UNIVERSITY OF TURKU
Turku School of Economics

Algorithmic trading on Finnish stock market using Deep Reinforcement Learning

Master's thesis
in Accounting and Finance

Author:
Saku Korpas

Supervisor:
Prof. Luis Alvarez Esteban

24.3.2024
Turku

Master's thesis

Subject: Accounting and Finance
Author: Saku Korpas
Title: Algorithmic trading on Finnish stock market using Deep Reinforcement Learning
Supervisor: Prof. Luis Alvarez Esteban
Number of pages: 87 pages + appendices 2 pages
Date: 24.3.2024

Abstract

The advancement in machine learning due to increased computational capacity and novel algorithms have resulted in copious amount of research done on financial markets where machine learning is harnessed for stock market trading. The promising results of deep learning in many fields have encouraged researchers and practitioners to utilize this novel technique to come up with ground-breaking algorithmic trading strategies that could consistently generate excessive returns. The merits of deep learning are extensive, such as its ability for feature learning, scalability, flexibility, and adaptability, which all are relevant features when considering financial markets. One of the deep learning approaches is deep reinforcement learning, which trains the algorithm by giving feedback for its actions in given environment. The algorithm aims to maximize the feedback it receives, thus convergencing to an optimal trading strategy.

This study aims to come up with an algorithmic trading strategy that can consistently outperform benchmark strategies by using Deep Q-Networks, a type of deep reinforcement learning. Additionally, the effect of introducing dynamic stop loss and take profit levels in feedback mechanism is studied. The research is conducted between the beginning of 2022 and the end of 2023 on three individual stocks on Finnish stock market and a broader market index. The information provided for the deep neural network are daily open price, lowest and highest price of the day, closing price, and trading volume of the day. The model performance is evaluated, such as its ability to learn over time, and ultimately the proposed trading strategy is benchmarked against other trading strategies.

The model performance analysis suggested that the complexity of stock market leads to large variations in model practicality. Although some improvement was detected where the average reward and Sharpe ratio increased over time, most circumstances indicated high fluctuation and randomness in the model. The proposed trading strategy did outperform benchmark strategies in multiple simulations, but also underperformed considerably in many other scenarios. Therefore, the results show that this strategy can not be used to consistently outperform the benchmark. Furthermore, the use of stop loss and take profit limits to guide the trading algorithm towards optimal trading policy was studied. Contrary to guiding the agent, the trading boundaries further contributed to the complexity of trading environment making the model performance more volatile. This was since crossing stop loss or take profit level triggered relatively large on-off reward which complicated the trading environment to a greater degree. The algorithm was able to generate higher average rewards, but at the cost of stability. Therefore, the application of trading limits into the feedback function did not enhance the performance of the deep q-network strategy.

Key words: deep learning, deep neural network, algorithmic trading, deep reinforcement learning, deep q-networks, stock trading

Pro Gradu -tutkielma

Oppiaine: Laskentatoimi ja rahoitus
Tekijä: Saku Korpas
Otsikko: Algoritminen kaupankäynti Suomen osakemarkkinoilla syvän vahvistusoppimisen avulla
Ohjaaja: Prof. Luis Alvarez Esteban
Sivumäärä: 87 pages + appendices 2 pages
Päivämäärä: 24.3.2024

Tiivistelmä

Koneoppimisen kehitys lisääntyneen laskentatehon ja uusien algoritmien ansiosta on johtanut runsaaseen määrään tutkimuksia, joissa koneoppimista hyödynnetään osakemarkkinoiden kaupankäynnissä. Syväoppimisen positiiviset tulokset monilla aloilla ovat rohkaisseet tutkijoita ja ammattilaisia hyödyntämään tätä uutta tekniikkaa kehittääkseen uusia algoritmisia kaupankäyntistrategioita, jotka kykenevät johdonmukaisesti tuottamaan ylituottoa. Syväoppimisen hyödyt ovat laajat, kuten sen kyky ominaisuuksien oppimiseen, skaalautuvuus, joustavuus ja sopeutumiskyky, jotka kaikki ovat olennaisia ominaisuuksia finanssimarkkinoiden kannalta. Yksi syväoppimisen lähestymistavoista on syvävahvistusoppiminen, jossa algoritmia koulutetaan antamalla palautetta sen toiminnasta tietyssä ympäristössä. Algoritmi pyrkii maksimoimaan saamansa palautteen ja siten löytämään optimaalisen kaupankäyntistrategian.

Tämä tutkimus pyrkii kehittämään algoritmisen kaupankäyntistrategian, joka voi johdonmukaisesti suoriutua paremmin kuin vertailustrategiat käyttämällä syviä Q-verkkoja, joka on yksi syvävahvistusoppimisen menetelmistä. Lisäksi tutkielmassa tutkitaan dynaamisien stop loss ja take profit -tasojen käytön vaikutusta palautemekanismissa. Tutkimus tehdään vuoden 2022 alusta vuoden 2023 loppuun kolmella yksittäisellä osakkeella Suomen osakemarkkinoilla sekä laajemmalla markkinaindeksillä. Syvälle neuroniverkolle annettu tieto sisältää päivittäisen avauskurssin, päivän alimman ja korkeimman hinnan, päätöskurssin ja päivän kaupankäyntivolyymit. Mallin suorituskykyä arvioidaan, kuten sen kykyä oppia ajan myötä, ja lopulta kehitettyä kaupankäyntistrategiaa vertaillaan muihin kaupankäyntistrategioihin.

Mallin suorituskyvyn analysointi osoittaa, että osakemarkkinoiden monimutkaisuus johtaa suuriin vaihteluihin mallin toimivuudessa. Vaikka kehitystä havaittiin, kun keskimääräinen palkkio ja Sharpe-luku kasvoivat ajan myötä, useimmissa tilanteissa mallissa ilmeni kuitenkin suurta vaihtelua ja satunnaisuutta. Kehitetty kaupankäyntistrategia suoriutui paremmin kuin vertailustrategiat useissa eri simuloinneissa, mutta myös alisuoriutui merkittävästi monissa muissa skenaarioissa. Tulokset osoittavat, että tätä strategiaa ei voida käyttää johdonmukaisesti tuottamaan ylituottoja verrattuna vertailustrategioihin. Lisäksi stop loss ja take profit -rajatasojen käyttöä algoritmin tueksi tutkittiin. Päinvastoin kuin että rajatasot olisivat ohjanneet algoritmia, kaupankäyntirajojen käyttö lisäsi kaupankäyntiympäristön monimutkaisuutta, mikä teki mallin suorituskyvystä entistä epävakaampaa. Tämä johtui siitä, että stop loss tai take profit -tason ylittäminen laukaisi suhteellisen suuren kertaluonteisen palkkion, joka monimutkaisti kaupankäyntiympäristöä entisestään. Algoritmi pystyi tuottamaan korkeampia keskimääräisiä palkkioita, mutta tämä heikensi mallin vakautta. Näin ollen kaupankäynnin rajojen soveltaminen palautemekanismiin ei parantanut syvää Q-verkkoa hyödyntävän strategian suorituskykyä.

Avainsanat: syväoppiminen, syvä neuroverkko, algoritminen kaupankäynti, syvä vahvistusoppiminen, syvä q-verkosto, osakekaupankäynti

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

1.1 Background

Financial markets of the 21st century are marked by an unprecedented influx of data and advances in technology. The rise of algorithmic trading (AT), an automated and rule-based trading method, has been a significant innovation to the industry, creating automation and efficiency by providing liquidity and faster trade executions. Nowadays, algorithms are used by traders and institutions to execute trades, manage portfolios, and optimize market interactions. Due to the fact that more data, in other terms, more information, is available now than ever before, the quest for effective AT strategies using this wide range of data has evolved. Many practitioners and researchers search for optimal AT strategies that could generate excess returns, but the financial market's nature of dynamic and unpredictability makes the task challenging.

Although the last decades have undergone rapid advancement in financial markets and some of challenges are resolved, new challenges have emerged alongside the past problems. Classical trading algorithms face variety of difficulties. Solutions that are strictly rule-based often lack the flexibility required to adapt to changing market conditions. This can result in adverse trading strategies or missed opportunities. Traditional models, although capable of learning some patterns from historical data, may struggle to capture the non-linear relationships and sudden shifts inherent to financial markets.

In response to the complexity of market dynamics, novel techniques using machine learning have emerged due to advanced computational power and modern algorithms. These techniques have then been integrated to find effective trading strategies, especially in form of AT. Deep learning, a subfield of machine learning, has been applied on many fields due to its ability to analyze vast amounts of historical data and make data-driven decisions in real-time, but especially because of its ability to further learn and develop its model. The financial market is particularly suitable environment for these algorithms as the previous methods have struggled with curse of dimensionality and non-linearity.

One of these novel methods is deep reinforcement learning (DRL) which has emerged as an approach for AT. This method offers promising results in the face of challenges presented by the nature of financial markets. DRL represents a combination of deep learning which consists of neural network, and reinforcement learning in which the algorithm learns by rewarding the positive actions and penalizing negative actions. The adaptability of DRL makes it well-suited for financial markets as it is able to capture complex and non-linear dynamics intrinsic to financial markets. Unlike traditional methods, which often rely on predefined rules and assumptions, DRL enables the model to learn from experience and environment, and adapt to changing market conditions.

Among the different DRL algorithms, Deep Q-Network (DQN) has been widely studied. DQN combines deep learning with Q-learning, a type of reinforcement learning. In early developments, DQN was mostly designed to play video games (Mnih et al., 2015). The algorithm provided excellent results, even succeeding human competitors. Later, in the context of financial markets, DQN has been used in stock price prediction and forecasting (Shah et al. (2020); Zhang et al. (2022)), portfolio optimization and management (Gao et al. (2020); Lucarelli and Borrotti (2020); Niu et al. (2022)), automated asset trading and trading strategies (Chen and Gao (2019); Park et al. (2020); Cartea et al. (2021)), and risk management (Clements et al., 2020).

While there have been numerous applications and studies on applying DQN for stock trading, it appears that the specific approach involving trading limits for trading agent has seen limited research. Kim and Kim (2019) used trading and stop loss boundaries in pairs-trading, which they later developed further (Kim et al., 2022), and Wen et al. (2021) used a fixed stop loss level, but other than that, the research on the topic seems limited. This study will add to the research by studying how using dynamic stop loss and take profit levels in reward function affect the performance of the trading strategy of DQN. The idea of stop loss and take profit limits in this study is to further guide the agent by providing additional feedback when a single trade crosses one of these thresholds. The agent receives extra penalty if the closing price crosses the stop loss level, whereas extra reward is given when the closing price crosses take profit level. While the boundaries give additional feedback to the agent, the basis that the agent is rewarded on daily returns remains the principal reward. To consider that the volatility, changes in time, the thresholds are set dynamically using Average True Range over past 14 trading days. The performance of this proposed approach is evaluated against Trading Deep Q-Network (TDQN) by Théate and Ernst (2021), which serves as a basis for this approach as well, and against other traditional trading strategies.

Furthermore, despite the extensive research in global financial markets, the application of DQN in the Finnish stock market seems to be unexplored. While the technique used is relatively generic, meaning that the models can be implemented on various assets, the unique characteristics of the Finnish stock market, such as its size and liquidity, may present distinct results and challenges. Investigating the applicability of DQN on the Finnish stock market could lead to a more tailored strategies that consider the specific nuances of this market.

1.2 Research questions and objective

This thesis compares the performance of a modified DQN algorithm that includes stop loss and take profit levels with TDQN algorithm originally created by Théate and Ernst (2021), and with other traditional trading strategies on Finnish stock market to examine whether this novel technique can achieve excess returns. Deep learning is an emerging field that can overcome constraints that have existed before, which provides a reason for the recent extended research done on the topic. Although DQN has been broadly researched and there exists wide range of different approaches, it seems that using trading limits has not been extensively researched. This study will address this by creating a DQN algorithm with novel reward function that considers the trading boundaries. In addition, focusing on Finnish stock market contributes to the further research of applying deep reinforcement learning to financial markets within a specific context. The main research question of this thesis are as follows:

1. Is the proposed DQN algorithm able to consistently generate excess returns against benchmark strategies on Finnish stock market?
2. What is the impact of introducing trading limits within the reward function on the proposed DQN algorithm?

