1 Average True Range (ATR)

Average True Range (ATR) is an indicator that shows the average price variation in a given period. ATR is not an indicator that shows the price direction. Instead, it tells the investor how volatile the specific asset is.

Fig. 12. OHLC chart (top) and Average True Range chart (bottom) of GOOGL (NASDAQ) stock. The chart period from May to August shows increased volatility (pump and dump pattern). Therefore, the ATR in this range is higher. The chart period from August to October shows a continuous price drop. Therefore, the ATR in this range is lower. Chart from https://www.tradingview.com.

To calculate the ATR, we need to calculate the first True Rate for a specific trading day:

$$True\ Rate\ =\ max((high\ -\ low),\ abs(high\ -\ close_{previous}),\ abs(low\ -\ close_{previous}))$$

To calculate Average True Rate (ATR) we need to take the average value of actual rates for a specific period (usually 14 days).

$$Average\ True\ Rate\ = \frac{1}{n}\sum_{i=1}^{n} True\ Rate_i$$

4. Reinforcement Learning

Deep Reinforcement Learning (DRL) is a subfield of Machine learning that is trying to solve problems using automatic learning of optimized decisions over time.

Each DRL contains two main elements

- Agent, which performs actions.
- Environment with which the agent interacts.

An agent has its state and can perform actions to change it. Each action has either positive or negative outcomes. These outcomes define how agents should behave. The agent's main goal is to maximize positive outcomes (rewards) across an episode. Episode defines all subsequent states of the agent that happened (from the initial to the last state).

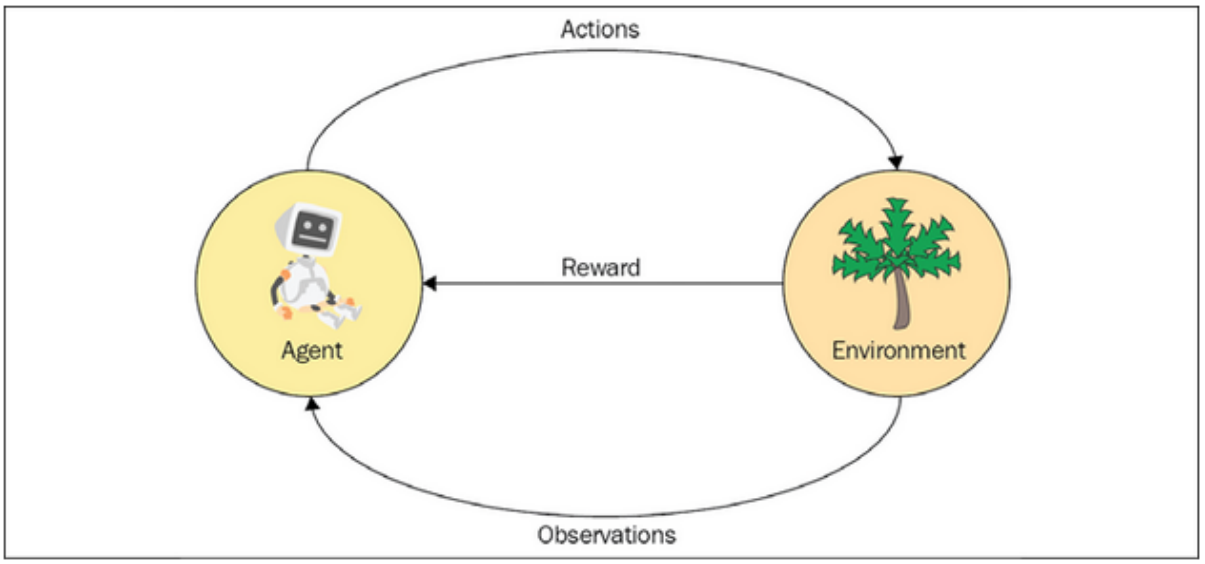


Fig. 13. How an agent interacts with the environment. Author: KC Tung [4].

4.1 Deep Q-Learning

4.1.1 Exploration and exploitation

The agent has to make decisions based on previous actions and needs to explore new ways. Finding the balance between these two approaches can cause problems, and no generic answer exists. This problem is called the exploration/exploitation dilemma. If an agent does not learn and only exploits

already learned paths, he cannot improve. On the other hand, if the agent explores new ideas too much, it decreases the reward (agents do not use the knowledge).

4.1.2 Elements of Deep Q-Learning

4.1.2.1 Agent

An agent is an object that interacts with the environment. The agent can perform specific actions, make observations and receive feedback from the environment as a reward. In the RL practical scenario, the agent can be:

- Lunar Lander: The software controls the spacecraft's thrusters to land the lander safely. The combination of landing time and damages can represent a reward.
- Mario Game: The software controls the game's character. A combination of collected gold and saved lives can represent the reward. The agent decides how to move the character (left, up, right).

4.1.2.2 Environment

The environment is an area which an agent operates and interacts with. The agent performs specific actions in the environment. It causes feedback from the environment in the form of rewards and observations.

4.1.2.3 Policy

The policy restricts the agent and sets some rules that control the agent. Using the previous example of the Mario Game policy restricts allowed actions that agents can perform (agents cannot move down if there is an obstacle).

4.1.2.4 Actions

Action is the most crucial interaction between the agent and the environment. The right actions define how well the problem is solved. The number of actions that an agent can perform is limited (e.g.,

Lunar Lander cannot go left). A set of rules restricts actions. DRL algorithm actions can have either discrete or continuous values. Discrete actions have a finite number of things that agents can do. It can be a finite number of moves that a pawn can do in a chess game. Continuous actions have an attached value usually in the range (e.g., 55% engine thrust).

4.1.2.5 Reward

Deep Reinforcement Learning (DFL) defines reward as a scalar number value an agent obtains from the environment after performing a specific action. It can be either positive or negative. The main idea of the reward is to define how well an agent behaves. The reward is essential because it tells if an agent should improve their behavior or if he goes the right way.

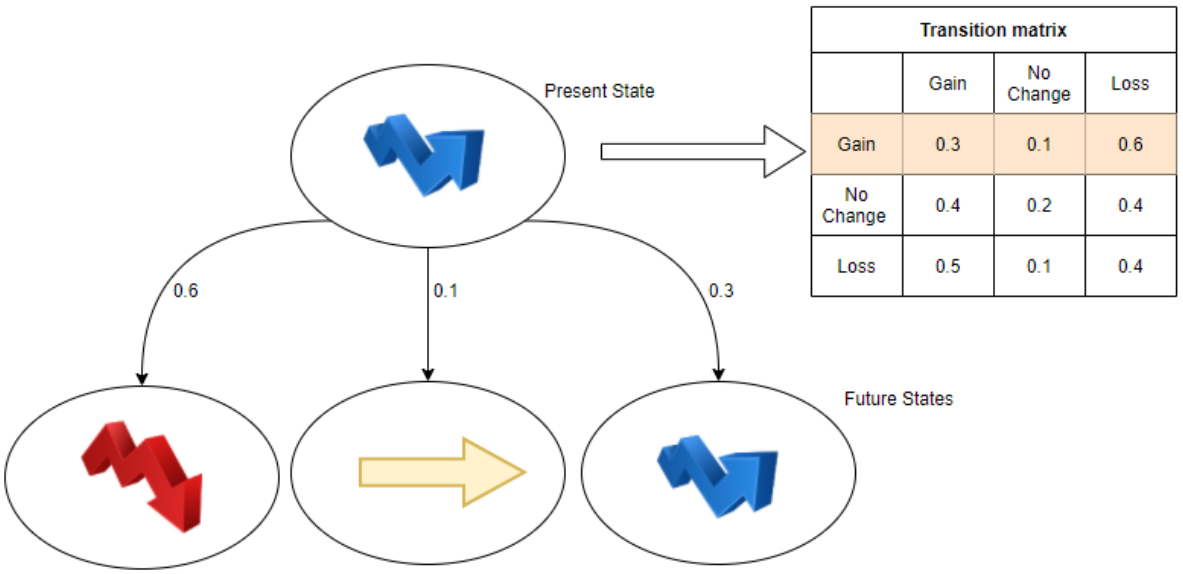
4.1.2.6 Observations

Observations are similar things as rewards. These are the secondary information provided to an agent from the environment. Observations can be helpful or not for an agent, but usually, these observations help the agent choose the right action. Observations represent in some form the state of the environment and can be helpful indicators. Observation can also have reward information (e.g., the moving stock price average). It is essential to distinguish between observations and rewards. Observations can be helpful or have no impact on an agent. The reward is the main driving force of agents. To sum up, observations are additional information that can improve the learning process of an agent.

4.1.3 Markov Decision Process

4.1.3.1 Process

Let us suppose that there is a system that can only be observed. The system can switch between various states (e.g., warm → cold), but the observer cannot force the change. From the system, we can create a list of subsequent states that happen over time (e.g. [warm, cold, cold, warm, warm, warm, cold, …]. The list should have a finite number of states, but the number can be huge. The sequence of observations over time is called history. All available states that a specific system can have is called **state space** (e.g. [warm, cold]). In order to say that this system implements the **Markov Process**, it needs to achieve the Markov property, which defines that the future state can be predicted only using the present state, **not the past**. In other words, the future state does not depend on the past states (or past N states) but only on the present state (what has happened before does not matter, what matters is now). This definition implies one crucial thing. States are unique and should be distinguishable from each other. If we can obtain **state space** (available states) and the current state, we can predict the future state using probability.



Transition matrix			
	Gain	No Change	Loss
Gain	0.3	0.1	0.6
No Change	0.4	0.2	0.4
Loss	0.5	0.1	0.4

Fig. 14. Simplified example of the stock market system. In the present state, the stock price is after gains. In the future, the price can go up, down, or not change. We set the transition probability to the following specific state using the current state. The left side of the picture presents the present state and available transitions with probability values. The top-Left corner presents a transition stable with all available transitions.

The list of following subsequent states that were observed during one experiment can be called an episode.

4.1.3.2 Reward

Systems that behave according to the Markov Process can transition from one state to another. If the system can reach various states, we introduce a reward system for reaching a specific state. The reward can be either negative or positive and does not need to be a constant value. The next subsequent transition of states can be grouped into a list called an episode. It can represent the whole experiment from one initial state to the last state (e.g., one trading year containing 252 trading days). The episode is a series of state transitions. Every transition of state has a reward. A question arises: How can the rewards be accumulated for the whole episode? How should each reward contribute to the overall result of the episode? Markov process describes the definition of return at a given time t as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{n} \gamma^k R_{t+k+1}$$

where:

R_{t+1}- reward from a state transition at time t+1

γ- discount factor (gamma)

Discount factors can have values from range <0, 1>. In other words, the discount factor determines how future transitions contribute to overall episode return. If the discount factor is equal to 1, every transition all transitions contribute equally to the episode return. If the discount factor equals 0, only the first transition contributes to the episode return. We can see how far we should look into transitions to estimate return using the discount factor. In most RL cases, the discount factor is between 0.9 and 0.99.

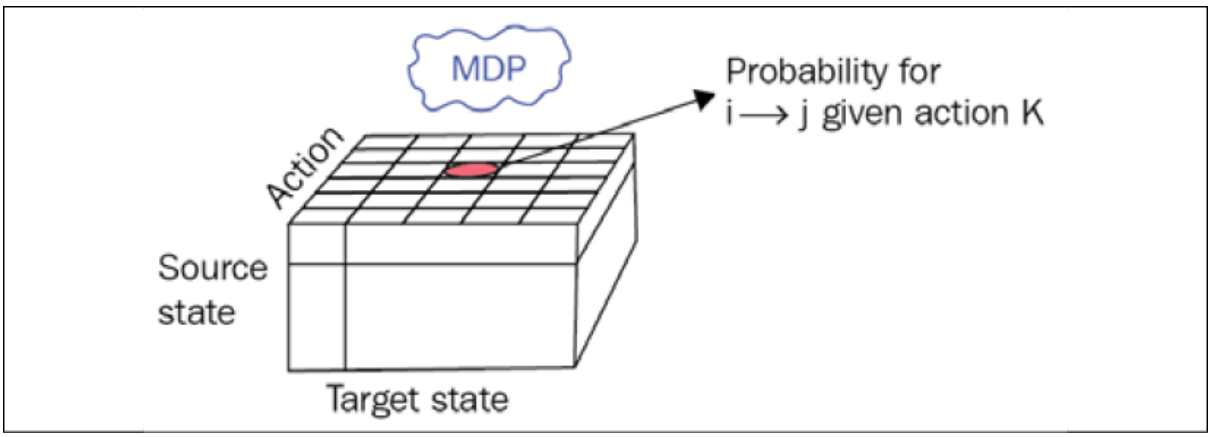
4.1.3.3 Value of state

By observing a chain of transitions in the Markov Process, we can get a value of expected return for a given state by averaging Markov chains for a given state.

$V(s) = E[G|S_t = s]$

4.1.3.4 Action space

Previously, the transition matrix was shown as the square matrix with source states in rows and target rows in columns. Each matrix column presented the transition probability from the source to the target state. Right now, we want to introduce actions to make any decisions. We must define a list of available actions an agent can perform. This can be described as an action space. We would like to connect available actions and states. In order to achieve it, we need to add dimension to the transition matrix. Additional dimensions will be a list of actions. This 3D matrix can estimate the transition probability from state i to state j using k action.



Fix. 15. Transition Probabilities. Author Miguel Morales [3]**.**

By looking at the 3D matrix, It can be deduced that probability depends not only on the source and targeted states. Right now, the transition probability also depends on the action the agent wants to perform.

4.1.3.5 Policy

Policy defines how an agent should behave. The environment can have multiple policies. The policy can impact returns. More technically, a policy can be defined as follows:

$$\pi(a|s) = P[A_t = a|S_t = s]$$

where:

A_t - action at step t

S_t - state at step t

In other words, the policy is a probability distribution over action for each state. Different policies have different outcomes, so defining the correct policy is essential. In Markov Decision Processes, the policy is defined only once and is not changing during the process.

4.1.3.6 Bellman Equation

Value in Deep Reinforcement Learning techniques can be described as an expected total reward, ie:

$$V(s) = E\left[\sum_{t=0}^{\infty} r_t \gamma^t\right]$$

where:

r_t - local reward from step t (single transition from state t-1 to t)

γ^t - discount factor where ($\gamma \epsilon [0, 1]$). The parameter determines how future rewards affect the total reward.

Let us suppose that there is an environment with four states (S_1, S_2, S_3, S_4). The agent begins from the state S_1 and can either move to the state S_2or S_3or S_4. The rewards of each action are known.

- Transition $S_1 \rightarrow S_2$has reward equal to 2
- Transition $S_1 \rightarrow S_3$has reward equal to -2
- Transition $S_1 \rightarrow S_4$has reward equal to 3

We can agree that transition $S_1 \rightarrow S_4$ has the best outcome (highest reward). From these examples, we can assume that our algorithm should always choose actions with the highest rewards. Nevertheless, is it always true?

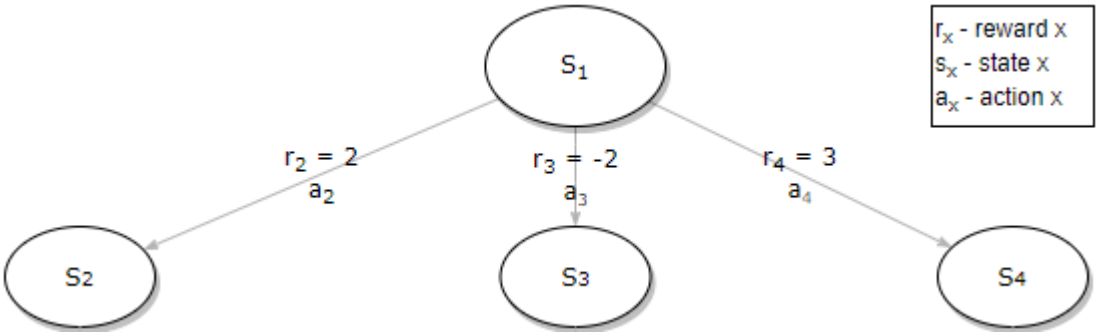


Fig. 16. Simple environment where agent starts from state S_1 and has three actions to choose from: move to state S_2, move to state S_3, move to state S_4.

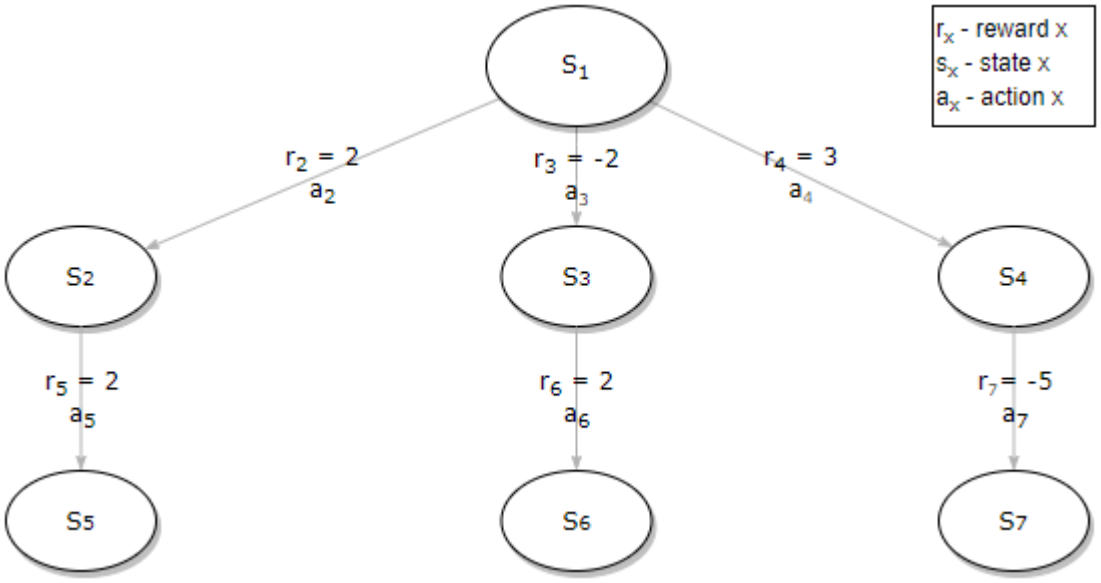
Let us add additional states:

- S_5 that is reachable only from state S_2 with a reward equal to 2
- S_6 that is reachable only from state S_3 with a reward equal to 2
- S_7 that is reachable only from state S_4 with a reward equal to -5

If we assume policy where states S_5,S_6,$S_7$are the states that finish the episode and the agent cannot go back, there are three possible episodes:

- Episode $S_1 \rightarrow S_2 \rightarrow S_5$ has reward 2 + 2 = 4
- Episode $S_1 \rightarrow S_3 \rightarrow S_6$ has reward -2 + 2 = 0
- Episode $S_1 \rightarrow S_4 \rightarrow S_7$ has reward 3 + -5 = -2

This example shows that choosing actions with highest rewards is not always the best solution. In this case, choosing actions with the highest rewards (transition $S_1 \rightarrow S_4$) leads to the worst outcome (episode reward is -2).



Fix. 17. An expanded environment where the agent starts from state S_1 and has 3 actions to choose: move to the state S_2, move to state S_3, move to state S_4. States S_5, S_6, S_7 are reachable from states S_2, S_3, S_5 respectively.

In order to get better outcomes, future rewards must be considered. Value of specific action we can describe as follows:

$$V_0(a = a_i) = r_a + \gamma V_a$$

where:

r_{a_i} - an immediate reward for taking action a_i

V_a - a long-term reward for the next states

γ - discount factor where ($\gamma \epsilon [0, 1]$). The parameter determines how future rewards affect the total reward.

The value of action $S_1 \rightarrow S_2$ will be:

$V_0(a = a_2) = r_2 + \gamma V_2$ because $V_2 = V_2(a = a_5) = r_5 + \gamma V_5$

$V_0(a = a_2) = r_2 + \gamma V_2 = r_2 + \gamma(r_5 + \gamma V_5) = r_2 + \gamma(r_5)$ because γV_5

The equations tell that future states are included in the total value (with discounted value).

The above equation can be expanded to find the best solution:

$V_0 = max_{a \in 1 \ldots N}(r_a + \gamma V_a)$

Previous examples were pretty straightforward. Each action had the same consequences (e.g., action a_2 causes always $S_1 \rightarrow S_2$ transition). However, more sophisticated models have actions that can have various consequences based on probability. Let us assume that action a_2 can do $S_1 \rightarrow S_2$ transition with p_2= 50% probability and $S_1 \rightarrow S_3$ transitions with p_3= 50% probability.

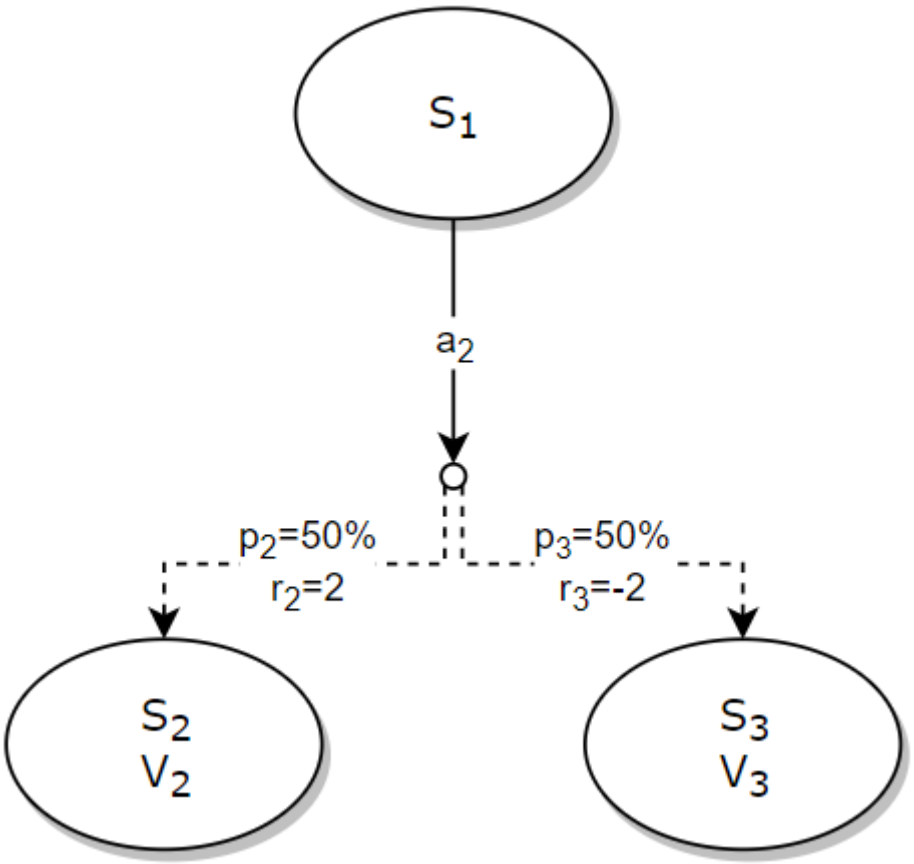


Fig. 18. Example of action, where the outcome is not determined. There is a 50% probability that action a_2 will cause $S_1 \rightarrow S_2$ transition and a 50% probability that the same action will cause $S_1 \rightarrow S_3$ transition.

The sum of all probabilities should be equal to 1 ($p_2 + p_3 = 1$). The expected value of an action a_2 can be described as:

$$V_0(a = a_2) = p_2(r_2 + \gamma V_2) + p_3(r_3 + \gamma V_3) = \sum_{s \epsilon S} p_{a_2, s_1 \to s}(r_{s,a} + \gamma V_s)$$

Simplifying the formula, we have:

$$V_0(a) = \sum_{s \epsilon S} p_{a,0 \to s}(r_{s,a} + \gamma V_s) = E_{s \sim S}[r_{s,a} + \gamma V_s]$$

Using this equation, we can determine the Bellman optimality equation as:

$$V_0 = max_{a \epsilon A} \sum_{s \epsilon S} p_{a,0 \to s}(r_{s,a} + \gamma V_s) = max_{a \epsilon A} E_{s \sim S}[r_{s,a} + \gamma V_s]$$

The Bellman optimality equation tells that the optimal value is equal to the action which gives the best current reward + long-term discounted reward for the next state.

4.1.3.7 Q-learning

In order to explain Q-Learning, we need to define the Q function. The previous section showed us how to get the Bellman equation of the value function. The Bellman equation of the Q-value function is similar, ie:

$$Q_{\boldsymbol{\pi}}(s, a) = r_{s -> s', a} + \gamma Q_{s', a'}$$

where:

$r_{s -> s', a}$ - an immediate reward of moving from state ***s*** to state ***s'*** by action ***a.***

$Q_{s', a'}$ - Q-value of the next state-action pair

γ - discount factor

$\boldsymbol{\pi}$ - policy, that describes the strategy of the agent to select certain action in state s

Q-value is the expected return by starting in state s and taking action a and then following the policy $\boldsymbol{\pi}$

Q-Learning algorithm tries to improve constantly the Q-value through the iteration. Using the ε-greedy policy, the algorithm chooses the following function to update Q-value:

$$Q_{\boldsymbol{\pi}}(s,a) \leftarrow Q_{\boldsymbol{\pi}}(s,a) \; + \alpha\left[r_{s->s',a} + \gamma \cdot max_{a' \epsilon A}(Q(s',a') \;) - Q_{\boldsymbol{\pi}}(s,a))\right]$$

where:

$Q_{\boldsymbol{\pi}}(s,a)$ - current Q-value

α - learning rate

$r_{s->s'}$- immediate reward

γ - discount factor

$max_{a' \epsilon A}(Q(s',a') \;)$ - the maximum possible Q-value of next state which can be obtained through action $a' \epsilon A$

4.1.4 Deep Q Network

Basic DQN algorithm describes Q-value as:

Deep Q Network is a neural network with multiple layers. The input of the neural network is n-dimensional state space. The output has m actions. Estimating the best action is done using a neural network by selecting the output of the NN with the highest value.