The analysis is conducted individually on three stocks (Nordea, Sampo, Bittium) and OMX Helsinki PI index. By selecting these four different environments, the performance of the DQN algorithm can be evaluated within varying conditions: individual stocks have their own characteristics while OMX Helsinki PI represents broader market portfolio. The goal of the study is to examine whether DQN can be used to learn past price patterns and exploit the experience to predict subsequent prices for excess returns. To evaluate the performance, different ratios are calculated for each strategy to evaluate returns, risk-adjusted returns, and other quantitative metrics of the strategies. The results are benchmarked against Buy and Hold (B&H), Trend following with moving averages (TF), Mean reversion with moving averages (MR), and original TDQN by Théate and Ernst (2021). Additionally, for a comparison purpose, a random trading strategy is used where random buy or sell actions are taken.

1.3 Structure of the thesis

The thesis structure is as follows. Section two provides theoretical background on relevant topics related to the study. First, an overview on stock market predictability is established, including two prominent theories on the topic: Efficient Market Hypothesis and Random Walk Theory. Then algorithmic trading and its implications for finance are explained briefly. Third, basics of reinforcement learning, deep learning and DQN are described. In addition, advantages and challenges of applying DQN on stock trading is presented. In the last subsection of theoretical framework, literature on how DQN is used in stock trading is reviewed, serving as a literature review on the subject. Section three will focus on the research data and methodology. The proposed approach together with DQN algorithm architecture and configuration are explained. In section four, the analysis is conducted and the results are presented. Evaluation metrics for the DQN are calculated and analyzed, after which the performance of modified DQN is benchmarked against other trading strategies. Finally, the fifth section will conclude the research and bring up discussion and further research topics in the field.

2 THEORETICAL FRAMEWORK

2.1 Predictability of stock market

Predicting stock market has long been a subject of scrutiny for investors, economists, and researchers. The hunt for optimal prediction strategy is understandable: given that one is able to foretell the future, one can make huge profits using this information. While it is not entirely unequivocal whether stock market can be successfully predicted, many studies stand behind this possibility. Studies have indicated negative correlation between inflation rate and stock market returns, suggesting that inflation rate could be used to predict stock market returns (Bodie (1976); Jaffe and Mandelker (1976); Lintner (1975); Nelson (1976)). Later Fama (1981) extended the analysis to measures of real activity such as capital expenditures and the average real rate of return on capital rather than just inflation. Fama (1981) cited that stock returns are influenced by forecasts of more relevant real variables, and negative correlation between stock returns and inflation is a result of negative relations between inflation and real activity. Research made by French et al. (1987) studied relationship between volatility and expected risk premiums, and positive correlation between the two variables was found.

Pesaran and Timmermann (1995) note that while many studies have been in support of stock market predictability using economic variables (such as interest rate, monetary growth rate, and inflation rates), the economic interpretation of the results remain controversial and far from evident. Pesaran and Timmermann (1995) conclude that support for the importance of predictable components in stock returns can be found, but the magnitude of predictability to be economically exploited seems depended on both business cycle and magnitude of shocks in stock markets. They also find that there is correlation between volatility and predictability of excess returns in stock market. During low volatility, it was found to be harder to generate excess returns and vice versa.

Other factor influencing the predictability is time horizon of prediction. Time horizon refers to the duration over which investors attempt to forecast the future movements of stock prices. Fama and French (1988a) discover that forecasting power increased with the return horizon. The explanatory power of the regressions grew when longer periods were considered compared to shorter periods. Fama and French (1988a) explain this to be result of "the variance of the fitted values growing more quickly than the horizon, whereas the variance of the residuals generally grows less quickly than the horizon". Fama and French (1993) suggest that over the long term, fundamental factors like company size and valuation metrics significantly impact stock returns. In favor of long-term stock predicting, Jegadeesh and Titman (1993) find that momentum plays a role in returns. In the study, it was found that buying stocks that have performed well in the past generate significant positive returns over 3- to 12-month holding periods. Ang and Bekaert (2007) however state that excess return predictability by the dividend yield is not statistically significant at long horizons, but find stock returns predictable on short horizons.

Efficient Market Hypothesis (EMH) and Random Walk Theory, both fundamental finance theories and cornerstones of finance research, challenge stock market predictability. These relevant theories are introduced in the following sections 2.1.1 and 2.1.2. Overall, research has shown both evidence for and against stock market predictability. The challenges of accurately predicting the complex and dynamic nature of financial markets has led to different approaches in this field. The search for a foolproof forecasting method continues to be an ongoing exploration, if there exists one.

2.1.1 Efficient Market Hypothesis

Efficient Market Hypothesis (EMH) is one of the foundational theories in financial theory that provides a framework for understanding the role of information in stock market. The origin of the idea that financial market returns are difficult to predict lies in research done by researchers such as Bachelier (1900), Mandelbrot (1966), and Samuelson (1965), but was later formalized in 1970 by Fama. According to EMH, financial markets are efficient, meaning that security prices fully reflect all available information at any time. Any new information announced will be valued in security prices immediately.

EMH classifies market efficiency in three forms depending on what information is considered: weak, semi-strong and strong form efficiency. Weak form efficiency includes only information about historical prices of securities. This means that using historical patterns or other technical analysis of past prices for generating abnormal returns is futile. The weak form tests are closely related to Random Walk Theory that is presented more thoroughly in Section 2.1.2. Semi-strong form efficiency acknowledges other publicly available information like announcements of annual reports in addition to historical prices. As a result, fundamental analysis in which financial statements and economic indicators are studied to predict undervalued or overvalued securities, is expected to offer no advantage. In strong form efficiency, information set is expanded to whether all available information is fully reflected in security prices. This form focuses that no investor has monopolistic access to some information that would result in higher expected returns. The strongest form of market efficiency leaves no room for any type of information-based advantage. Fama (1970) gives an example of situation whether a manager of mutual funds seem to have access to special information to generate abnormal returns.

Given this hypothesis, it is difficult to gain excess or consistently above-average returns through stock trading due to information efficiency of financial markets. Firstly, as new information becomes available, it is quickly reflected in security prices. This means investors have little to none opportunity to exploit information lags. Secondly, financial markets exhibit a competitive environment. The presence of many sophisticated participants makes it difficult for single investor to consistently gain an information advantage. Thirdly, not only all available information is priced in security prices, it is also priced in efficiently. Any potential inefficiencies are quickly corrected by market participants. Fourthly, financial markets are adaptive. If one would find a trading strategy that consistently yields excess returns, other market participants would adopt similar strategies which would lead to convergence of returns and thus making the strategy non-profitable.

EMH provides an important theoretical foundation on efficiency of financial markets, but has faced criticism also. Fama (1970) states that there is no important evidence against weak and semi-strong form efficiency, and only limited evidence for strong form efficiency. Later Fama (1991) acknowledged that the extreme form of efficiency where all available information is reflected in security prices does not hold in financial markets, but EMH still works as a theoretical framework for approximating the level of available information on the market, and how deep the EMH reaches. In the same study, Fama (1970) raises the joint-hypothesis problem in market efficiency where both the efficiency of the market and the validity of a specific asset pricing model is simultaneously tested. These two hypotheses are contradictory to each other as the result of a such test is unambiguous: what part is the market inefficient and what part is it due to bad model of market equilibrium. Therefore, market efficiency can not be tested per se, making it difficult to directly accept or reject the theory.

For long, the idea of EMH that news spread very quickly and are reflected into stock prices immediately remained well accepted among academic financial economists. As a result, technical analysis nor fundamental analysis would achieve higher returns than randomly selecting stocks. By the start of twenty-first century, the dominance of EMH had become less universal, making room for thought that stock market is at least somewhat predictable. New research rose economists to speculate that psychological and behavioral elements, past price patterns, and fundamental valuation metrics could be utilized for stock market prediction, or even for excess returns. (Malkiel, 2003). This notion was encouraged by works on short-term prediction like Lo and MacKinlay (1999) in which consecutive moves to the same direction broke weak form efficiency, and Lo et al. (2000), who found pattern recognition to offer modest predictive power. On long-term prediction, Fama and French (1988b) found that 25 to 40 percent of the variation can be predicted with negative correlation with past returns, and Poterba and Summers (1988) similarly found support for this mean reversion in stock market returns. This could be due to tendency of overreaction in stock market prices in which optimism overweights the pessimism which fluctuates prices from their fundamental values and later return to mean (De Bondt and Thaler, 1985). Additionally, support on seasonal and day-of-the-week abnormal returns have been conducted (French (1980); Keim (1983)), and in some cases fundamental metrics have seem to explain significant part of the variance of future returns (Nicholson (1960); Campbell and Shiller (1998)).

Despite positive results for predictability, whether the strategies are economically significant or if the predictability stays consistent over time is uncertain. Additionally, the finding that many predictable patterns disappear quickly after publishing, hinder the actual execution of these strategies. (Malkiel, 2003).

When consideration of psychological factors gained popularity, it created the behavioral finance discipline. The area studies the influence of psychological factors of investors on financial decisions. Study made by Kahneman and Tversky (1979) found several psychological factors governing in behavioral economics. Some of the key findings were that people reason their actions based on the potential value of losses and gains rather than the final outcome, people are more sensitive to losses than to gains (loss aversion), and people tend to overweight small probabilities and underweight large probabilities (the certainty effect). Shiller (2000) argues that at times, market valuations are driven more by psychological factors, such as investor emotions and herd mentality, rather than fundamental economic principles. These psychological factors may create inefficiencies to financial markets and therefore challenges EMH.

Another challenging approach to EMH is noise trading. First introduced by Black (1986), the noise trading refers to trading activity of investors who are not well-informed about the value of the securities they trade. This noise can introduce randomness and inefficiency into market as security prices may deviate from their fundamental values. However, noise creates uncertainty and risk for both informed and uninformed investors, preventing informed investors from exploiting and taking advantage of uninformed investors. Noise makes it difficult to distinguish between changes in prices that reflect changes in fundamental value and changes that reflect noise. De Long et al. (1990) showed how uninformed traders with erroneous beliefs can influence prices and actually earn higher returns than rational informed traders. De Long et al. (1990) also note that noise trading can create excess volatility, mean reversion, and other anomalies in financial markets.

2.1.2 Random Walk Theory

Random Walk Theory suggest that stock market prices follow so-called "random walk", meaning that stock prices exhibit a random and unpredictable pattern. According to this theory, past price movements or patterns are not reliable indicators of future price movements. Random Walk Theory is closely associated with EMH as the idea of random walk is that if information is immediately reflected in stock prices, then tomorrows' news will reflect only tomorrow's price change, thus being independent of price changes today. Because of news being unpredictable, the price changes must be unpredictable and random. Given the unpredictability, both uninformed and informed investors will obtain similar returns. (Malkiel, 2003).

The concept of Random Walk Theory date back to 1863, when Jules Regnault published the book "Calcul des Chances et Philosophie de la Bourse", suggesting the first modern theory of random walk of stock prices by using a random walk model (Jovanovic and Le Gall, 2001). Later Bachelier (1900) was the first to model stochastic process ("random fluctuation"). The theory gained significant attention in the academics in 1960s and 1970s along with EMH. Samuelson (1965), Fama (1965a,b), Cootner (1967) further contributed to support the idea that stock market follows random walk. Fama (1965a) notes that according to Random Walk Theory, consecutive price changes are independent and identically distributed, and the series of price changes has no memory, thus the past prices give no indication of future in any meaningful way. In 1973, Malkiel published a book "A Random Walk Down Wall Street" where he argues that trying to time the market or pick individual stocks based on technical or fundamental analysis is similar to taking a random walk and is unlikely to result in consistent outperformance. The approach advocates for passive holding strategies over actively managed strategies.