Deep Q-Learning selects Q-value by learning weights of DQN. Then Deep Q Network maps states into actions:

$Q(s,a,\Theta) \approx Q^{*}(s,a)$

where:

s - state

a - actions

Θ - set of weights of DQN

The DQN algorithm learns to improve its performance by adjusting the weights of the neural network to minimize the difference between the predicted Q-value and the target Q-value. The target Q-value is calculated using the Bellman equation, which is a recursive formula that represents the expected future reward for each state-action pair. The algorithm uses gradient descend method that computes the squared difference between target Q-value and current predicated Q-value. It uses a deep neural network with multiple layers to estimate the Q-value function, which represents the expected future reward for taking a particular action in a given state.

4.1.5 Extensions

4.1.5.1 Noise Network

Addition of Nosie Network (NN) is a technique in DRL that adds additional stochastic perturbation to the input observations. This noise is then added to the observations. This approach helps the agent to cope with incomplete observations in the environment and introduces robustness and generalization. The stock market is a good example where incomplete data are used to predict the next moves.

4.1.5.2 Double Deep Q Network

Double Deep Q-Network (DDQN) was proposed to improve Deep Q-Network by DeepMind researchers in 2016. In basic Q-Learning, there is a problem with overestimations of action value.

DDQN uses two neural networks: the online network and the target network.

Step No.	Online Network	Target Network
1	Calculate quality for all actions: $Q^{online}(s_{i+1}, a_i)$	
2	Get action with the best quality: $argmax(Q^{online}(s_{i+1}, a_i)) \rightarrow a_{best}$	

3		Calculate quality for best actions(a_{best}): $Q^{target}(s_{i+1}, a_{best})$
4	Use Q^{target} to update Online Network weights	
e.g. after 1000 steps		Update weights using Online Network's weights

4.1.5.3 Dropout Network

Dropout is a regularization technique in deep learning that can help prevent overfitting of neural networks. In a dropout network, a subset of neurons is randomly dropped out or "turned off" during each training iteration, reducing their contribution to the output of the network.

In a dropout network, the dropout rate is defined by a parameter that determines the part of neurons that are randomly dropped out during training. Typically, dropout rates between 0.1 and 0.5 are used, meaning that 10% to 50% of the neurons dropped out during each training iteration. During training, the dropout process is applied to each layer.

5. Creating Trading Agent

In this step, the thesis explained all the necessary theories behind finances and reinforcement learning. This part will focus on the implementation of financial data and Deep Reinforcement Learning in the form of a trading bot that, based on financial indicators, makes decisions about investing in order to make profits. The implementation of the trading agent is written as the script application in Python Language. The application connects with an external API to get historical prices of the specific asset and process prices and, using Deep Reinforcement Learning Techniques, predict where the price will go the next day. Based on this prediction, the trading bot simulates taking long/cash/short positions on the market. This simulation can be helpful for real traders to take an actual long/short position in a broker account. Many trading platforms (e.g., cryptocurrency trading

platform Binance) offer APIs that can place orders using HTTP requests (trading bot can be extended to send these requests automatically). The trading bot has two phases of running. Firstly, in the learning phase, the bot learns how to play on the stock market using historical asset prices. This stage is the longest one because it requires thousands of learning iterations. During this phase, the trading bot updates the weights of its neural networks (during training) and saves the weights to file (to preserve neural networks). The second phase, execution one, tries to predict the next market moves using existing (from the learning phase) neural networks.

5.1 Data Preparation

The primary information used to create alpha factors and then train the neural network was historical stock prices with volumes with the following fields:

- “date” - date of the trading day
- “open” - the price at which an asset first trades upon the opening of an exchange
- “low” - the lowest price of the asset at a given trading day
- “high” - the highest price of the asset at a given trading day
- “close” - the last price at which an asset trades at a given trading day
- “volume” - the amount of an asset that changes hands over the course of a trading day

This type of aggregated market data are called OHLCV (Open, High, Low, Close, Volume).

The data was provided in JSON format:


```
[ {
       "date" : "2020-03-02 15:59:00",
       "open" : 297.230000000000,
       "low" : 297.230000000000,
       "high" : 298.280000000000,
       "close" : 298.252300000000,
       "volume" : 78679246
  }, {
       "date" : "2020-03-02 15:58:00",
       "open" : 296.190000000000,
       "low" : 296.190000000000,
       "high" : 297.430000000000,
       "close" : 297.230000000000,
       "volume" : 77982786
  }, {
       "date" : "2020-03-02 15:57:00",
       "open" : 295.860000000000,
       "low" : 295.860000000000,
       "high" : 296.580000000000,
```

```
        "close" : 296.190000000000,
        "volume" : 77400704
    }, ...
]
```

The data source is Financial Modeling Prep API which provides real-time stock price, company financial statements, major index prices, historical stock data, forex real-time rate, and cryptocurrencies. The Financial Modeling Prep API goes up to 30 years back in history. For this work, the prices were obtained for each trading day. The prices and volumes were adjusted, meaning that all corporate actions, like stock splits, were taken into account to reflect historical prices accurately.

	open	high	low	close	volume
2003-09-11 00:00:00	27.62000	28.35000	27.29000	28.03000	56780983
2003-10-01 00:00:00	27.82000	28.73000	27.81000	28.63000	59919488
2003-10-02 00:00:00	28.58000	28.95000	28.41000	28.62000	42558926
2003-10-03 00:00:00	29.27000	29.80300	29.20000	29.61000	66587711
2003-10-06 00:00:00	29.68000	29.80000	29.38000	29.56000	28963086
2003-10-07 00:00:00	29.28000	29.99000	29.15000	29.95000	47948306
2003-10-08 00:00:00	29.92000	30.00000	29.50000	29.67000	41063128
2003-10-09 00:00:00	29.99000	30.39000	29.48000	29.77000	75925049
2003-10-10 00:00:00	30.16000	30.49000	30.09000	30.43000	46198463
2003-10-13 00:00:00	30.71000	30.94000	30.48000	30.80000	41483004
2003-10-14 00:00:00	30.79000	31.10000	30.53000	31.08000	68244550
2003-10-15 00:00:00	32.77000	32.78000	31.62000	31.76000	110210043
2003-10-16 00:00:00	31.50000	32.26000	31.40000	32.23000	55166354
2003-10-17 00:00:00	32.27000	32.39000	31.57000	31.66000	53272419
2003-10-20 00:00:00	31.61000	32.21000	31.56000	32.16000	43139650
2003-10-21 00:00:00	32.28000	32.33000	32.01000	32.12200	50316117
2003-10-22 00:00:00	31.77000	32.50000	31.41000	31.52000	49911598
2003-10-23 00:00:00	30.92000	31.39100	30.89000	31.22000	48010490

Fix. 19. Raw asset price DataFrame from Financial Modeling Prep API.

5.2 Setting the input

In order to provide data for neural networks, different trading indicators were provided as input. Implementation of these trading algorithms was provided by the open-source python library TA-Lib which is a technical analysis library. Based on the closing/open/low/high price and volume at a given day, TA-Lib calculates different trading indicators like Relative Strength Index, Aroon, Average True Rate, etc. The following indicators were used as the input:

- 1-day percentage change between the current day and a previous day

- 2-day percentage change between the current day and a previous day
- 5-day percentage change between the current day and a previous day
- 10-day percentage change between the current day and a previous day
- 21-day percentage change between the current day and a previous day
- Schaff Trend Cycle (STC)
- Aroon Up Indicator
- Aroon Down Indicator
- Parabolic Stop and Reverse (Parabolic SAR) Downward Trend Indicator
- Stochastic RSI
- Moving Average Convergence Divergence
- Average True Rate
- Stochastic Oscillator
- Ultimate Oscillator (ULTOSC)

Please keep in mind that in order to use these inputs correctly, all values must be normalized (values from 0 to 1). The input will be provided to neural networks, which have simple biases and weights, and comparing, e.g., asset values from different periods, will not make any sense. Single asset price says nothing about future trends. However, the normalized percentage price change can say something about a short trend. The data of a single asset is, e.g., from the last ten years, and the market looked different back then. The training data will be from various periods and therefore feeding neural networks with absolute price values will be incorrect.

Fix. 20. During the Dot-Com Bubble Amazon stock dropped approximately 80%. In absolute price, it was ~3 USD (price adjusted). Today, minor corrections like ~15% can cause a price drop of ~20 USD. This example shows how inaccurate absolute values are.

The preprocessed data were stored in panda DataFrame, where each row represents a trading day and has a specific trading indicator.

	returns	ret_2	ret_5	ret_10	ret_21	stc	aroon_up	aroon_down	psar_down_indicator	rsi	macd	atr	stoch	ultosc
2008-01-16 00:00:00	-6.92116	-5.67905	-3.34130	-4.17583	-3.31883	-1.17536	-0.34322	1.43106	-0.22579	0.26784	-1.81405	1.22850	-0.17338	-1.70281
2006-01-18 00:00:00	-6.39800	-5.07056	-3.56764	-2.27352	-2.06958	-0.05787	-1.38541	1.43106	4.42893	-1.54323	-0.12547	0.18760	0.29858	-1.93775
2004-07-14 00:00:00	-5.90327	-4.46948	-2.98054	-2.97470	-2.26787	-1.17621	-1.50120	1.43106	-0.22579	-0.76712	-0.79663	0.53060	1.24727	-1.63628
2008-09-29 00:00:00	-5.62092	-2.87918	-1.77888	-2.11619	-3.63885	-1.17621	-1.15381	1.43106	4.42893	-0.08581	-2.01669	1.05452	0.22171	-1.60180
2016-01-15 00:00:00	-5.09125	-2.77197	-1.49219	-2.65457	-2.12657	-1.17566	-0.11162	1.43106	-0.22579	0.09047	-0.92780	1.03755	0.37509	-1.76335
2019-04-26 00:00:00	-5.02963	-4.39299	-2.75091	-1.20770	-0.26245	-0.35787	0.81477	1.43106	4.42893	-1.54323	2.14273	2.65399	1.83997	-0.86334
2008-12-01 00:00:00	-5.02629	-4.14058	-1.13645	-1.14396	-3.04564	0.84752	-0.80641	0.73882	-0.22579	0.30732	-1.67122	1.07502	1.17195	-0.55850
2018-07-27 00:00:00	-4.80521	-3.71909	-2.17168	-1.71477	-0.37426	0.04710	-1.50120	1.43106	-0.22579	-0.13951	-0.72064	2.57460	1.27741	-0.85918
2008-01-04 00:00:00	-4.53653	-4.33594	-4.09797	-2.62139	-1.91783	-0.95924	-0.92221	1.43106	-0.22579	-1.54323	-0.06691	0.79554	0.13057	-1.96463
2008-11-06 00:00:00	-4.42208	-6.01916	-3.76249	-0.89660	-2.02558	1.06734	-1.50120	1.43106	4.42893	-0.54105	-1.71887	1.79586	1.81903	-1.06694
2008-11-14 00:00:00	-4.30532	-0.62712	-2.38260	-3.28437	-2.22148	-0.79036	-1.50120	1.43106	-0.22579	-0.36620	-1.64957	1.59982	-0.53125	-0.51546
2006-07-20 00:00:00	-4.19585	-2.50108	-0.94819	-1.82339	-0.84558	-1.12482	-0.11162	1.43106	4.42893	0.46114	-0.14598	-0.29462	-0.34925	-1.57198
2008-11-05 00:00:00	-4.13125	-1.53636	0.17242	0.57480	-0.87606	1.10640	-1.38541	0.62345	-0.22579	0.07726	-1.77850	1.75725	1.51015	-0.09083
2004-09-03 00:00:00	-4.08921	-2.65036	-2.38056	-1.44120	-2.11158	0.84441	-1.50120	1.43106	-0.22579	-0.46285	-1.45567	0.18523	-0.02356	-1.07575
2008-10-02 00:00:00	-3.99040	-3.35566	-1.96973	-2.09714	-2.75649	-1.17621	-1.50120	1.43106	-0.22579	0.82102	-2.05209	1.23970	0.11257	-0.90386
2008-12-18 00:00:00	-3.67019	-3.62276	0.42936	2.17472	1.08519	1.12310	1.04636	-0.76104	-0.22579	0.15703	-0.31094	1.01109	1.05559	0.28296
2008-12-04 00:00:00	-3.64917	-1.59017	-2.28719	0.37416	-2.93119	1.08868	-1.15381	0.39270	-0.22579	0.40911	-1.49193	1.03624	-0.14525	-0.45290

Fig. 21. Preprocessed asset data by TA-Lib.