If Random Walk Theory holds correct, forecasting random future stock prices would be of no real value. While the evidence leans in the favor of Random Walk Hypothesis, the opposite results have been obtained. Lo and MacKinlay (1988) tested the random walk hypothesis on sample period of 1962-1985 US stock market, rejecting the entire sample period and all subperiods. This supports the circumstance that stock returns can be predictable to some extent. Lo and MacKinlay (1999) later published a book "A Non-Random Walk Down Wall Street", clearly referencing to works of Malkiel, where they discuss topics such as market anomalies, technical analysis, and behavioral finance.

In defence of rejecting Random Walk Theory, Fama and French (1988b) recorded that long return horizons, in this case beyond a year, showed strong correlation between time series and its lagged values. For shorter periods like daily and weekly holding, the autocorrelation remained weak.

2.2 Algorithmic trading and its implications for finance

Algorithmic trading has emerged as a dominant force in the financial markets beginning in 1990s. AT involves the use of computer programs to automate the trading process, from market analysis and decision-making to order execution. In AT, algorithms analyze data to detect and exploit temporary trading opportunities on a set of rules. Although AT is often associated with high-frequency trading (HFT), in which a large number of market orders are executed at extremely high speeds, these two are not identical. HFT is a subset of AT, but AT does not necessary have to execute orders at high speeds. The advantage that computers bring over humans is speed; having advantage in both processing information and acting on information. However, there is also difference in sense that computers are more prune to higher correlation in trading actions as preprogrammed algorithms may react similarly to a given signal. (Chaboud et al., 2014). AT is used for variety of purposes by many different types of market participants. Quantitative fund managers use it to determine portfolio holdings and formulate trading strategies, hedge funds and broker-dealers use it to supply liquidity, and statistical arbitrage funds to trade on patterns in data. (Hendershott et al., 2011).

The practise has received both endorsement and backlash. As HFT require computers that are not generally available to large part of investors, it has attracted controversy and questions of fairness. On the other hard, it has led to declined transaction costs and bid-ask spread, and increased liquidity. (Goldstein et al., 2014). It is also seen that by using speed advantage over humans, AT have a positive impact on informativeness of prices as algorithmic traders use market orders to exploit their information (Biais et al., 2011). Chaboud et al. (2014) arrive to similar results, indicating that AT improves informational efficiency by speeding up price discovery. They note that AT may also inflict higher adverse selection costs for slower traders due to HFT front-running market orders. Brogaard et al. (2014) conducted an analysis on how HFT impact price discovery and price efficiency. Results were that HFTs increase efficiency of prices as they trade in the direction of permanent price changes and opposite direction of transitory pricing errors. It was also noted that no evident was found for HFTs contributing to market instability between years 2008 and 2009. A controversial topic among the finance industry was whether HFT was responsible for the 2010 "Flash Crash" in which E-mini S&P 500 stock index experienced a rapid and extreme drop in prices, followed by a swift recovery, all within a matter of minutes. Later Kirilenko et al. (2017) studied the involvement of HFTs in the crash, and found that HFTs "did not take on large risky inventories relative to the large and temporary selling pressure".

The recent advancements in artificial intelligence have impacted AT industry radically. Integrating artificial intelligence into algorithmic trading has allowed for processing increased amounts of data, learning patterns to a greater degree, and enhancing predictive modeling of trading algorithms. Whereas more traditional AT require pre-programmed rules, AI-driven models can analyze historical data to identify intricate patterns, correlations and anomalies. By continuously learning from previous analyses, it allows the algorithms to adapt to changing conditions.

2.3 Reinforcement learning

2.3.1 Basics of reinforcement learning

Sutton et al. (1998) introduced reinforcement learning, a self-taught process where an agent seeks to maximize the total reward it receives when interacting within a given environment. By this trial-and-error process, the agent progressively learns optimal way to maximize the reward function, an objective feedback from environment. Following this, an optimal policy for that specific environment can be formed.

Basic concepts of reinforcement learning include agent, environment, state, action, reward, agent's policy, and value function. The concepts of many of these are very general. An agent is an entity that interacts with its environment, action is any decision made by agent, and state is any factor that the agent considers when taking an action. A policy or a policy function is a mapping from possible states to possible actions. A policy can be a stored policy such as a simple look-up table, or computed policy where an action sequences that produce the greatest value is searched using a model of the environment through an ever-changing tree of values. A value function or reward function evaluates the desirability of a state (or state-action pair), providing a measure of the expected cumulative reward that the agent can achieve from that point forward. This function is used to estimate the long-term consequences of agent's actions, helping the agent to make informed decisions. The value function is constantly updated based on the experienced outcomes of agent's actions. A key factor of the value function is the time discounting in which future rewards are discounted so that rewards further in the future should be of less value. (Sutton et al., 1998).

According to Sutton et al. (1998), one way to represent this learning is Markov Decision Process (MDP). Carta et al. (2021) constitute this process with a 4-tuple (S, A, P_a, R_a) where:

- S is a set of states;
- A is a set of actions;
- $P_a(s_t, s_{t+1})$ is the probability that an action a in state s_t will lead to the state s_{t+1}. States s_t and s_{t+1} originate from the environment, according to the action a done by the agent;
- $R_a(s_t, s_{t+1})$ is the immediate expected reward received in the transition from a state s_t to a state s_{t+1} by executing action a.

The goal of the learning policy π to return maximal reward can be formulated mathematically as

$$\text{argmax}_{\pi}\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R_{\pi(s_t)}(s_t, s_{t+1})\right],$$

where $\pi(s_t)$ is the action a_t defined by a policy π in a given state s_t, and γ is the discount factor.

Sutton et al. (1998) analyze three different types of reinforcement learning methods. First, in dynamic programming approach, a set of algorithms may be used to compute the optimal policy by giving a complete model of environment as a MDP. These algorithms use a step by step approach by updating of value functions incrementally to search for good policies that maximize rewards. To find a good policy, an action in the environment is executed, and following this, the value of the state before the action is reassessed by considering the values of all potential states reachable from the preceding state, taking into account their respective probabilities of attainment. In real world, dynamic programming faces some problems. The assumption that the complete model of environment is available is often unrealistic and for larger tasks, the algorithm quickly becomes computationally uncontrollable as iterations increase by the size of state space. Second method is Monte Carlo methods which instead of solving all possible sequences as in dynamic programming, solves the problem by averaging sample returns. Benefits of Monte Carlo methods is that it does not assume complete knowledge of the environment, and although a model is required, the model only need to generate sample transitions and not represent complete dynamics of environment. The application of Monte Carlo is limited to tasks with periodic structure as calculation of the returns is only possible at the end of some sort of episode. This means Monte Carlo methods only learn in an episode-by-episode level, not step by step. While this expands the computational efficiency, the method may miss knowledge that step-by-step approach could learn. Third, a combination of dynamic programming and Monte Carlo methods is presented. This method, temporal difference learning, can learn directly from experience with environment without a model of environment like Monte Carlo methods, and learn incrementally by updating the value function and policy function after each action like in dynamic programming. Temporal difference learning involves the sequence of actions, where the value of the following state is utilized to update the values of the states that came before it. The value of the preceding state is updated by a fraction of the difference between the value of that state and the former state. To conclude, temporal difference learning combines the best of both dynamic programming and Monte Carlo methods.

In reinforcement learning, the agent usually acts greedily to look for maximal rewards. Adversarially, constantly focusing on maximising the rewards (i.e, greedy policy) may not result in the best policy. For example, if we have an environment where one can queue multiple times to a counter where one receives either \$1, or can flip a coin and win \$1000 if it is heads. It is clear that the expected return is higher for flipping a coin ($0.5 * \$1000 = \$500$ vs. \$1 for not flipping). Now in a situation where the agent chooses to flip a coin but it is not heads (agent receives \$0), the agent learns that the action for flipping a coin does not provide any rewards. When repeated, the agent will always choose the action to take \$1 as it rewards the agent by taking that specific action. It is noticed that the optimal policy has not been achieved.

To overcome this problem, exploration can be introduced to the algorithm. This means that the agent randomly takes random actions to "explore" the environment. Give the earlier example, the agent could then learn that there is probability to receive \$1000 for flipping a coin, and optimize the policy accordingly. Conversely, it is desired that the agent actually learns and exploits its experience, not functioning solely based on random actions. This is known as exploration-exploitation trade-off. This is a balance between exploration in which new features about the environment is discovered by selection of sub-optimal action, and exploitation in which the agent uses previous knowledge about the environment to get the best results known. One way to approach this is a ϵ-greedy selection in which there is a small probability ϵ where the agent will select an action from the sub-optimal actions. (Coggan, 2004). The ϵ value can be higher in the beginning (the agent explores more) and decayed over time so that the exploration declines.

2.3.2 Deep Learning

Application of deep learning to existing artificial intelligence problems have demonstrated great success, indicating its superiority over conventional machine learning techniques in many cases. Thanks to its excellence in detecting complex structures in high-dimensional data and the fact that it requires very little engineering by hand, deep learning is applicable to many domains of science, business and government. Some areas where deep learning has significantly advanced the research include image and speech recognition, various fields of health science (such as predicting the activity of potential drug molecules and reconstructing brain circuits), and natural language understanding (topic classification, sentiment analysis, and question answering). (LeCun et al., 2015). An advantage of deep learning model is not only that it can be used on extremely large amount of data but also that input data can be in various forms: sequential data (any kind of data where order matters), image or 2D data, or tabular data (rows and columns) (Sarker, 2021).

While conventional machine learning techniques are limited to processing natural data in their raw form thus requiring careful engineering and domain expertise to transform the raw data to suitable representation or feature vector, deep learning methods can learn multiple levels of representation and detect patterns in the input without strict feature engineering. Deep learning achieves this by building multiple layers of interconnected modules, called artificial neural networks. Each layer refines and transforms the data representation, allowing the network to capture complex relationships. (LeCun et al., 2015). Given this explanation, deep learning can be seen as a learning of data representations rather than learning focused on solving specific task. A more generalized definition of deep learning is given by Zhang et al. (2018a) that define deep learning as "a process not only to learn the relation among two or more variables but also the knowledge that governs the relation as well as the knowledge that makes sense of the relation".

The structure of deep neural network (DNN) consist of neurons or "nodes", layers, connections, and activation function. There are different types of nodes that are connected to each other between layers: input nodes, hidden nodes, and output nodes. These nodes are organized into layers similarly: input layer that receives the initial input data, multiple hidden layers that are the layers between the input and the output layer that process the information, and finally output layer that produces the final output. Nodes in one layer are connected to nodes in the next layer by weighted connections. These weights represent the strength or importance of the connection between the nodes and are adjusted during the learning process. Each node has an activation function that determines its output based on the weighted sum of its inputs. (Sarker, 2021). An example of activation function is sigmoid activation function that maps any real valued number to a value between 0 and 1 which can be interpreted as probability (Wood, 2020). Figure 1 illustrates the neuron input and output.

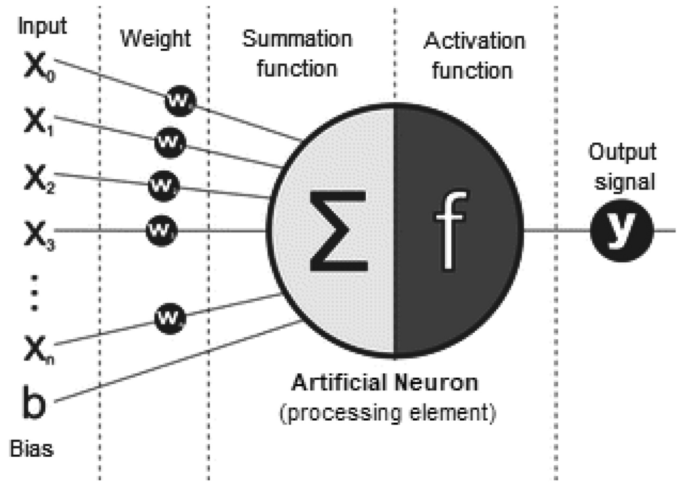


Figure 1: Neuron (Sarker, 2021)

When training DNN to complete a specific task, an objective function is used to measure how well the algorithm performs. The connections of nodes, the weights, act as adjustable parameters in this objective function. To minimize the error and improve performance of the model, the learning algorithm calculates a gradient vector for each weight. This vector tells how much the error would change if the weights would be adjusted. The gradient vector is then utilized to adjust the weight vector in the opposition direction. To compute the gradient of an objective function, a backpropagation procedure is often used which utilizes the chain rule of derivatives commonly known in math domain. (LeCun et al., 2015). The chain rule states that derivative of $f(g(x))$ equals to $f'(g(x)) \times g'(x)$ (Khan, n.d.). What is insightful is that the gradient can be calculated by working backward from the gradient with respect to the output of each node, thus the name backpropagation. When this is repeated through all nodes, starting from the output layer all the way to input layer, the gradients can be propagated. After that, the gradients can be computed with respect to the weights of each node. (LeCun et al., 2015). The chain rule is illustrated in Figure 2.

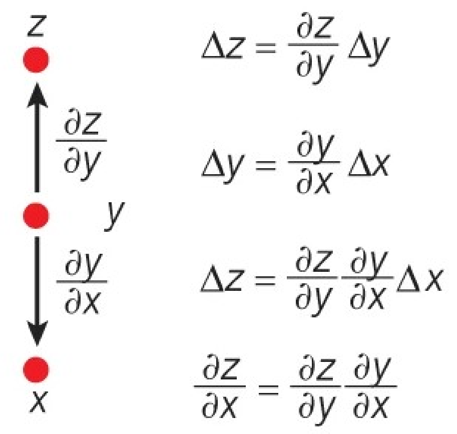

z
y
x
$\frac{\partial z}{\partial y}$
$\frac{\partial y}{\partial x}$
$\Delta z = \frac{\partial z}{\partial y} \Delta y$
$\Delta y = \frac{\partial y}{\partial x} \Delta x$
$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$

Figure 2: Illustration of chain rule (LeCun et al., 2015)

In Figure 3, the structure of multilayer neural network and backpropagation process is depicted. The architecture consists of input nodes (input data) on input layer, hidden nodes on hidden layer one, hidden nodes on hidden layer two, and output nodes on output layer. The backpropagation process is divided to forward pass and backward pass.

In the forward pass of a neural network, each node computes a weighted sum of the outputs from the nodes of the previous layer. Then, it applies a non-linear activation function to this sum to get the node's output. For example, the rectified linear unit (ReLU) function, $f(z) = \max(0, z)$, is one of common activation functions used. Forward pass process starts at the input layer, where each node represents original data x_i. These node outputs then move to the first hidden layer (H1), where each node calculates its weighted sum from the input layer's nodes and applies the activation function. The outputs from H1 nodes then move to the next hidden layer (H2), and so on. Finally, the output layer nodes receive weighted sums from H2 nodes, passing it through an activation function, and produce the final network outputs. Each arrow between nodes represents a weight, and all nodes (except those in the input layer) perform calculations based on their inputs and weights. The final output is obtained after passing through all layers of this network.

During the backward pass, the process works in backwards order compared to forward pass. Therefore, the backward pass kind of "undoes" the steps from the forward pass. First, the outputs are compared with the correct answers to get error derivatives. These derivatives quantify the amount of difference between predicted and target outputs, illustrating the magnitude of errors at each node. This is calculated using the equation $\frac{\partial E}{\partial y_j} = y_j - t_j$, where y_j is the predicted output and t_j is the target output (given that cost function is $0.5(y_l - t_l)^2$). In the output layer, the error propagation is examined by studying how adjustments to the input affect the observed errors. This process involves looking at the error derivatives ($\frac{\partial E}{\partial z_l} - \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$) with respect to the output and input, incorporating the gradient of the output activation function. On the hidden layers, such as H2 and H1, the influence of individual weights on the total error is assessed. This is calculated by $w_{kl}\frac{\partial E}{\partial z_l}$. With this equation, it can be studied how modifying each weight impacts the error landscape.

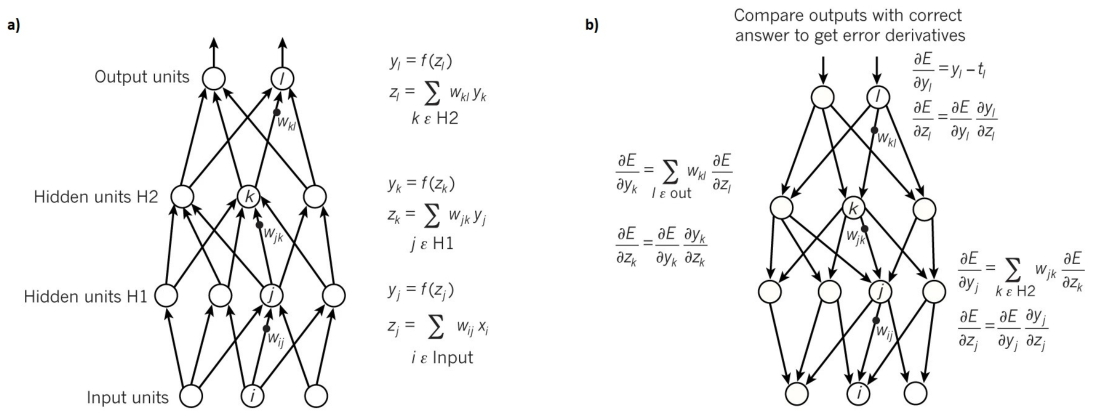

a)
Output units
Hidden units H2
Hidden units H1
Input units
$y_l = f(z_l)$
$z_l = \sum_{k \in H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \in H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \in Input} w_{ij} x_i$
b)
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \in out} w_{kl}\frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk}\frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$

Figure 3: a) Forward pass in backpropagation process b) Backward pass in backpropagation process (LeCun et al., 2015)

Sarker (2021) created a taxonomy for deep learning applications, mapping known algorithms to three major categories: supervised learning, unsupervised learning, and hybrid learning and others. Supervised learning refers to the fact that a model is given data samples and their respective outcomes whereas in unsupervised learning precise supervisory information such as target class labels are not provided. Unsupervised learning is convenient when the labels are not available or extracting feature information is expensive or time consuming. The hybrid learning and others can be interpreted as an integration of these two learning methods and other relevant techniques.

Under these categories, there are a large number of different deep learning algorithms. One of the many is feedforward neural network, a supervised learning method, in which the architecture is designed so that the information only moves forward from input layer to hidden layers to output layer (contrary to recurrent neural networks where information is looped back and forth). To avoid misunderstandings, it is important to clarify the difference between feedforward neural network and backpropagation. Feedforward neural network is a architecture of neural networks where it takes in an input vector and calculates an output vector. Backpropagation, which was presented earlier, is not an architecture but an training method which functions as in Figure 3. In backpropagation, the network learns to adjust its parameters (weights and biases) based on the difference between the predictions and the actual outcomes. Then optimization algorithms, such as gradient descent, are applied to iteratively optimize the network's weights, minimizing the error and improving its performance. Many feedforward neural networks use backpropagation in training the algorithm.

However, Sarker (2021) notes that building an efficient deep learning model is a challenging task because of the variations of real-world problems and the dynamic nature. It is also mentioned that deep learning models are also usually considered as "black-box" models where the functioning and reasoning of deep learning model can be difficult to interpret. Training the deep learning model is both depended on large amount of data and high computational capabilities. The training period itself is much slower than for conventional machine learning models (even weeks compared to seconds or hours) but once the model is trained, testing the model on new data is faster with deep learning models as the process requires little computational effort.

2.3.3 Deep Q-Network

The roots of DQN algorithm lie in first concepts of Q-learning. Q-learning fundamentals were first introduced in a study *"Learning from Delayed Rewards"* written by Watkins in 1989. Later in 1992, Watkins and Dayan published an article *"Q-learning"*. In the article, Q-learning is defined as a form of model-free reinforcement learning that follows the usual reinforcement learning settings where the agent takes an action at a particular state and evaluates the consequences of the action. The consequences are measured as a immediate reward or penalty the agent receives. In addition to immediate reward or penalty, the future rewards are also considered as a long-term discounted reward. By trying all possible actions in all possible states repeatedly, the agent learns how the rewards are maximized i.e., what the best actions to take are. In model-free learning, the agent learns a policy or a value directly from the environment without explicitly modeling the dynamics of the environment. The structure of Q-learning is illustrated in Figure 4.

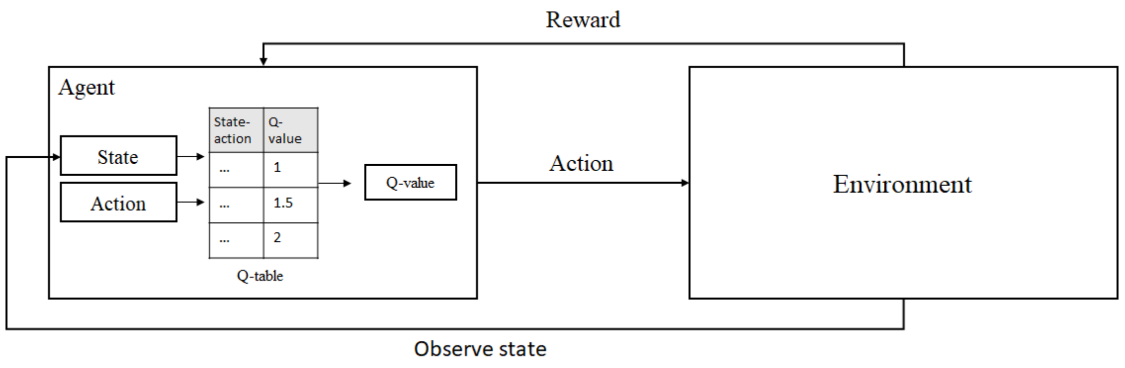


Figure 4: Functioning of Q-learning

The Q-learning process can be formed mathematically based on works of Bellman (1957). The Q-function to calculate the value of actions is a weighted average of current value and new information, and represents the expected cumulative reward for taking a particular action in a given state:

$$Q^{\pi}(s_t, a_t) \leftarrow (1-\alpha) \times Q(s_t, a_t) + \alpha \times \Big(R_{t+1} + \gamma \times \max Q(s_{t+1}, a_{\pi})\Big), \tag{1}$$

where $Q^{\pi}(s_t, a_t)$ is the expected discounted reward for taking action a_t at state s_t and following policy π afterwards, α representing the learning rate of the algorithm, $Q(s_t, a_t)$ the current value, R_{t+1} the reward received taking action a_t in state s_t, γ represents the discount factor, and $\max Q(s_t, a_{\pi})$ the maximum reward possible to get from state s_{t+1}.

Carta et al. (2021) describe the process that at each time t, the agent takes action a_t and observers a reward R_t, enters into a new state s_{t+1} that depends on the previous state s_t and the selected action a_t, and finally the Q-value is updated as in Equation 1. This will then guide the next action of the agent. The Q-values can be mapped to Q-table, that has the Q-values for all combinations of states and actions. From Q-table, the maximal value is observed and optimal action a^* is taken. This can be represented mathematically as follows:

$$a^* = \text{argmax}_a(Q(s, a)) \tag{2}$$

The Q-learning algorithm is considered off-policy learning algorithm. This means that two different policies are used in the policy learning process: one policy is used to estimate the value functions while another is used to control the improvement process. This means the agent learns the optimal policy without actually following it directly. The Q-values are updated by using experiences from behavioral policy, which may differ from the target policy that the agent is actually trying to learn. To conclude, the behavioral policy acts as a policy for exploring the environment and collecting experiences, and the target policy is the policy the agent aims to optimize. (Bertoluzzo and Corazza, 2012).