5.3 Design ML model to generate signals

After the aggregated OHLCV data are preprocessed by the TA-Lib library, they can be provided to the neural network. The trading bot has five layers of a dense neural network. The neural network's input layer consists of the normalized values of financial indicators (Average True Rate, Stochastic Oscillator, etc.). Then there are three dense layers with 256 neurons in each layer. The output layer contains three neurons that return float values from 0 to 1. Each output-layer neuron represents each action (Short Position, Keep Cash, Long Position). The action of the neuron with the highest value is selected as the next trading bot move. The activation function of each neuron is ReLU.

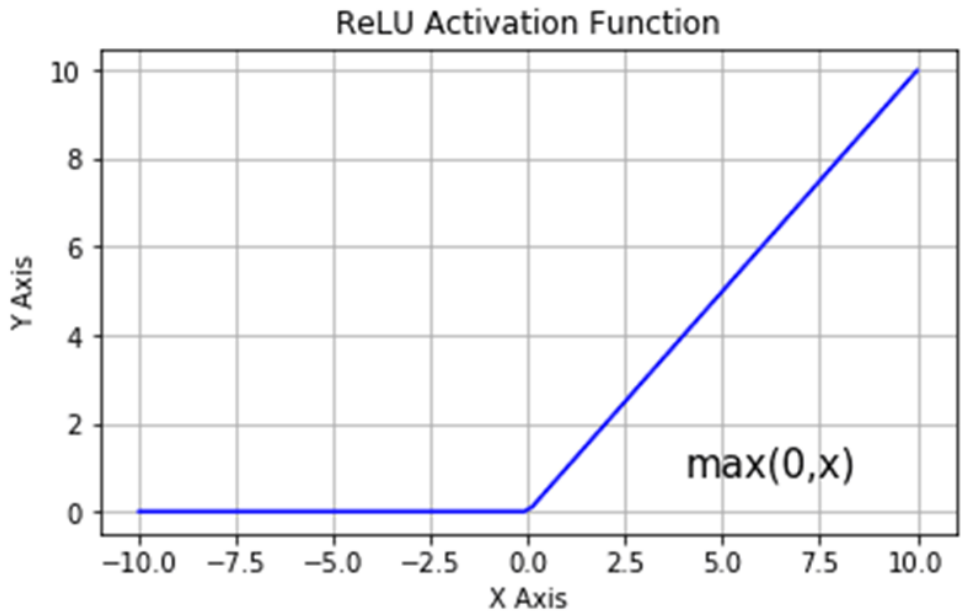


Fig. 22. ReLU activation function. Image adapted from https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/.

In order to minimalize and optimize parameters, the Adam optimizer is implemented across all neurons. It is a stochastic gradient descent method. To compute the error, the Mean Square Error (MSE) is used.

The last hidden layer is the Dropout layer. This layer, with a certain frequency, this layer sets inputs to o to specific neurons. This technique prevents overfitting. The selected dropout rate value for the last hidden layer network is 0.1, meaning there is a 10% chance that the input to the neuron will be 0 during training (will not take part in the training process).

The training bot has two types of learning algorithms. Deep Q-Learning uses one neural network to predict the next actions (described above). The second option - Double Depp Q-Learning, uses two neural networks with the same structure but slightly different weights. The first neural network predicts the next action (Online Network). Then the quality of choice is calculated by the second neural network (Target Network) calculates the choice's quality. The quality of choice is then used to update the weights of the Online Network. After a couple of (e.g., 100 steps), the Target Network is updated using the weights of the Online Network.

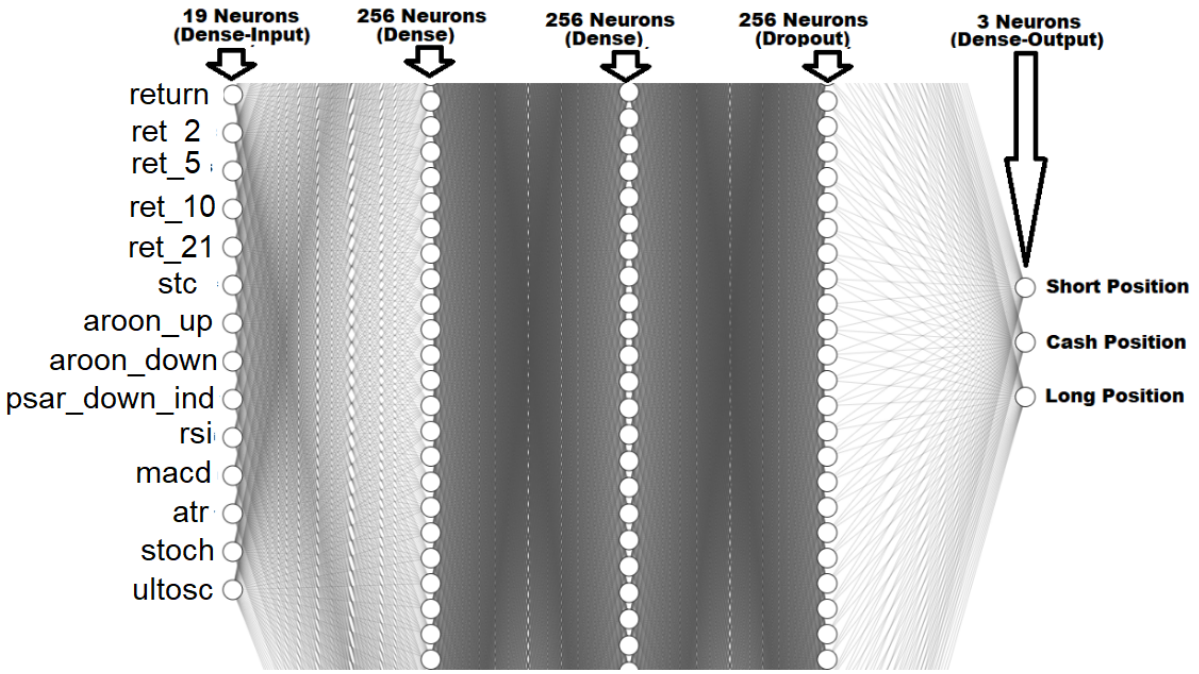


Fig. 23. Visualisation of neural network used in the trading bot.

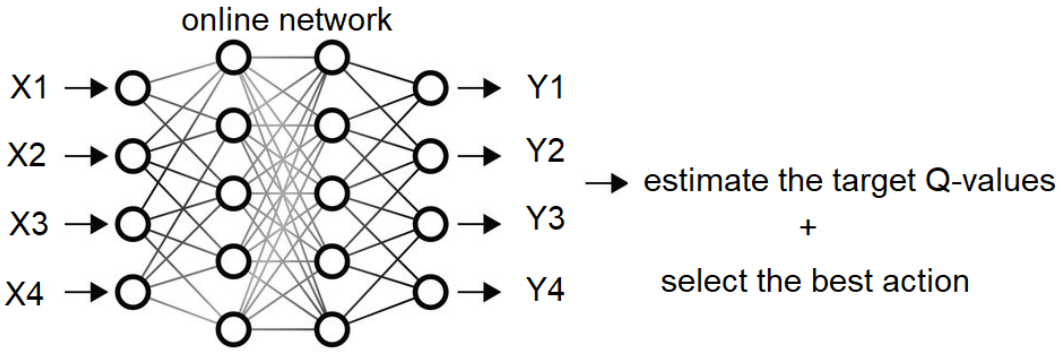


Fig. 24. Visualisation of neural network in DQN configuration.

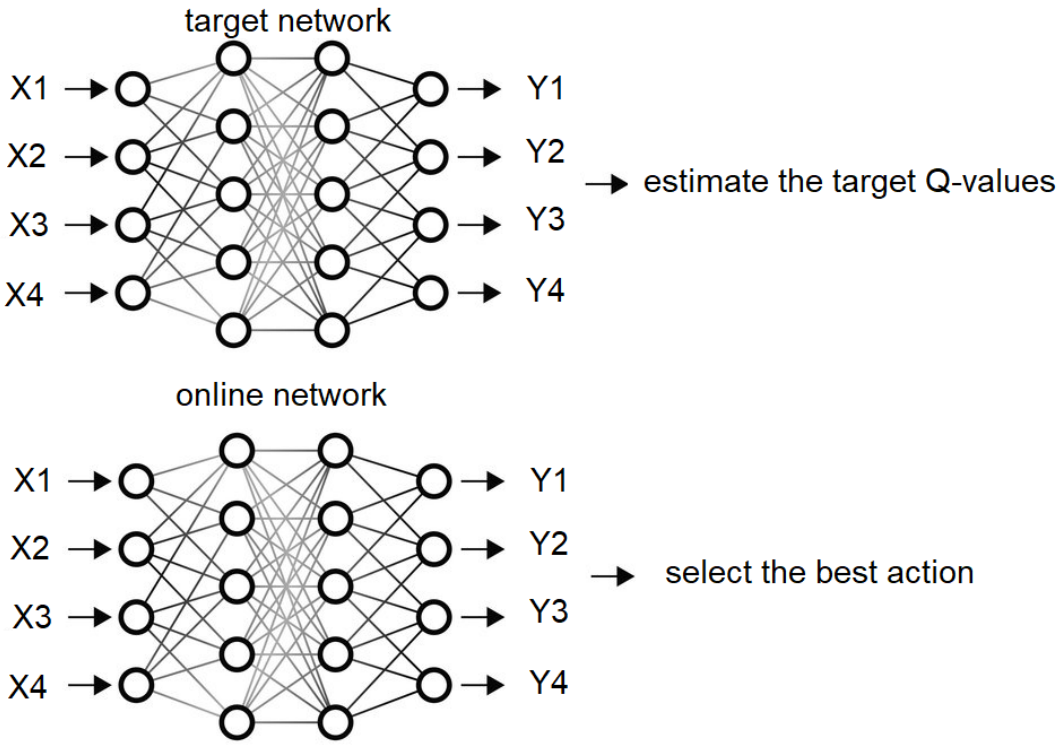


Fig. 25. Visualisation of neural network in DDQN configuration. The online network is used to select the best action in a given state based on its current estimates of Q-values. During each training iteration, the online network is updated using a stochastic gradient descent algorithm to minimize the difference between its predicted Q-values and the target Q-values. The target network, on the other hand, is used to estimate the target Q-values that are used in the Q-learning update.

5.4 Learning process

5.4.1 Training Neural Network (NN)

The trading bot can run in 2 modes - training and execution. During training mode, based on historical stock data, the trading bot tries to predict the correct position for the next trading day to have a positive return. The correctness of the action is immediately validated using historical data. If the decision is wrong (negative return), specific trading actions are applied to update neural networks. The neural network was not updated instantly after each observation (single trade). Firstly, to train the network, the amount of observation must be more significant than 4096. When there were more than 4096 observations during each trade, the algorithm took 100 observations (in random order) and used them to train NN.

The workflow can be described the following way:

1. Wait until there are more than 4096 observations

2. If there are more than 4096 observations, select 100 random observations and use them for training the network.

The DDQN method has two NN - online NN and target NN. The online NN is trained like the above. After nth trades, the target NN replaces its own weights with the weights of the online NN. The parameter τ defines the frequency of target network update in the following way:

```
        if self.total_steps % self.tau == 0:
            self.update_target()
```

If τ is equal to 100, then for every 100 steps, the target NN will be updated. The workflow can be described as follows:

1. Wait until there are more than 4096 (**batch_size**) observations
2. If there are more than 4096 observations (**batch_size**), select 100 random observations and use them for training the **online NN**
3. After 100 training (or defined by τ) of **online NN** (step 2). Copy the weights of the **online NN** and update the **target NN** with these weights in the following way:

```
self.target_network.set_weights(self.online_network.get_weights())
```

5.4.2 Decision Making Policy

The NN, in the beginning, has weights with zero values and therefore the outputs will be with zero value too. Even after initial training, the value of outputs will make no sense. Point 4.1.1 explains what exploration and exploitation are. In the beginning, there is nothing to exploitation, and the decision of what action needs to be chosen needs to rely on exploration. In terms of the trading bot, exploration is done by selecting a random decision (short, neutral, long position):

```
np.random.choice(self.num_actions).
```

The ratio of exploration/exploitation (probability of selecting exploration over exploitation) decreased over training time and ended with full exploitation:

```
self.online_network.predict(state)
```

When the NN is trained enough, all decisions on which action to choose can be based on prediction from the trained NN, which is exploitation. Between full exploration and full exploitation, there is a parameter **ε** with normalized values from 0 to 1 that define with what probability choose exploration. It begins with a value of 1.0 (always choose exploration) and decreases over time and, in the end, is

equal to 0.0 (always choose exploitation). The below code sample shows how the decision process works:

```
            if np.random.rand() <= self.epsilon:
                return np.random.choice(self.num_actions)
            else:
                q = self.online_network.predict(state)
                return np.argmax(q, axis=1).squeeze()
```

5.5 Results evaluation

The main simulations check the performance of the trading bot using Intel stock (INTC). The bot was learning using historical data from 2003-01-01 to 2020-01-01. Then, using new data from 2020-01-01 to 2022-06-01, the bot performed backtesting to check its behavior in conditions where the new data appears. The bot puts one of three positions (long, cash, short) each day before the market opens. Then, when the stock exchange was closed, the result was validated. Three types of neural networks and algorithms were validated:

- DQN algorithm - basic Q-Learning algorithm with single neural network
- DDQN algorithm - Q-Learning algorithm with two neural networks. One network evaluates Q-value during learning (online network). The second one, the target network, selects the best actions and is updated periodically by the online network's weights (the online network is cloned and replaces the target network periodically). It prevents overestimation bias.
- DDQN algorithm + noisy network - the same algorithm and NN structure as DDQN, but to some neurons, random noise is added to the weights during training (to prevent bias).