In 2013, Mnih et al. (2013) were the first to create a deep learning algorithm using variant of Q-learning. Their algorithm was designed to play Atari games. The success of the paper can be depicted by the fact that the authors' research lab "DeepMind Technologies" was acquired for £400m by Google shortly after publication of the article (Gibbs, 2014). As of this date, Google DeepMind plays a crucial part in Alphabet's (Google parent company) business (Pichai, 2023). Later, Mnih et al. (2015) published a study where the DQN agent was able to beat professional human games tester across a set of 49 games of Atari 2600. Even though DQN algorithms were first designed to play games, the technique has later seen wide adoption in wide range of industries such as malware defence (Khowaja and Khuwaja, 2021), automated driving systems (Zhang et al., 2018b), robotics (Jiang et al., 2019), and stock market trading (Park et al., 2020). Based on deep learning categorization made by Sarker (2021), DRL algorithms are part of "relevant others" category, as DRL approaches the problems differently compared to many other deep learning algorithms. Figure 5 shows the basic concept of DQN functioning.

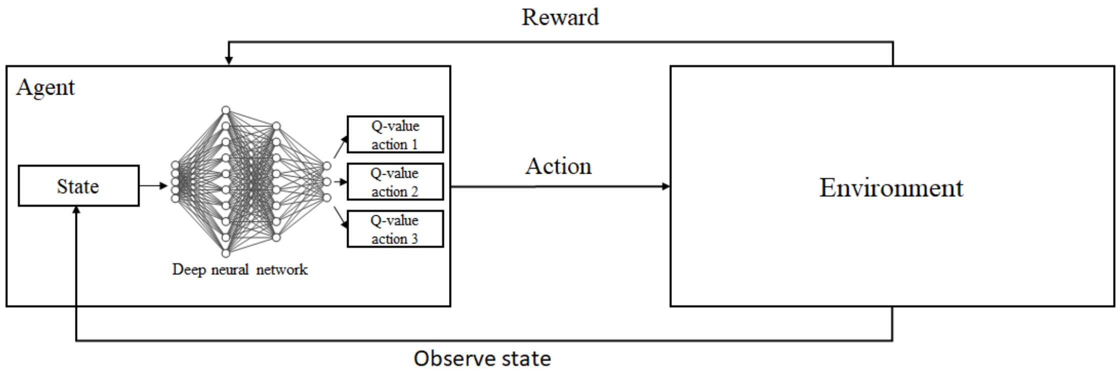


Figure 5: DQN schematic structure

Although the key concept of DQN is very similar to traditional Q-learning (the agent receives feedback based on its actions and attempts to maximize the rewards it is given), it replaces the Q-table with neural network. This DNN is used to learn weights in order to approximate the Q-function. During the training process of DQN, the aim is to find weights w that predicted Q-values are as accurate as possible. This process begins by initializing Q-values for all actions to zero, and the agent randomly takes an action a in the initial state s, receiving feedback from state s_{t+1}. In the beginning the policy is rather arbitrary, but the agent quickly learns from the environment and adjusts the DNN weights. To train the algorithm, it calculates first the Q-function such as Equation

1 to approximate the so-called target function and then weights are updated by using the following equation:

$$w_{t+1} = w_t - \alpha \nabla_w \mathbb{E}\left[(Q_{w_t}(s,a) - Q(s,a))^2\right], \tag{3}$$

where $\nabla_w \mathbb{E}\left[(Q_{w_t}(s,a) - Q(s,a))^2\right]$ is the gradient of the mean squared error between the Q-value predicted by the network $Q_{w_t}(s,a)$ and the true $Q(s,a)$ which is calculated by using equation

$$Q(s,a) = \sum_y p_{sy}(a) \times \left[R(s,y,a) + \gamma \mathrm{max}_b Q(y,b)\right], \tag{4}$$

where s is the state given as input, y is any other state given by the environment according to agent's action, a and b are actions and p_{sy} is the probability of moving from state s to y given that action a was done. $R(s,y,a)$ represents the immediate reward from taking action a and moving from state S to y and γ is the discount factor. (Carta et al., 2021). Contrary to Q-learning, DQN outputs multiple Q-values, one for each action. The action with highest Q-value is then executed.

The training or learning of the agent is based on its experience with the environment. In standard reinforcement learning algorithms, the experience of the agent is immediately discarded after it has been used for learning. To speed up the learning and break temporal correlations (statistical relationship or dependence between observations over time), a fixed-size memory replay can be used where the experience is stored. This allows the agent to learn from earlier memories. When the fixed-size memory reaches its limits, oldest memories are discarded. For updating the weights of the algorithm (training process), a random batch of experiences is used from the experience replay buffer. (Liu and Zou, 2018). Experience replay method also prevents that the learning would happen only through the experience of a specific situation (Park et al., 2020).

These batches are then used to calculate the temporal difference error, which is the difference between the predicted Q-value and the target Q-value. The predicted Q-value is obtained from the Q-network for the given state-action pair while the target Q-value is computed using the Q-function which consists of the observed reward and the estimated maximum future rewards. Computing the target Q-value is where the DNN is trained. After the temporal difference error is calculated, backpropagation is used to minimize the temporal difference error by adjusting weights of the DNN. To stabilize training, target Q-network with a fixed set of weights is used for generating target Q-values during the calculation of temporal difference errors. This target network is updated periodically with the weights from the trained Q-network so that it can also predict more accurate Q-values.

When this process is repeated iteratively, the algorithm develops towards optimal policy within the environment. However, achieving well functioning policy requires experimenting with different hyperparameters. Hyperparameters are parameters whose value is fixed and chosen before a learning algorithm is trained. These hyperparameters are set by the user and greatly affects the performance of the algorithm. Common hyperparameters are learning rate, discount factor, experience replay size, and batch size.

Finally, once the algorithm is trained and fine-tuned successfully, it can be used on new, out-of-sample data in similar environment. That is one of the advantages of DNN: the training of the algorithm is time-consuming and computationally expensive, but once it is trained, it can be leveraged on new data very efficiently.

2.3.4 Advantages and challenges of DQN in stock trading

Many features of DQN make it well-suited for financial markets. Firstly, the financial markets data present a challenge due to its heterogeneousness: it is in many forms (structured data, unstructured data, and semi-structured data), complex in types and huge in volume, and mixed with a lot of noise. Compared to traditional machine learning models, applications of deep learning to analyze and predict financial data has significantly improved the performance of AT strategies. (Wang and Yan, 2021). Deep learning methods have the ability to search the hidden patterns in high dimensional data (LeCun et al., 2015), which may be one of the factors contributing for succession of deep learning techniques in finance. In addition, DQN is a model-free approach which is extremely suitable for environments like stock market where modeling the entire environment is unfeasible (Jin and El-Saawy, 2016).

Secondly, stock trading involves making sequential decisions over time based on historical and real-time market data. DQN is the most popular choice for sequential decision-making problems (Ramaswamy and Hüllermeier, 2022), making it well-suited for modeling financial markets where actions are taken step by step. The sequential aspect of DQN is particularly relevant in topics like capturing trends, detecting reversals, and identify different market cycles as sequence of historical prices are taken into account.

Thirdly, financial markets incorporate non-linearity and complex patterns. DQN can learn non-linear patterns that would otherwise be difficult to model, and by using experience replay memory, DQN can remember and learn from past experiences to formulate optimal investing strategy or policy. Deep learning can extract features automatically from raw input data which can derive complex patterns from financial markets data (Thakkar and Chaudhari, 2021). This decreases the manual feature extraction and modeling required from user, as DQN can learn from its own actions and rewards and not rely on predefined rules.

Finally, there exists evidence that novel DQN algorithms have outperformed benchmark strategies such as buy-and-hold and other traditional investing strategies: Chakole and Kurhekar (2020) demonstrated a novel trend following approach DQN that outperformed benchmark strategies on U.S. and Indian stock markets; Carta et al. (2021) used a multi-agent DQN on U.S. stock market and German stock market with promising results; and Théate and Ernst (2021) used Double DQN on various U.S. stocks and indices, surpassing the benchmark strategies on average. However, the results of such studies should be approached with caution. The performance of such studies can be subject to specific settings, and while suggesting to offer great results in such circumstances, the results may differ greatly when reproduced in distinct settings. Ioannidis (2005) criticizes that in general research, the most of the published research findings are probably false due to multiple testing, small sample sizes, publication bias, and replication failure. Despite the magnitude of the statement, it is a good reminder for the readers of such research.

Despite its success, DQN faces challenges when applied to stock trading. To begin with, the black-box problem in which the functioning or acting of algorithm is difficult to interpret is present with DQN (Zahavy et al., 2016). For example, given the studies where DQN has outperformed traditional strategies, it can be challenging to explain why DQN has returned higher returns. The agent making trading decisions acts "on its own", and root causes for such actions can be difficult to retrieve. Zahavy et al. (2016) also mention that when DQN is not performing well, it is difficult to find the cause for poor performance and even harder to find ways to improve it.

In stochastic environments, like stock markets, DQN algorithms tend to overfit as it overestimates action values. In the learning process, the maximum action values are used as an approximation for the maximum expected action value which may result in overfitting. (Hasselt, 2010). The problem of overfitting is that after the model is trained on training data, applying it to out-of-sample test data that was not part of training data will result in poor outcomes. Overfitted models also tend to capture noise in the training data, decreasing the robustness of such models in changing market conditions. This will also weaken its ability to generalize well when met with unseen data.

Thakkar and Chaudhari (2021) note that real-time prediction of stock market is challenging, even impossible for some methods, and the time required to train a prediction model needs to be improved. Processing through multiple layers, the learning of large number of parameters, performing backpropagation, and iteratively enhancing the model is both computationally intensive and time-consuming. However, the availability of ever increasing powerful hardware lessens these constraints (Carta et al., 2021).

Related to creation of deep neural network algorithms, the selection of hyperparameters, such as learning rate or discount factor, is crucial for the final efficiency of the algorithm. The results for particular hyperparameter should be evaluated against other possible values (i.e, fine-tuning of the algorithm). Despite deploying a functional algorithm, the influential factors of stock market such as economic, psychological, and social aspects are both challenging to identify and introduce to the algorithm. (Thakkar and Chaudhari, 2021). Even if deep learning models could create fairy-tale models, it is always limited to what it is given as an input data.

2.4 Previous studies

Early developments of reinforcement learning in stock market research began when researchers started exploring reinforcement learning for portfolio optimization and trading strategy development. One of the first studies were by Neuneier (1995, 1997) who used Q-learning for multi-asset portfolio optimization, followed by Gao and Chan (2000) who derived Q-learning algorithm combined with Sharpe ratio maximization algorithm. Moody et al. (1998) and Moody and Saffell (2001) came up with recurrent reinforcement learning algorithm for a multi-asset long-short trading strategy. Later, more studies utilizing Q-learning were introduced. Casqueiro and Rodrigues (2006) used Q-learning to create a trading strategy that similar to Gao and Chan (2000), attempted to maximize the risk-adjusted returns. O et al. (2006) developed an algorithm that incorporated both stock selection and asset management based on four pattern-based predictors. Lee et al. (2007) proposed an approach that incorporated multiple Q-learning agents instead of one. In this approach, each agent had its own goal, for example one agent acting as buy signal agent and another sell signal agent, that interacted with each others throughout the learning process.

The research done on Q-learning continued on 2010s with range of different approaches, many of which were contributed by researchers at Google DeepMind. In the beginning of the decade, Hasselt (2010) introduced Double Q-learning. Instead of updating a single Q-function, two Q-learning functions are used in the learning process. Each Q-function is updated crosswise, so that Q^a is updated from Q^b for the next state and vice versa. The motivation for approach was that single Q-learning was found performing poorly in stochastic environments. The Double Q-learning reduces overestimation and enhances performance in stochastic environments by maintaining two separate sets of Q-values where one set is used for action selection, and the other set is used for action evaluation.

Silver et al. (2014), most of them working at Google DeepMind, presented Deterministic Policy Gradient (DPG) algorithms for reinforcement learning with continuous actions. They found that the estimating DPG is much more efficient than estimating the usual stochastic policy gradient. DPG is closely connected to Q-learning as in both approaches given optimal action-value function $Q^{\pi}(s, a)$, the optimal action a^* can be derived from Equation 2. The motivation for PDG emerges from that when there exists finite number of discrete actions, the maximum Q-value action can easily be computed, but when the action space is continuous, evaluating the action space is non-trivial for classical Q-learning. It would require to calculate $\max_a Q(s, a)$ every time the agent wants to take an action in the environment. As the action space is assumed to be differentiable with respect to action parameter, a gradient-based learning rule for policy $\mu(s)$ can be used. Then, contrary to optimizing expensive subroutine for every $\max_a Q(s, a)$, it can be approximated with $\max_a Q(s, a) \approx Q(s, \mu(s))$. (OpenAI, n.d.). Later, this technique has been utilized for research on stock market by Liang et al. (2018), Khemlichi et al. (2021), and Zhang et al. (2021).

Schaul et al. (2016), researchers at Google DeepMind, came up with Prioritized Experience Replay (PER). In regular experience replay, the agent stores past experiences, or transitions, in a replay buffer and samples batches of experiences randomly during the training process. PER functions very similarly, but prioritizes which transitions are replayed. Using this method, it is thought that the agent can learn more effectively from some transitions over others by assigning different priorities to different experiences in the replay buffer based on their temporal difference error. Schaul et al. (2016) found that using PER, the learning was sped up by a factor two and led to state-of-the-art performance compared to benchmark.

Wang et al. (2016), also under Google DeepMind, continued improving the efficiency of learning in Q-learning further. This time, Wang et al. (2016) established Dueling Q-Networks, a new neural network architecture for model-free reinforcement learning. Whereas in traditional Q-learning the Q-value for a state-action pair represents the cumulative expected reward starting from that state and taking a particular action, this Q-value is decomposed into two components in Dueling Q-Networks. The two components are value function and advantage function. Value function represents the value of being in particular state and advantage function represents the advantage of taking a specific action in that given state. These two are then combined as a Q-value. This architecture further improves the ability to learn the state-value function efficiently.

After Mnih et al. (2015) achieved promising results by combining deep learning and

Q-learning in playing video games, it also raised the curiosity of market participants and researchers within finance domain. Many practitioners and researchers initiated studies to examine whether this DQN method could be used for excess returns in stock market. One of the earliest implementations of DQN to financial markets is done by Jin and El-Saawy (2016). They used the technique to manage stock portfolio of two stocks, one with a low volatility and one with a high volatility. The results were on par with benchmark, some models outperforming in terms of raw returns. Since then, DQN has been researched with a large number of variations. Huang (2018) developed a recurrent DQN for forex trading with additional modifications such as deploying considerably smaller replay memory and "activation augmentation" which lessens the need for random exploration as it provides extra feedback signals for all actions to the agent. Even considering transaction costs, the results were positive in most of the simulation settings. Li et al. (2019) extended the value-based DQN and actor-critic (A3C) to the trading of U.S. stocks and futures contracts. To capture temporal patterns, they utilized a Long Short-Term Memory (LSTM) module. Shin et al. (2019) contributed to the research of DQN in stock market by deploying a multimodal DQN with Convolutional Neural Network (CNN) and LSTM layers on 256 stocks on Korean stock market. Wu et al. (2020) used gated recurrent units to create two trading strategies, Gated DQN and Gated DPG, to trade on three different stock markets. Several technical indicators were used with OHLCV data to depict the stock market environment. Park et al. (2020) and Hirsa et al. (2021) studied DQN performance on multi-asset environment where the agent had multiple assets to trade with.

While these studies primarily employed the "traditional" architecture of DQN with variations, there is also research that explores the combination of deep learning with Double Q-learning, PER, and Dueling Q-Networks. Double Q-learning with deep learning has found applications across a broad range of financial literature: Carapuço et al. (2018) used Double Q-learning for trading on forex market; Ning et al. (2021) utilized the technique for optimal trade execution; Théate and Ernst (2021) employed it on trading stocks and indices on U.S. stock market; and Massahi and Mahootchi (2024) tested it on commodity futures markets. Much like double Q-learning, PER has seen adoption in many algorithms developed for trading (Li et al. (2019); Liu et al. (2020); Wang et al. (2021)). Dang (2019) experimented Dueling Q-Networks in stock trading and compared its performance with DQN and Double DQN. Gao et al. (2020) combined Dueling Q-Networks with Convolutional Neural Network (CNN) in stock market portfolio management. Chen et al. (2023) created a stock prediction algorithm by connecting recurrent neural networks and Long Short-Term Memory with Dueling Q-Networks.

To address recent literature on the use of DQN on stock trading, publications on the topic for the last three years have been collected to Table 1. The review aims to give a view on different methods and approaches that have been conducted recently.

Table 1: Applications of DQN in trading over the past three years

Publication	**Data Set**	**Features**	**Period**	**Network architecture**
Ansari et al. (2022)	Global stock market	Closing prices	2000-2021	GRU
Awad et al. (2023)	Gold price	Closing prices & sentiment data	2000-2021	NLP, VMD, LSTM
Brim and Flann (2022)	U.S. stock market	Candlestick images	2013-2019; Jan 2020-Jul 2020	Double DQN, CNN
Carta et al. (2021)	U.S. and German stock market	Historical OHLC	2007-2017	Dueling DQN, Multiple agents
Cui et al. (2023)	U.S. stock market	OHLCV, Sharpe ratio	2007-2017; 2018-2020	Double DQN, Multi-scale CNN
Hao et al. (2022)	Chinese index futures	OHLCV, Pre-close, Pct-change, Money	2005-2022	Classical DQN
He et al. (2022)	U.S. stock market	Closing prices	1980-2022	Double DQN, LSTM, TD3
Hirsa et al. (2021)	S&P 500 futures	Varying period returns	2000-2019	DQN
Hu (2023)	U.S. stock market & Bitcoin	OHLC	1999-2020	PER, Noisy Networks, Dueling DQN, Double DQN
Hu (2022)	Gold price & Bitcoin	Closing prices	2016-2019	LSTM, CPMA
Huang et al. (2023)	Global stock markets	OHLCV, technical indicators	2007-2022	LSTM, Dual action & Dual Environment
Kumar et al. (2022)	Global stock markets	OHLCV	2014-2021; Jun 2021-Jun 2022	SHAP
Lazov (2023)	Cryptocurrency pair ADA/USDT	Closing price, volume	2018-2021	Double DQN, FLN
Li et al. (2021)	Chinese stock market	Open, close, technical indicators, sentiment data	2016-2021	ARBR, PCA
Luo and Duan (2023)	U.S. stock market	Prices, technical indicators	2013-2022	DQN
Nagy et al. (2023)	U.S. stock market	Limit order book data	2012, 2022	Double DQN, Dueling DQN, APEX, LSTM
Ning et al. (2021)	U.S. stock market	Limit order book data	2017-2018	Double DQN
Otabek and Choi (2024)	Bitcoin	Historical price, sentiment data	2014-2019	Multi-level DQN
Parkavi et al. (2022)	U.S. stock market	Historical price	Not indicated	Dueling DQN, Double DQN, LSTM
Pigorsch and Schäfer (2022)	U.S. stock market	Moving average of prices	2010-2021	DQN
Qiu et al. (2022)	Bitcoin, gold	Historical prices	2016-2021	EEMDAM-LSTM
Sarkar (2023)	Indian stock market	OHLC	2010-2020	DQN
Suliman et al. (2022)	Cryptocurrency markets	OHLCV	Jul 2020-Aug 2020	Dueling DQN
Théate and Ernst (2021)	U.S. stock market	OHLCV	2012-2017; 2018-2019	Double DQN
Tran et al. (2023)	Bitcoin	OHLC	Mar 2022-Aug 2022	Double DQN, Dueling DQN
Wang et al. (2023)	Chinese stock market	OHLCV, trading amount, turnover rate, volume ratio	2011-2020	LightGBM
Ye et al. (2022)	Chinese stock market	Limit order book data, trade data	2014-2015	Multiple agents
Zhou and Tang (2021)	Chinese stock market	Historical price, technical indicators, sentiment data	2017-2021	ARBR
Zhu and Zhu (2022)	U.S. stock market, Bitcoin	Historical price, volatility, spread	2017-2021	CNN

3 DATA AND METHODOLOGY

3.1 Dataset description

This study aims to generate excess returns on Finnish stock market using novel AT technique DQN. While DQN has been applied to many markets (see Table 1), it appears that the technique has not yet been put into use on Finnish stock market. This specific market differs from many of the markets studied earlier in form of market size and liquidity. Subject to less analysis, Finnish stock market and especially smaller individual stocks may turn out to be less efficient and result in different outcomes of DQN. The dataset for this study consists of 3 Finnish stocks (Nordea, Sampo, Bittium) and OMX Helsinki PI index. The three stocks are chosen to evaluate the performance of DQN on individual stocks, and OMX Helsinki PI to represent the broader Finnish stock market. The reasoning for choosing these three individual stocks is as follows: Nordea has the highest market capitalization and is one of the most traded stock in Finnish stock market; Sampo represents a blue chip stock of Finnish stock market; and Bittium is chosen to depict a smaller growth company with less liquidity. These three stocks also categorize into different industries: banking, insurance, and wireless business respectively. OMX Helsinki PI includes all the shares listed on the Helsinki Stock Exchange (currently 139 components), and reflects the current status and changes in the Finnish stock market. (Nasdaq, n.d.).

The data set consists of daily observations of opening price, highest price, lowest price, closing price, and volume traded. Together this data set is abbreviated as OHLCV. A long research period is selected to minimize the impact of any particular market environment, to capture long-term trends, and to incorporate different market regimes. Data is retrieved from 1^{st} of January 2015 to 31^{rd} of December 2023. This time period includes variety of exceptional market conditions such as COVID-19 crisis and rapid rise in interest rates. The data is split into training period during which the DQN agent is trained, and testing period during which the performance of DQN is tested on out-of-sample data. The training period spans from 1^{st} of January 2015 to 31^{rd} of December 2021. The testing period in which the performance is analysed covers from 1^{st} of January 2022 to 31^{rd} of December 2023.

3.2 Proposed approach

The proposed approach of this study closely follows the works of Théate and Ernst (2021). They have made the experimental code publicly available[1], which lays the basis for this study. This experimental code is adjusted to meet this study's modified approach to include trading limits into the rewarding of the agent.

This study proposes a sequential decision-making algorithm DQN to execute an AT strategy. This trading strategy follows the programmed policy of DQN agent. This rule-based policy is sequential process where the agent trades step by step, following the optimal policy $Q^{\pi}(s_t, a_t)$, where s_t is the state (information) and a_t is optimal action of the agent at time step t. The process updates the available market information (state s_t), executes the optimal policy $Q^{\pi}(s_t, a_t)$ to get optimal action a_t, executes action a_t and receives feedback from interacting with the environment, and finally takes next time step from t to $t+1$ and continues this process iteratively.

In real world trading environment, making trading actions usually involves transaction fees. Neglecting transaction costs can result in models that are disconnected from real-world trading scenarios, indicating promising returns in test environment, but performing poorly when carried in real trading environment. That is why the approach in this study is executed in an environment without transaction costs, but also with transaction costs. By analyzing both environments, the performance of the trading strategy can be evaluated in both worlds, and possible differences can be studied. When transaction costs are taken into account, for each trading action taken a percentage of the capital is lost by deducting this from the agent's cash value. Taking an action is set to cause transaction costs of 0.2% for environment with transaction costs.

In order to deploy this strategy, it requires the setting up of the trading environment, trading agent, and some assumptions related to these. The more precise specifications of this is explained in the subsequent sections.

3.2.1 State space

The state of the agent is a key part of DQN. It represents the information that the agent has given that time. Optimistically, the state s_t would include all information influencing market prices to capture the complex trading environment. Some factors, although not exhaustive, are different technical indicators, macroeconomic information,

[1]`https://github.com/ThibautTheate/An-Application-of-Deep-Reinforcement-Learning-to-Algorithmic-Trading`

news information, and any extra information such as companies' confidential information or rumours. (Théate and Ernst, 2021).