For each configuration, there are five different simulations with different amounts of episodes (full trading year training):

- 250 episodes
- 500 episodes
- 1000 episodes
- 1500 episodes
- 2000 episodes

In summary, the DDQN algorithm and DDQN algorithm + noisy network configuration performed quite well. Below are three sections that explain the results in more detail.

5.5.1 DQN with different episodes amount

In this test, a trading bot with Deep Q-Learning (DQN) was tested using different amounts of episodes. The tested stock was Intel (INTC ticker). The bot was trained using a historical daily price of the stock from the date 2003-01-01 to 2020-01-01. The validate the performance, the trading bot was executed using new data from the date 2020-01-01 to the date 2022-06-01. There were five simulations with different amounts of episodes (1 episode - 250 trading days - 1 trading year):

- 250 episodes
- 500 episodes
- 1000 episodes
- 1500 episodes
- 2000 episodes

Trading bot had the following parameters' values:

- batch_size - 4096
- γ - 0.99
- τ - 100 (defines how often update target network)
- $\boldsymbol{\varepsilon}_{start}$ - 1.0 (defines decision making policy and ration of exploration/exploitation)
- $\boldsymbol{\varepsilon}_{exponential\ decay}$ - 0.99 (defines decay of $\boldsymbol{\varepsilon}_{start}$)

The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. In this research (with DQN configuration), the agent could not generate returns better than the market and lost some initial capital.

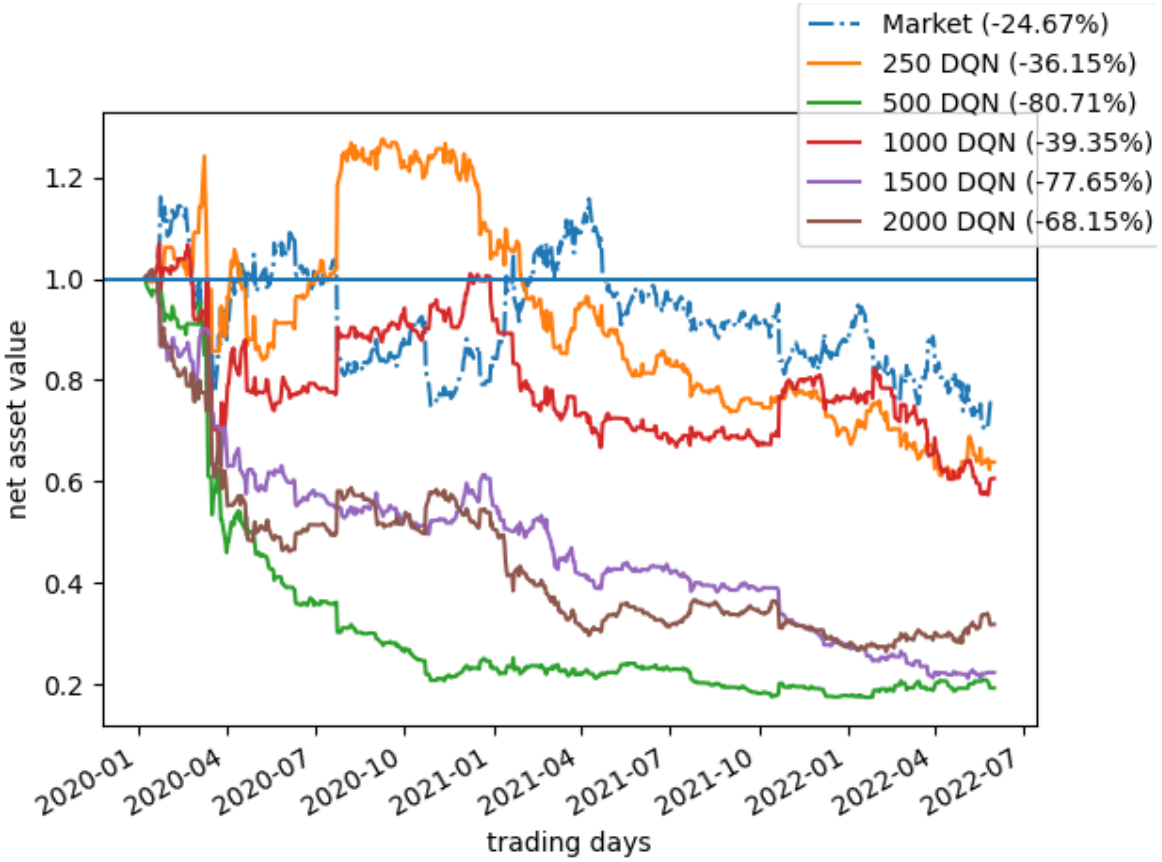


Fig. 26. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

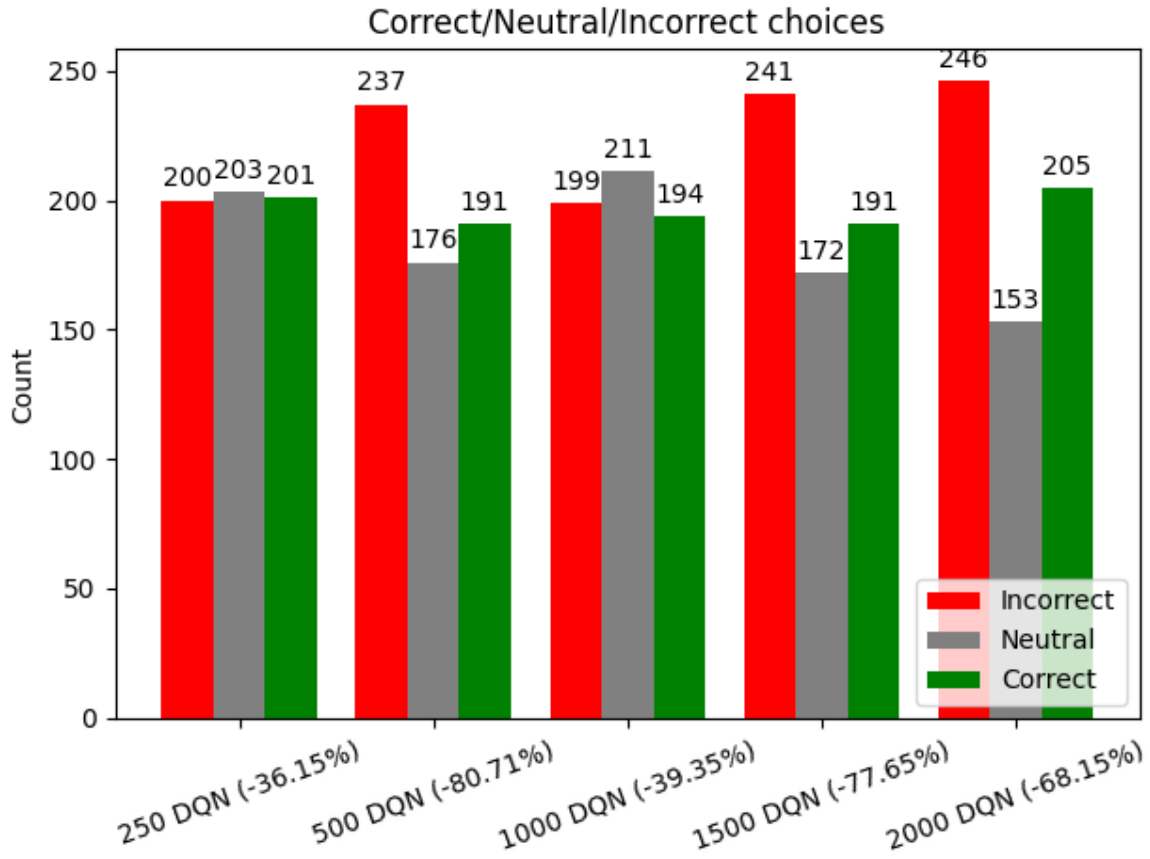


Fig. 27. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The incorrect choice is the position (short or long) that results in a negative return (e.g., a short position when the stock price went up or a long position when the stock price went down). In reverse is the correct choice (e.g., short position when the stock price went down or long position when the stock price went up). The neutral choice is when the agent holds the cash (does set any position).

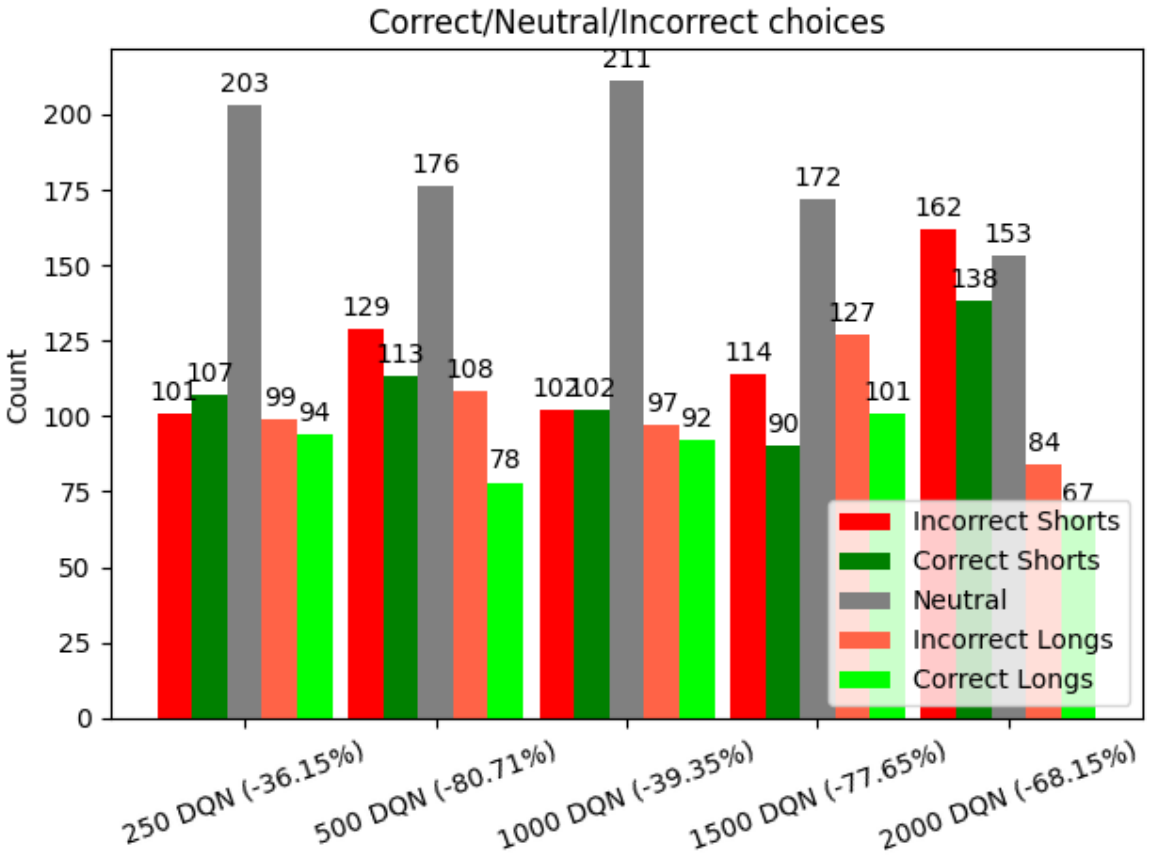


Fig. 28. Five results of agents' games on the stock market and their action correctness distribution (splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The chart shows incorrect/correct short/long positions and neutral positions.

5.5.2 DDQN with different episodes amount

In this test, a trading bot with Double Deep Q-Learning (DDQN) was tested using different amounts of episodes. The tested stock was Intel (INTC ticker). The bot was trained using the historical daily price of the stock from the date 2003-01-01 to 2020-01-01. The validate the performance, the trading bot was executed using new data from the date 2020-01-01 to the date 2022-06-01. There were five simulations with different amounts of episodes (1 episode - 250 trading days - 1 trading year):

- 250 episodes
- 500 episodes
- 1000 episodes
- 1500 episodes
- 2000 episodes

Trading bot had the following parameters' values:

- batch_size - 4096
- γ - 0.99
- $\boldsymbol{\varepsilon}_{start}$ - 1.0 (defines decision making policy and ration of exploration/exploitation)
- $\boldsymbol{\varepsilon}_{exponential\ decay}$ - 0.99 (defines decay of $\boldsymbol{\varepsilon}_{start}$)

The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. The bot trained with 500 episodes (1 episode - 1 trading year - 250 days of trading) and gained 39.01%. With 2000 episodes, the performance of the trading bot decreased rapidly (a loss of 50,87%). Overall, with fine-tuning, the agent could generate positive returns.