The state space of the agent is limited in this study, and consists of the information tensor from the environment (OHLCV data) and a weight vector of the current portfolio. This can be represented similarly as in study made by Park et al. (2020), which represents the state s_t in time step t:

$$s_t = (X_t, W_t), \tag{5}$$

$$W_t = \Big(W_{t,0}, W_{t,1}, W_{t,2}, ..., W_{t,I}\Big)^T, \tag{6}$$

$$X_t = \Big[p_t^O, p_t^H, p_t^L, p_t^C, V_t\Big], \tag{7}$$

where W_t is the weight vector of the current portfolio, and X_t the information gathered from the environment, which is composed of the opening stock market price p_t^O of the day, the highest price p_t^H during the day, the lowest price p_t^L during the day, the closing price p_t^C of the day, and total volume V_t exchanged during the entire day.

If given a fixed time window of size n, the agent considers also past information tensors $(X_{t-n+1}, X_{t-n+2}, ..., X_t)$ instead of just current time step t. This can be though as an n-lag autocorrelation of the tensor. Mathematically this can be formed as

$$P_t^x = \big[p_{t-n+1}^x | p_{t-n+2}^x | ... |, p_t^x\big] \qquad \forall x \in \{O, H, L, C, V\}, \tag{8}$$

where every row represents each asset in the portfolio and the columns represent the series of recent OHLCV data.

3.2.2 Action space

Just like state space, action space is a crucial part of reinforcement learning as well. It determines the set of possible actions the agent can take in the environment. As a matter of fact, similar to the complexity of the state space, the action space in terms of stock trading is relatively intricate as well. Firstly, stock trading can happen at any given time while the stock market is open. This means that the action space is continuous, resulting into a non-trivial action space for classical Q-learning. While there is not entirely infinite amount of state-action pairs, the time complexity is clearly out of bounds for computing Q-values for each state-action pair. Secondly, given that one can buy, hold, or short assets in the portfolio, the range of possible actions is also quite substantial. Despite the first thought that this action space is only limited to three options, it does not take into account that one must decide what is the amount traded as the trader can buy or sell only part of the shares. This expands the action space to continuous quantity that ranges from zero (hold) to the total value of portfolio capitalization (buy or short) if fractional shares are allowed.

Likewise to Théate and Ernst (2021), this study reduces the action space by introducing similar limitations on these two matters. To manage the first problem (continuous timeline), the continuous process Δt is discretized into discrete trading steps t. This means the trading period is split to numerous time windows of equal size. In this case, the time windows are set to one day. This discretization limits the agent to trade only once per day, close to the closing price. To deal with second problem (continuous trading size), the action space is reduced to either going long or short with all available resources, meaning that the agent maximizes the number of shares or maximizes the number of shares borrowed (shorting). Instead of having continuous trading size, the action space hence limited to two possible actions.

This reduced action space consists of two actions as per Théate and Ernst (2021), N_t^{Long} and N_t^{Short}. The trading simulation is conducted as follows: the trading agent starts with €100,000, and can either go long with all available cash or short with all available cash. Therefore, the agent can not buy or sell shares and simultaneously hold cash in the portfolio. For trading actions, two important constraints are defined. Firstly, while share values can be negative (short) or positive (long), cash value can not go negative which limits the amount of shares the agent can buy. Secondly, there is a possibility that in case of shorting, the agent would not be able to repay the share lender in case of significant losses. This is mitigated by constraining the lower bound so that the agent's portfolio can not go below level where it could not return to neutral position anymore. Going long with all available cash can be expressed mathematically as number of shares acquired:

$$N_t^{\text{Long}} = \begin{cases} \left\lfloor \dfrac{v_t^c}{p_t(1+C)} \right\rfloor & \text{if} \quad a_{t-1} \neq Q_{t-1}^{\text{Long}}, \\ 0 & \text{otherwise}, \end{cases} \tag{9}$$

where v_t^c is the cash value of the portfolio at time step t, p_t is the trade price, and C is the trading cost in percentages. By executing this action, the number of shares owned by the agent is $S_t^{\text{Long}} = n_t + N_t^{\text{Long}}$, where n_t is the number existing shares in the portfolio, and Q_t^{Long} indicates the newly acquired shares.

The other possible action, going short by converting the shares into cash value v_t^c and borrowing the shares (shorting) results in the agent having a portfolio with $-N_t^{\text{Long}}$

number of shares. This process can be expressed as:

$$N_t^{\text{Short}} = \begin{cases} -2n_t - \left[\dfrac{v_t^c}{p_t(1+C)}\right] & \text{if} \quad a_{t-1} \neq Q_{t-1}^{\text{Short}}, \\ 0 & \text{otherwise}, \end{cases} \tag{10}$$

However, the N_t^{Short} faces the lower bound constraint. In order to return to neutral position $n_t = 0$ (pay back the borrowed shares), the cash value v_t^c has to be sufficient. This introduces a lower bound $\underline{N_t}$, derived by Théate and Ernst (2021) as follows:

$$\underline{N_t} = \begin{cases} \dfrac{\Delta_t}{p_t \epsilon (1+C)} & \text{if} \quad \Delta_t \geq 0, \\ \dfrac{\Delta_t}{p_t(2C + \epsilon(1+C)} & \text{if} \quad \Delta_t \leq 0, \end{cases} \tag{11}$$

where ϵ represents a percentual maximum relative change in prices ("the maximum market daily evolution supposed by the agent") and $\Delta_t = v_t^c - n_t - p_t(1+\epsilon)(1+C)$. Δ_t represents the difference between the maximum assumed cost to return to neutral position ($n_t = 0$) at the next time step $t+1$. This explains whether the agent can return to neutral position in the worst assumed case at the next time step give that no action is taken at the current step.

In the first case, $\Delta_t \geq 0$, the agent may have problem with returning to neutral position (paying back the borrowed shares), thus constraining the lower bound as above. In the second case, $\Delta_t \leq 0$, the agent has no problem returning to neutral position as the agent owns positive amount of shares ($n_t \geq 0$), or owns negative number of shares and the price decreases, leading to favorable trade for the agent. This constraints the lower bound as mentioned.

Give this lower bound constraint for action N_t^{Short}, the new equation for taking short position is the following equation:

$$\underline{N_t^{\text{Short}}} = \max\bigl(N_t^{\text{Short}}, \underline{N_t}\bigr). \tag{12}$$

3.2.3 Reward function

Rewards serve as a feedback mechanism that guides the agent's decision-making, thus being an important component of DQN. In context of stock trading, the goal is to exploit the algorithm for excess gains. Therefore, it would be intuitive to choose the feedback of the agent to be portfolio returns:

$$r_t = \frac{v_{t+1} - v_t}{v_t} \tag{13}$$

However, the reward mechanism is developed further in this study. In order to guide the agent to find an optimal trading strategy to a greater degree, additional stop loss and take profit levels are introduced. These levels are calculated once the action of the trading agent changes (i.e., for every new entry to long or short position). If the closing price crosses one of these levels while the agent maintains same trading position, triggering the stop loss or take profit will result in additional reward. Given that stop loss is triggered, the agent is penalized for a fixed amount, and if take profit level is hit, the agent is rewarded for a fixed amount.

Whereas normally the design of stop loss and take profit are that once a limit is crossed, the trader returns to neutral position. This means that in long position the position is sold, and in short position the shares are bought back. However, as indicated, the actions in this study are limited to going long or short with all available resources, meaning that the typical return to neutral position once a threshold is triggered is not feasible. Therefore, the agent is not forced to take any action to any direction once crossed stop loss or take profit levels, but rather given an one-time fixed reward for crossing the level. The reasoning for not forcing the agent to change position is that generally, it is not desired to artificially decide the trading positions of the agent as it would be contradicting to the ideology of DQN. With given approach, the reward mechanism guides the agent so that it receives extra reward for performing well and extra penalty for performing poorly. Once one of the levels are triggered, it can no longer trigger again before the trading position is changed. However, it can be noted that it is possible for the agent to trigger both of the thresholds during a trading position.

When calculating the stop loss and take profit levels, it is important that the levels are set as favourable as possible. There lies a risk that if the levels are set too close to closing price, the thresholds keep triggering extremely often resulting in somewhat meaningless threshold levels. On the other hand, setting the levels too far from closing price will result in threshold levels that never trigger. While setting the levels may be difficult to begin with, the dynamic nature of financial markets make the task all the more challenging. As the volatility changes in time in stock markets, the threshold levels should be wider during more volatile times as the price fluctuates more, and narrower during less volatile environment when the price development is more stable.

A suitable approach for placing the stop loss and take profit levels is Average True Range (ATR). First introduced by Welles (1978), ATR is a technical indicator that generates trading signals based on fluctuations in price. However, it can also be used simply as a measure of volatility. ATR is a n-period moving average of true range. The true range is calculated based on daily high, low, and closing prices as follows:

$$TR = \max\big[(\text{high} - \text{low}), |\text{high} - \text{close}_{t-1}|, |\text{low} - \text{close}_{t-1}|\big],$$

where $\text{close}_{\text{t}-1}$ is the previous closing price. The largest value of the three is chosen as a true range. Once the true range is calculated, the average of true range over n-period is calculated to obtain ATR. This means the ATR functions on a rolling basis. The first value of ATR is simply the average of true range over n periods:

$$ATR = \frac{1}{n}\sum_{i-1}^{n} TR_i,$$

and ATR at time step t is calculated on rolling basis as follows:

$$ATR_t = \frac{ATR_{t-1} \times (n-1) + TR_t}{n}$$

By using ATR as a measurement for volatility, it can be leveraged to dynamically set the stop loss and take profit levels as the price fluctuations for the past n-days are considered. Therefore, wider threshold levels are placed when the stock market has been volatile in the recent past, and narrower levels are set during more stable times. In this study, a 14-day time period is used for ATR. It is noted that the stop loss and take profit levels are only calculated if the trading position changes. When entering into new trading position, the thresholds can be equated as:

$$\text{Stop Loss} = \begin{cases} p_t^C - (\text{ATR}_t \times M) & \text{if} \quad a_t = Q_t^{\text{Long}}, \\ p_t^C + (\text{ATR}_t \times M) & \text{if} \quad a_t = \underline{Q_t^{\text{Short}}}, \end{cases}$$

$$\text{Take Profit} = \begin{cases} p_t^C + (\text{ATR}_t \times M) & \text{if} \quad a_t = Q_t^{\text{Long}}, \\ p_t^C - (\text{ATR}_t \times M) & \text{if} \quad a_t = \underline{Q_t^{\text{Short}}}, \end{cases}$$

where M is a multiplier for the stop loss and take profit range. By adding multiplier, the threshold levels can be fine-tuned so that suitable range is found. After fine-tuning the parameter in this study, a multiplier of 1.5 was chosen.

Now that the stop loss and take profit levels are established dynamically based on ATR, the original reward function (Equation 13 can be updated. The default reward function remains the core of reward mechanism, but the agent is now penalized if the price crosses the stop loss level and rewarded if take profit level is crossed. The upgraded reward mechanism can be formed mathematically as follows:

$$r_t = \begin{cases} \Delta_v - R_{SL} & \textit{if} \quad Q_{t-1}^{\text{Long}} \quad \text{and} \quad p_t^C \leq \text{Stop Loss}, \\ \Delta_v + R_{TP} & \textit{if} \quad Q_{t-1}^{\text{Long}} \quad \text{and} \quad p_t^C \geq \text{Take Profit}, \\ \Delta_v - R_{SL} & \textit{if} \quad \underline{Q_{t-1}^{\text{Short}}} \quad \text{and} \quad p_t^C \geq \text{Stop Loss}, \\ \Delta_v + R_{TP} & \textit{if} \quad \underline{Q_{t-1}^{\text{Short}}} \quad \text{and} \quad p_t^C \leq \text{Take Profit}, \\ \Delta_v & \text{otherwise}, \end{cases}$$

where $\Delta_v = \dfrac{v_{t+1} - v_t}{v_t}$, R_{SL} is the extra penalty for triggering stop loss, and R_{TP} the extra reward for triggering take profit. $R_{SL} = 0.15$ and $R_{TP} = 0.15$ are are chosen for this study.