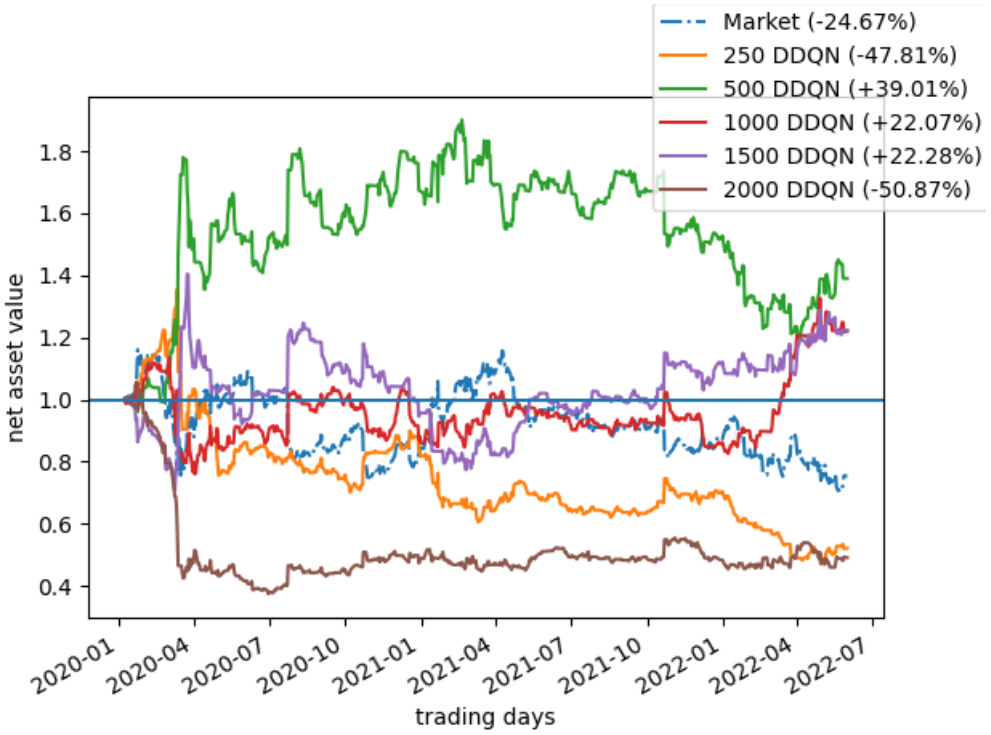


Fig. 29. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

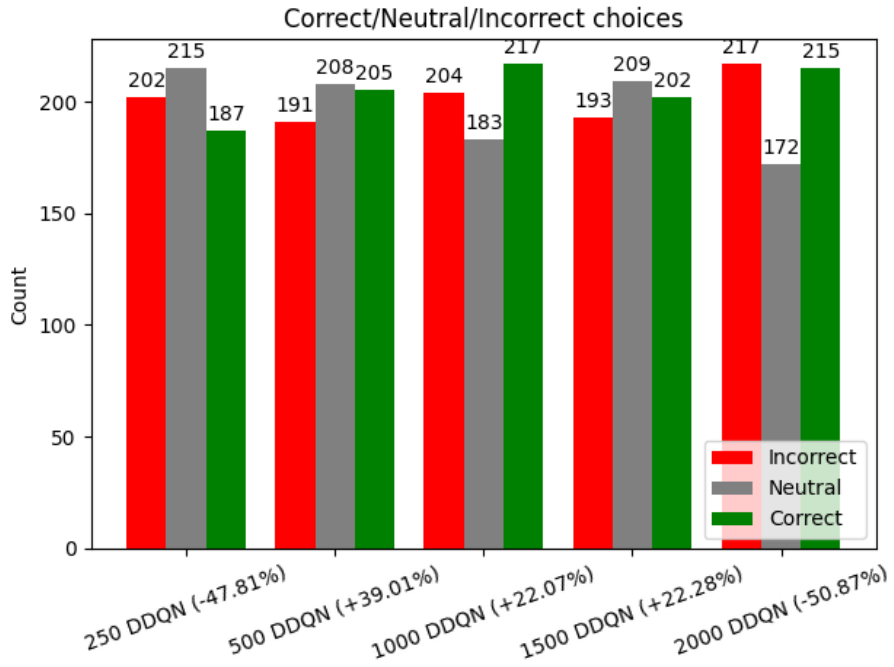


Fig. 30. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. The result with the lowest neutral decisions performed the worst (2000 DDQN).

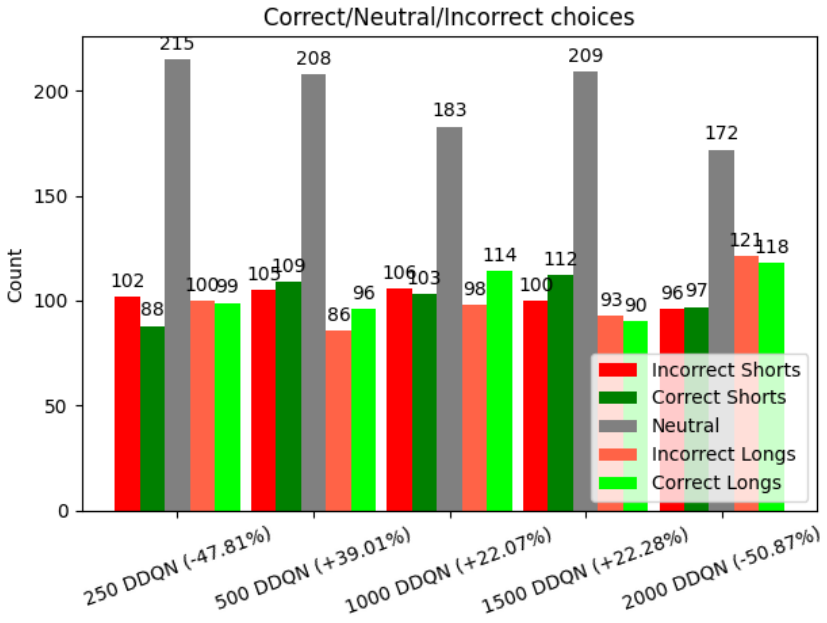


Fig. 31. Five results of agents' games on the stock market and their action correctness distribution (with splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash.

During that time, the stock price dropped by 24.67%. The result with the lowest neutral decisions performed the worst (2000 DDQN).

5.5.3 DDQN + Noisy Network with different episodes amount

In this test, a trading bot with Double Deep Q-Learning (DDQN) + Noisy Network was tested using different episodes. The tested stock was Intel (INTC ticker). The bot was trained using the historical daily price of the stock from the date 2003-01-01 to 2020-01-01. The validate the performance, the trading bot was executed using new data from the date 2020-01-01 to the date 2022-06-01. There were five simulations with different amounts of episodes (1 episode - 250 trading days - 1 trading year):

- 250 episodes
- 500 episodes
- 1000 episodes
- 1500 episodes
- 2000 episodes

Trading bot had the following parameters' values:

- batch_size - 4096
- γ - 0.99
- τ - 100 (defines how often update target network)

The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. The bot trained with 500 episodes (1 episode - 1 trading year - 250 days of trading) was able to gain 44.70%. With 1500 episodes, the performance of the trading bot decreased rapidly (a loss of 59,04%).

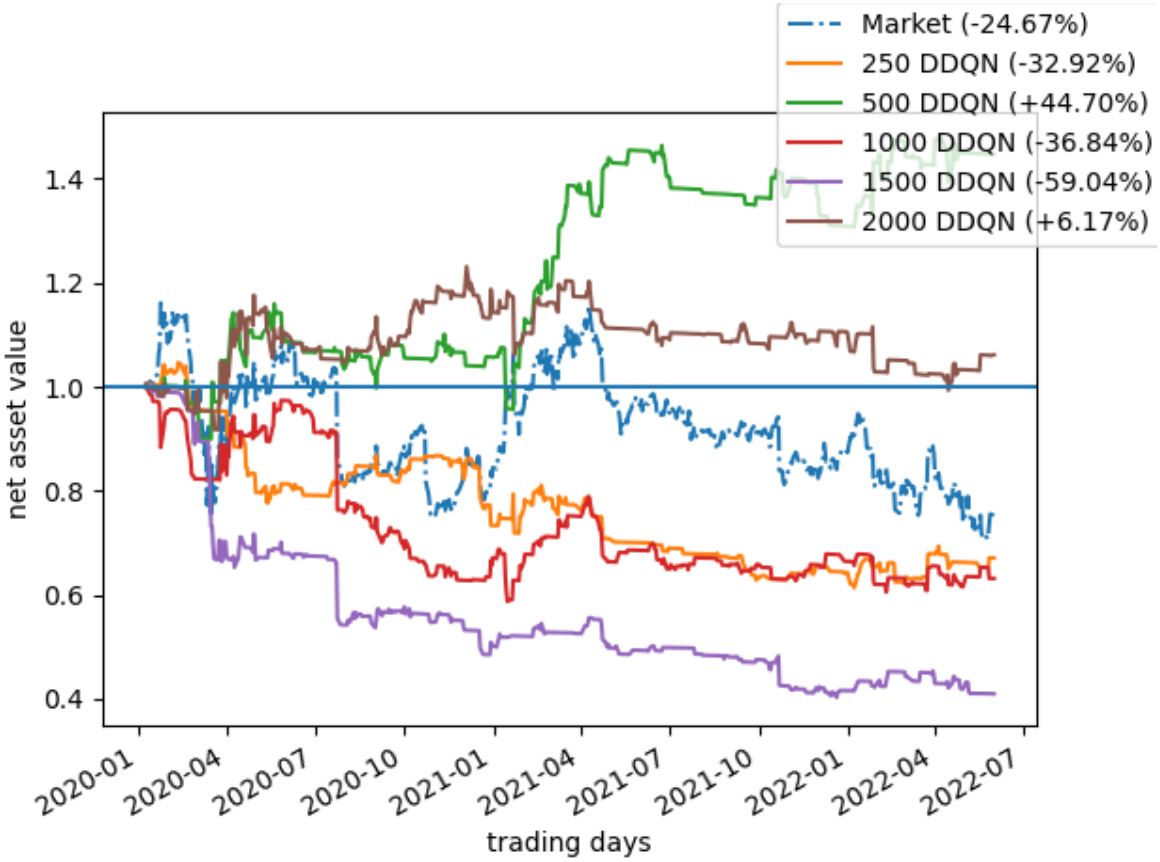


Fig. 32. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

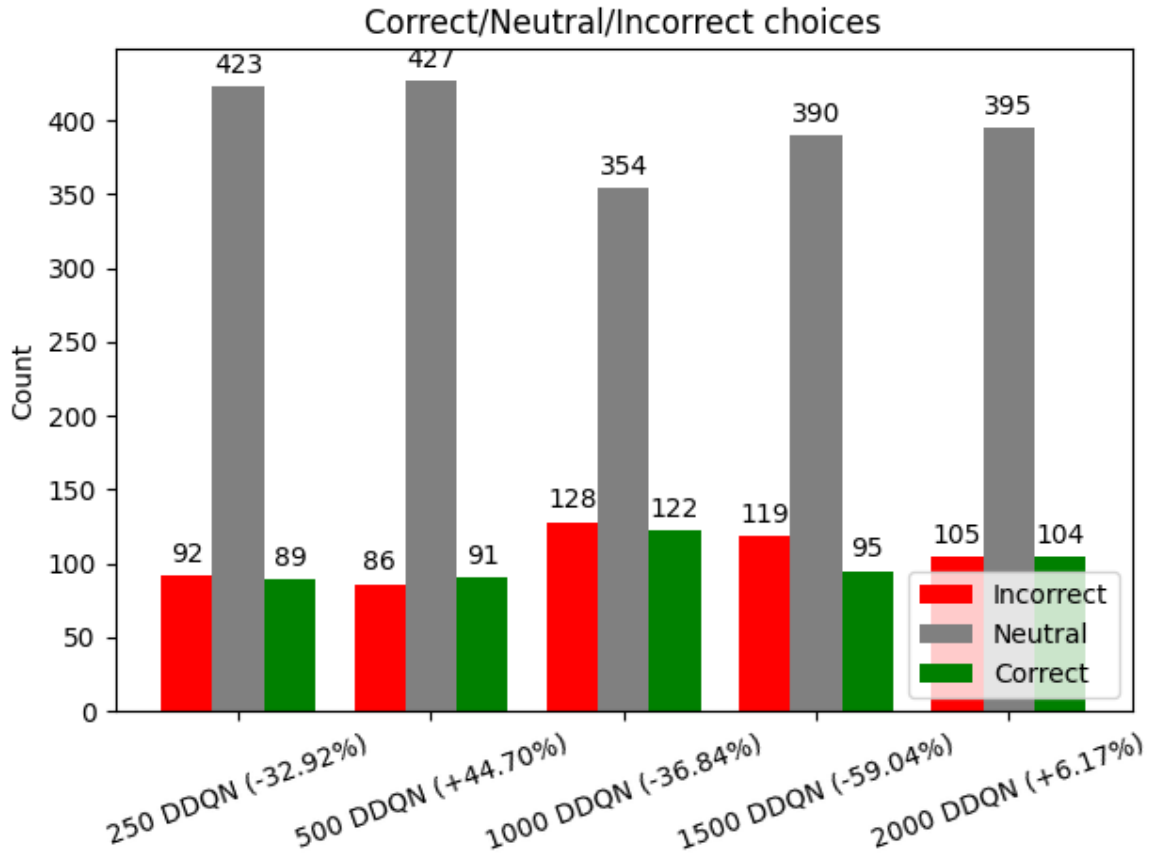


Fig. 33. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. Compared to previous results (without Noise Network), there are more neutral positions. The chart also shows that the return is usually upbeat if the number of correct choices is more significant than the number of incorrect ones.

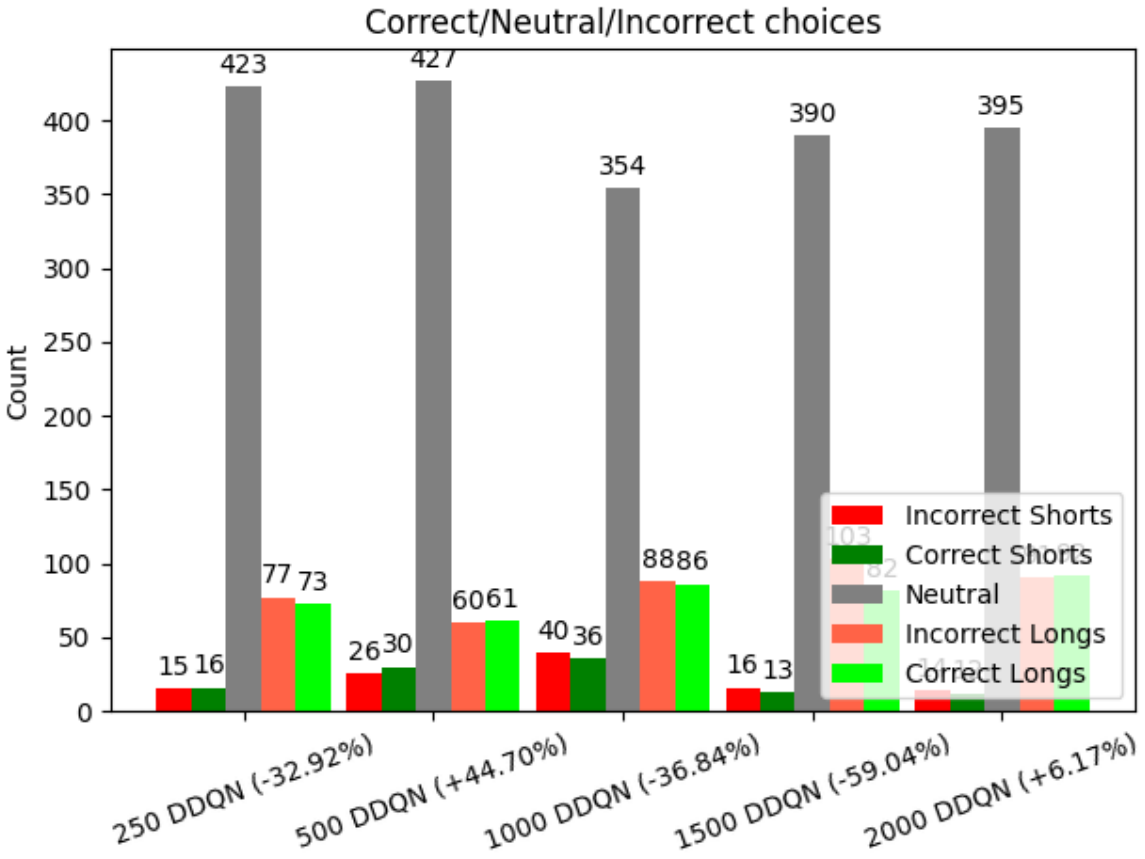


Fig. 34. Five results of agents' games on the stock market and their action correctness distribution (with splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash.

5.5.4 Other simulations

The paper also presents other simulations. Section 5.5.4.1 presents the results of trading cryptocurrency (Bitcoin) during the bear market and INTC stock during the bull market. The main idea was to show how the bot behaves during different market cycles. Section 5.5.4.2 shows the modified neural network structure results that allow additional outputs representing leverage position. In this section, the agent could use leverage position (5xShort, 4xShort, 3xShort, 2xShort, 1xShort, Neutral, 1xLong, 2xLong, 3xLong, 4xLong, 5xLong). Section 5.5.4.3 shows the result of the agent that randomly selects choices (no neural network was there).

5.5.4.1 Cryptocurrency (BTC)

The agent can also trade cryptocurrencies. In this example, an agent with DDQN configuration trading. The selected range of time to test performance was when the price of Bitcoin dropped almost 30% over the half-year. The agent was surprisingly good during the “bear market” (market drop price).

The agent could save the asset value by short positions or holding cash. The following chart shows that the agent can hedge against market losses, even for volatile markets like cryptocurrencies.

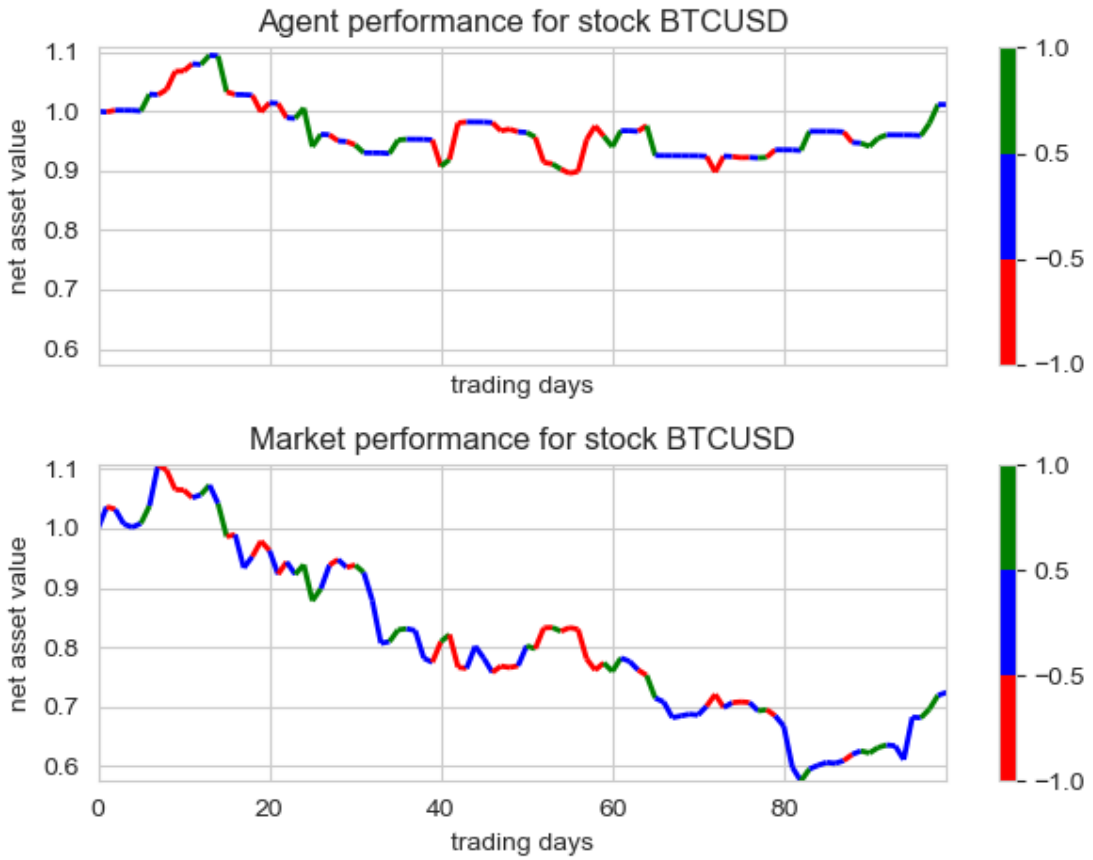


Fig. 35. Agent performance (top chart) compared to the market (bottom chart). Green lines - are periods where the agent keeps a long position, blue lines - are periods where the agent holds cash and red lines - are periods where the agent holds a short position. The data in the following chart was not used to train the network (new data).

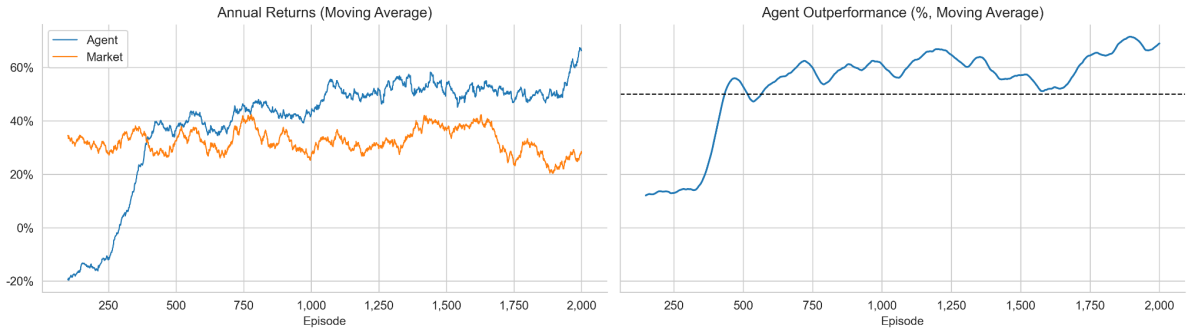


Fig. 36. Agent performance compared to the market (buy-and-hold stock strategy).

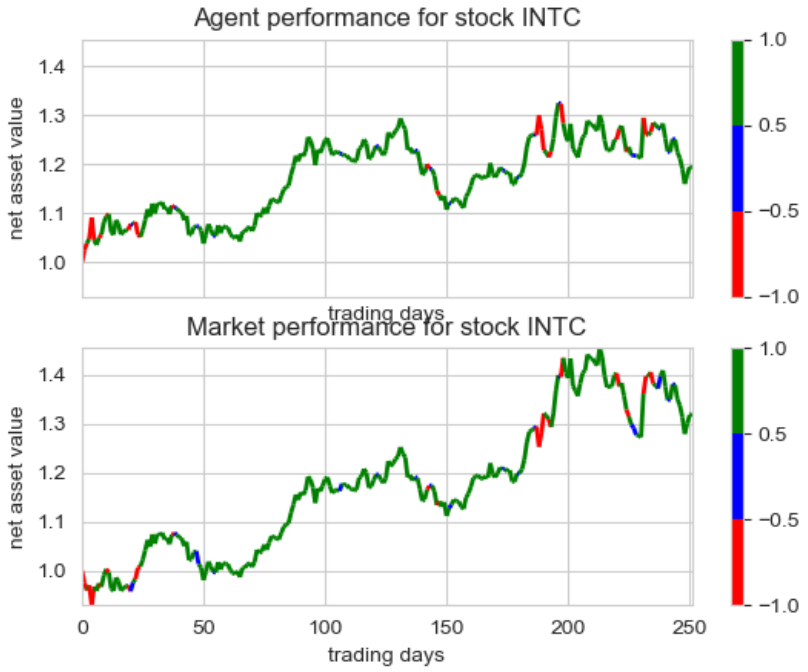


Fig. 37. Agent performance (top chart) compared to the market (bottom chart). Even if the agent performs better on trained data, the new data shows that an agent could not reach the same return as the market (buy-and-hold stock strategy). However, the agent was able to have positive returns.

5.5.4.2 Trading with 5x, 4x, 3x, 2x, 1x leverage (DDQN)

In this simulation, the output of NN was modified. Instead of choosing three actions (Long/Short/Neutral position), more actions were introduced. In trading, there is an option to trade using borrowed funds to leverage position. Using, for example, 100$ own capital, the trader can, with 5x leverage, set a long or short position with 500$. If there is a 500$ long position, a 1% gain is a 5$ gain instead of 1$ without using leverage. However, leverage is a double-edged sword. If there is a 500$ long position, a 1% loss is a 5$ loss instead of 1$ without leverage. In this experiment, 11 outputs were added:

1. 5X Long action
2. 4X Long action
3. 3X Long action
4. 2X Long action
5. 1X Long action
6. Keep Cash action

7. 1X Long action
8. 2X Long action
9. 3X Long action
10. 4X Long action
11. 5X Long action

The results show that the agent loses all capital and is useless to trade using leverage. In this example, the agent performs extremely poorly and loses all the initial capital.

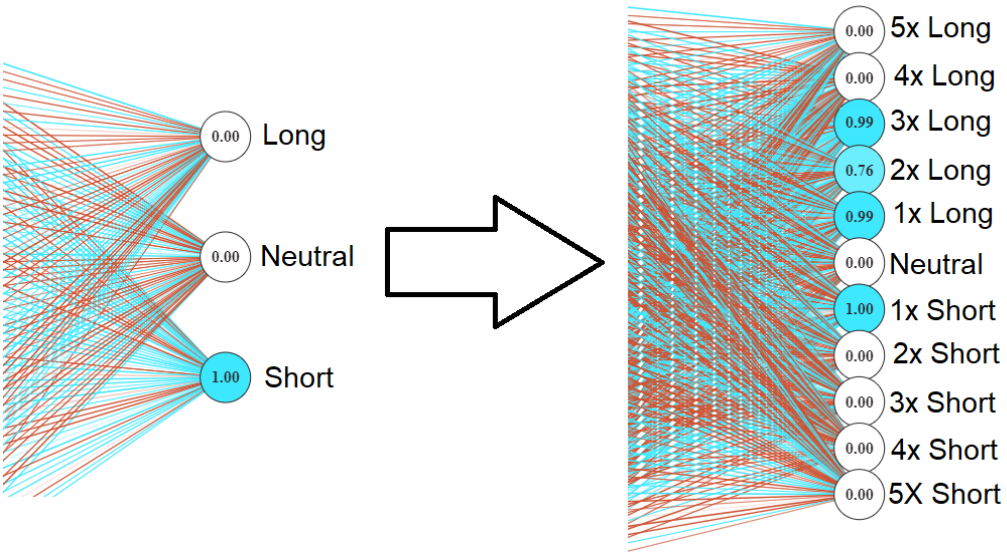


Fig. 38. Modification of NN for leverage trading. Overall the output of NN has 11 outputs - 5 for long positions, 5 for short positions, and 1 for a neutral position (keep the cash).

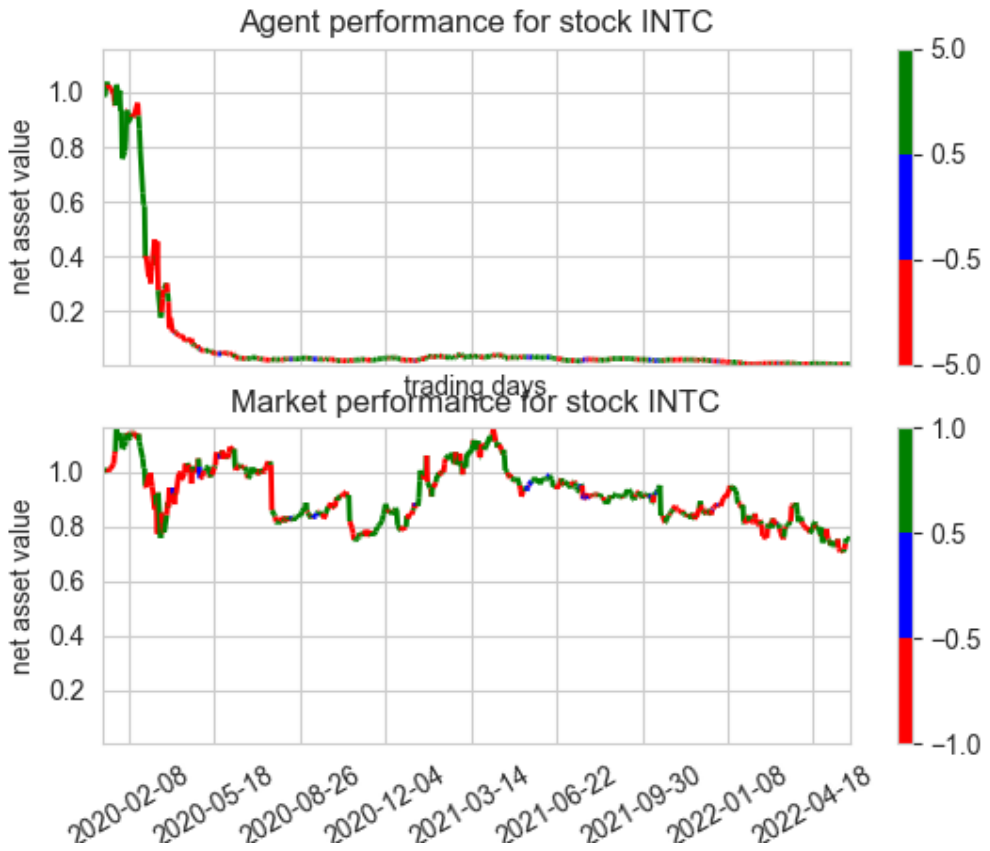


Fig. 39. Agent performance (top chart) compared to the market (bottom chart). The agent loses all NAV. During the trading year, the agent lost 99% of the capital. Trading with leverage is too tricky, and one wrong decision can significantly impact results.

5.5.4.3 Trading bot with random choices

The simulation of the trading bot with random choices can be analyzed to determine how the agent's behavior with random choice compares to DRL choices. As expected, the agent's performance with random choices could be better because it lost nearly 20% of its initial capital.

Fig. 40. Agent performance (top chart) compared to the market (bottom chart). The agent with random choices results in poor performance on the stock market.

6. Conclusion

The above results shows that, in some cases, the agent has good performance and can generate positive returns on the stock market. However, it requires a fine-tuning of the network parameters and learning process. The extension of Double Deep Q-Learing significantly improves the performance of the agent. The noisy network extension can also help improve performance in some cases. The agent works well during the bear market due to the short position. During a great bull market (more than 30%), the agent usually cannot keep up with market returns. However, the returns are still positive. Overall, the fine tunned trading agent can play conservatively - hold the capital during a bear market and provide positive returns during a bull market. In order to tune the agent, multiple backtest, and simulations must occur (with different configurations). Moreover, the agent needs accurate historical data. These findings suggest that the use of reinforcement learning techniques, such as Double Deep Q-Learning, can be a valuable tool for developing trading agents that can generate positive returns in the stock market. Future research can explore further improvements to the agent's performance, such as incorporating more sophisticated strategies or integrating multiple agents.

7. Bibliography

[1] Stefan Jansen. Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python, 2nd Edition. O'Reilly Media, Inc., 2020.

[2] Maxim Lapan. Deep Reinforcement Learning Hands-On - Second Edition. Packt Publishing, 2020.

[3] Miguel Morales. Grokking Deep Reinforcement Learning. Manning Publications, 2020.

[4] KC Tung. TensorFlow 2 Pocket Reference. O'Reilly Media, Inc., 2019.

[5] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv preprint arXiv:1710.02298, 2017.

[6] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. arXiv preprint arXiv:1706.10295, 2017.

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), 2016.

[8] Jiri Pik and Sourav Ghosh. Hands-On Financial Trading with Python: A practical guide to using Zipline and other Python libraries for backtesting trading strategies. Packt Publishing, 2018.

[9] Marcos López de Prado. Advances in Financial Machine Learning. John Wiley & Sons, Inc., 2018.

[10] Sebastien Donadio and Sourav Ghosh. Learn Algorithmic Trading: Build and deploy algorithmic trading systems and strategies using Python and advanced data analysis. Packt Publishing, 2019.

[11] Jeffrey M Bacidore. Algorithmic Trading: A Practitioner's Guide. Wiley Finance, 2013.

[12] Thangjam Ravichandra Singh. A STUDY ON MONTE CARLO SIMULATION FOR STOCK PRICE FORECASTING, Swiss School of Business and Management, 2021

[13] Damir Cavar and Matthew A. Josefy, "Mapping Deep NLP to Knowledge Graphs: An Enhanced Approach to Analyzing Corporate Filings with Regulators," in Proceedings of the 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2018, pp. 272-276.

[14] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. (2014). Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]. Retrieved from https://doi.org/10.48550/arXiv.1406.2661.

[15] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D.N. (2018). StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. arXiv:1710.10916v3 [cs.CV]. Retrieved from https://doi.org/10.48550/arXiv.1710.10916v3.

8. List of Figures

(compared to 50 and 200 SMA). The SMA indicator is a primary trend indicator in technical analysis. Chart from https://www.tradingview.com.

Fig. 11. OHLC chart (top) and Aroon indicators (bottom) of S&P 500 index. The yellow line of the bottom chart represents the Aroon Up indicator which measures uptrend strength. The blue line of the bottom chart represents the Aroon Down indicator which measures downtrend strength. Crossing these lines indicates trend change. Chart from https://www.tradingview.com.

Fig. 12. OHLC chart (top) and Average True Range chart (bottom) of GOOGL (NASDAQ) stock. The chart period from May to August shows increased volatility (pump and dump pattern). Therefore, the ATR in this range is higher. The chart period from August to October shows a continuous price drop. Therefore, the ATR in this range is lower. Chart from https://www.tradingview.com.

Fig. 13. How an agent interacts with the environment. Author: KC Tung [4].

Fig. 14. Simplified example of the stock market system. In the present state, the stock price is after gains. In the future, the price can go up, down, or not change. We set the transition probability to the following specific state using the current state. The left side of the picture presents the present state and available transitions with probability values. The top-Left corner presents a transition stable with all available transitions.

Fix. 15. Transition Probabilities. Author Miguel Morales [3]**.**

Fix. 16. Simple environment where agent starts from state S_1 and has three actions to choose from: move to state S_2, move to state S_3, move to state S_4.

Fix. 17. An expanded environment where the agent starts from state S_1 and has 3 actions to choose: move to the state S_2, move to state S_3, move to state S_4. States S_5, S_6, S_7 are reachable from states S_2, S_3, S_5 respectively.

Fig. 18. Example of action, where the outcome is not determined. There is a 50% probability that action a_2 will cause $S_1 \rightarrow S_2$transition and a 50% probability that the same action will cause $S_1 \rightarrow S_3$ transition.

Fix. 19. Raw asset price DataFrame from Financial Modeling Prep API.

Fix. 20. During the Dot-Com Bubble Amazon stock dropped approximately 80%. In absolute price, it was ~3 USD (price adjusted). Today, minor corrections like ~15% can cause a price drop of ~20 USD. This example shows how inaccurate absolute values are.

Fig. 21. Preprocessed asset data by TA-Lib.

Fig. 22. ReLU activation function. Image adapted from https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/.

Fig. 23. Visualisation of neural network used in the trading bot.

Fig. 24. Visualisation of neural network in DQN configuration.

Fig. 25. Visualisation of neural network in DDQN configuration. The online network is used to select the best action in a given state based on its current estimates of Q-values. During each training iteration, the online network is updated using a stochastic gradient descent algorithm to minimize the difference between its predicted Q-values and the target Q-values. The target network, on the other hand, is used to estimate the target Q-values that are used in the Q-learning update.

Fig. 26. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

Fig. 27. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The incorrect choice is the position (short or long) that results in a negative return (e.g., a short position when the stock price went up or a long position when the stock price went down). In reverse is the correct choice (e.g., short position when the stock price went down or long position when the stock price went up). The neutral choice is when the agent holds the cash (does set any position).

Fig. 28. Five results of agents' games on the stock market and their action correctness distribution (splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The chart shows incorrect/correct short/long positions and neutral positions.

Fig. 29. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

Fig. 30. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. The result with the lowest neutral decisions performed the worst (2000 DDQN).

Fig. 31. Five results of agents' games on the stock market and their action correctness distribution (with splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. The result with the lowest neutral decisions performed the worst (2000 DDQN).

Fig. 32. Five results of agent games on the stock market and market performance (blue line). The agent was trading Intel stock (INTC) from 2020-01-01 to 2022-06-01.

Fig. 33. Five results of agents' games on the stock market and their action correctness distribution. The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash. During that time, the stock price dropped by 24.67%. Compared to previous results (without Noise Network), there are more neutral positions. The chart also shows that the return is usually upbeat if the number of correct choices is more significant than the number of incorrect ones.

Fig. 34. Five results of agents' games on the stock market and their action correctness distribution (with splitting on correct/incorrect short/long positions). The agents were trading Intel stock (INTC) from 2020-01-01 to 2022-06-01. The bot was allowed to short or long positions or to keep the cash.

Fig. 35. Agent performance (top chart) compared to the market (bottom chart). Green lines - are periods where the agent keeps a long position, blue lines - are periods where the agent holds cash and red lines - are periods where the agent holds a short position. The data in the following chart was not used to train the network (new data).

Fig. 36. Agent performance compared to the market (buy-and-hold stock strategy).

Fig. 37. Agent performance (top chart) compared to the market (bottom chart). Even if the agent performs better on trained data, the new data shows that an agent could not reach the same return as the market (buy-and-hold stock strategy). However, the agent was able to have positive returns.

Fig. 38. Modification of NN for leverage trading. Overall the output of NN has 11 outputs - 5 for long positions, 5 for short positions, and 1 for a neutral position (keep the cash).

Fig. 39. Agent performance (top chart) compared to the market (bottom chart). The agent loses all NAV. During the trading year, the agent lost 99% of the capital. Trading with leverage is too tricky, and one wrong decision can significantly impact results.

Fig. 40. Agent performance (top chart) compared to the market (bottom chart). The agent with random choices results in poor performance on the stock market.