3.2.4 Model architecture and configuration

The architecture and configurations of the DQN is described in this section. The architecture developed by Théate and Ernst (2021) is adopted for this study, where a DNN structure is used for approximating the Q-function of the state and action. This DNN is a classical feedforward neural network, where the information is passed from input layer to output layer (see Figure 3). The input layer consists of six neurons, one for each variable (OHLCV, and portfolio weights). In addition to input and output layers, three fully connected hidden layers are used in DNN. First layer has 32 neurons, second has 64 neurons and Third has 128 neurons. In order to reduce overfitting, four batch normalization layers by Ioffe and Szegedy (2015) are applied to adjust and scale the activation functions. In simple terms, batch normalization normalizes the input of each layer in the batch to have zero mean and unit variance. In addition to batch normalization layers, four dropout layers are also added. Likewise to batch normalization layers, dropout layers also reduce overfitting by randomly setting to zero ("dropping out") a fraction of the input units during training. Finally, the output of the DNN is mapped to two output neurons on the output layer: the actions $\underline{N_t^{\text{Short}}}$ and N_t^{Long}.

This model also includes experience replay buffer introduced by Liu and Zou (2018). In the training process, a random batch with a predefined batch size is taken from experience replay buffer to speed up the learning. As an optimizer algorithm, Adaptive Moment Estimation or more friendly, "Adam", is employed. Introduced by Kingma and Ba (2014), Adam is "an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments". Adam dynamically adapts learning rates for individual parameters, solving challenges that varying gradient scales creates. In order to execute a robust exploration-exploitation strategy, the model follows ϵ-greedy policy. The ϵ starting value is 1 (exploring only), but gradually decreases to close to 0 (exploiting only).

Théate and Ernst (2021) have made some modifications to classical DQN, which are explained. Firstly, to reduce overestamations, Double DQN approach (Hasselt, 2010) is utilized. Secondly, to evaluate the loss of the algorithm, Huber loss is used. A loss refers to the difference between the predicted Q-values and the target Q-values. Common ways to calculate the loss is a mean squared error (MSE) loss and mean absolute error (MAE). Compared to these two, Huber loss does not penalize large errors like MSE and is differentiable at 0 contrary to MAE, thus providing a smooth and stable training. Huber loss is mathematically expressed as follows:

$$H(x) = \begin{cases} \frac{1}{2}x^2 & \text{if} \quad |x| \leq 1, \\ |x| - \frac{1}{2} & \text{otherwise..} \end{cases} \tag{14}$$

Thirdly, wide range of smaller modifications are implemented: gradient clipping technique is used to prevent the gradients of the neural network's parameters from becoming too large during the training process that can lead to instability; instead of initializing deep neural network weights randomly, Xavier initialization is deployed to initialize the weights such that the variance of the activations are the same across every layer (Ng, n.d.); regularization techniques such as L2 regularization and Early Stopping are implemented to further reduce overfitting; and finally, the existing data is augmented with data augmentation techniques to introduce new trading data that slightly differs from the original data but possess the same financial phenomenon.

The Double DQN architecture used in this study is briefly explained. In a traditional Q-learning update, the target Q-value for a state-action pair is calculated as the sum of the immediate reward and the maximum estimated Q-value for the next state as per Equation 1. Contrary to classical DQN, Double DQN separates the choice of optimal action and the evaluation of its value. To do so, two Q-networks, Q^a and Q^b, are used. (Hasselt, 2010). Therefore, for example Q^a can now be used to select the best action and Q^b to evaluate the Q-value of that action. The roles of these networks are then switched iteratively. This method alters the Equation 1, as the target Q-value is calculated by using the another Q-network. The Q-function can now be formed as:

$$Q^{\pi}(s_t, a_t) \leftarrow (1-\alpha) \times Q(s_t, a_t) + \alpha\Big(R_{t+1} + \gamma \times Q^b\big(s_{t+1}, \text{argmax}_a\, Q^a(s_{t+1}, a_{\pi})\big)\Big),$$

where the role of Q^a is to select the action a that maximizes Q-value in the next state

s_{t+1}, and the role of Q^b is to assess the action by evaluating the Q-value of the action selected by Q^a in the next state S_{t+1}

Hyperparameter tuning is important part of training the DQN agent. In order to find optimal hyperparameters for DQN, a trial-and-error based approach is carried out. The performance of DQN is analyzed based on the evaluation metrics presented in the subsequent section. Following the fine-tuning process, the hyperparameters conducted in this empirical research are listed in Table 2.

Table 2: Description of hyperparameters

Hyperparameter	**Value**	**Hyperparameter**	**Value**
DNN hidden layers	3	learning rate (α)	$5e^{-5}$
DNN first hidden layer dimension	32	dropout value	0.3
DNN second hidden layer dimension	64	discount factor (γ)	0.9
DNN third hidden layer dimension	128	batch normalization layers	4
batch size	64	number of episodes	200
time window size	30	replay memory size	10000
ATR rolling window size	14	ATR Multiplier	1.5
Stop loss reward (R_{SL})	0.15	Take profit reward (R_{TP})	0.15

3.3 Evaluation metrics and benchmark strategies

Evaluation metrics and benchmark strategies play an important role in assessing the performance of modified DQN, and how the trading boundaries influence the trading strategy. First, the model performance is analyzed. This part focuses to assess whether the agent learns during training process and shows progressive results over iterations. Ideally, the learning process would be stable, and results would indicate slow but steady improvement in training statistics over time. The training statistics used in this study are ϵ-value, training loss, Q-values for each action, average reward per training episode, and Sharpe ratio over episodes for both training and testing period. The ϵ-value gives information on the exploration-exploiting strategy. The higher the ϵ-value is, the more likely the agent is to take random action ("exploring"), whereas lower ϵ-value results more likely in exploitation of previous information. The training loss refers to the measure of the difference between the predicted Q-values and the target Q-values, which the DNN attempts to minimize during the training process. Average reward per training episode is the average reward that the agent collects during one training episode. This is a efficient measure for analyzing whether the agent improves over time, as the average rewards should increase as the agent converges to optimal strategy. Same should be true for Sharpe ratio, which should develop over iterations. Sharpe ratio is depicted for both training and testing period, which allows to examine the performance during testing period also. The two should follow relatively close to each other. For example, if Sharpe ratio improves in training process, but falls greatly behind or fluctuates a lot in testing period, this would indicate overfitting. This is because the agent learns optimal policy in training data, but when applied to out-of-sample data, the learned policy is efficient only for training data and results in poor performance on new data.

To assess how the influence of trading boundaries change the results compared to original TDQN, average rewards per training episode are calculated for both TDQN and modified DQN, and these are mapped together visually to evaluate their development. In addition, statistics about triggering trading boundaries are included to aid in evaluation of how often the limits are triggered and what is their impact on the feedback received by the trading agent. Lastly, the trading activity is illustrated for environments without and with 0.2% transaction costs. The price development of the securities and capital development are depicted together with trading actions and limit triggers. This allows to further analyze the behavior of trading agent visually, possibly indicating in which scenarios does the agent excel and which not, how does transaction costs affect the activity, and providing information how and when are trading limits triggered.

Second, the proposed approach is benchmarked against multiple alternative trading strategies. Benchmark strategies give a reference how the proposed trading strategy's performance bear in comparison with the benchmark. Incorporating these benchmark strategies offers a valuable perspective on its effectiveness compared to more traditional trading strategies. The modified approach in this study is compared to following benchmarks:

- **Buy & Hold**: this strategy simply buys the stock in the beginning and holds the initial portfolio until the end. Therefore, no investments actions are done after the initial purchase.

- **Random strategy**: the strategy incorporates random actions (buy or sell) at each state. This strategy is considered to benchmark arbirarily chosen trading actions.

- **Trend following with moving averages (TF)**: the strategy employs an analysis of moving averages to identify and follow the prevailing trend of stock. It involves the buying of stocks whose value has increased in the previous period and vice versa.

- **Mean reversion with moving averages (MR)**: the strategy assumes that asset prices will revert to their historical average or mean. Thus, stocks are bought when below the average and sold when above.
- **TDQN**: the initial version of DQN of this study, originally developed by Théate and Ernst (2021).

Wide range of different evaluation metrics for are used to quantify the results of trading strategies mentioned above. The initial portfolio value is set to €$100,000$, and the results are calculated for both without transaction costs and transaction costs of 0.2%. Financial metrics are then used to evaluate the monetary gains or losses, risks, profitability, and distribution of the returns in the strategy. The following metrics are calculated for each trading strategy:

- **Cumulative Return (%)**: the gains or losses relative to original portfolio value. Indicates how much the trading strategy has yielded in percentages.
- **Standard deviation**: the annualized degree of variation of returns. Indicates the level of risk or price fluctuation in the trading strategy.
- **Sharpe Ratio**: risk-adjusted return. Evaluates the excess return of the strategy per unit of risk.
- **Maximum Drawdown**: largest peak-to-trough decline in the portfolio value in percentages. Focuses on capital preservation and risk management.
- **Profitability**: percentage of profitable trades. Measures the proportion of trades that resulted in a profit.
- **Skewness**: the distribution of trading yields. Measures the degree and direction of skewness in the distribution of returns.

Finally, the capital development for testing period is illustrated for all strategies. This allows to examine visually how the strategies have performed in certain periods and compare each strategy's development relative to other strategies. This can be used to determine which trading strategy has outperformed others, and which strategy has resulted in the poorest performance.

4 RESULTS

4.1 Model evaluation

The empirical research was conducted on three Finnish stocks; Nordea, Sampo, and Bittium; and a Finnish stock index OMX Helsinki PI to evaluate whether the novel DQN trading strategy with stop loss and take profit levels can generate excess returns compared to its benchmarks. The DQN algorithm was trained between 1^{st} of January 2015 and 31^{st} of December 2021, and tested on a shorter testing period from 1^{st} of January 2022 to 31^{st} of December 2023. Evaluating the performance of proposed DQN is essential for understanding how well the algorithm is learning and making decisions in the environment. As the goal of the agent is to learn from the past price movements and exploit that information for profitable trading strategy, it is necessary to investigate that the agent is not solely performing random actions. To analyse that, various metrics are calculated to assess the performance and behavior of the agent during the learning process.

First, to handle the exploration-exploitation trade-off, ϵ-greedy policy was used. To illustrate this, the decaying of ϵ-value is depicted in Figure 6. As the epsilon is decayed over the iterations using exponential function, the decaying begins rapidly, but as the number of iterations increases, the rate of decaying slows down. In the beginning, the agent explores relatively often, but towards the end, the agent is exploiting the gathered information on every step. This is desired progress for the agent, as it is allowed to explore and find information about the environment, and later exploit that information for finding optimal policy.

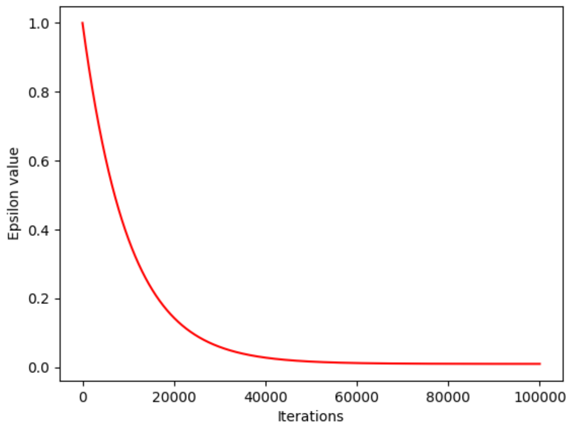
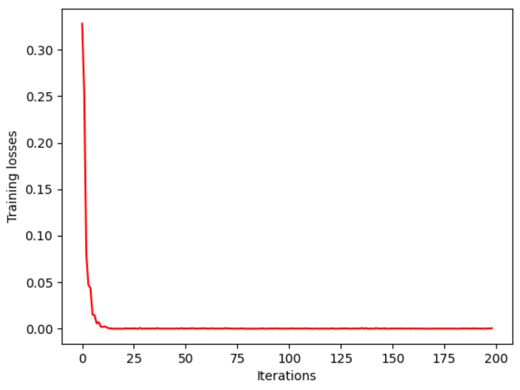
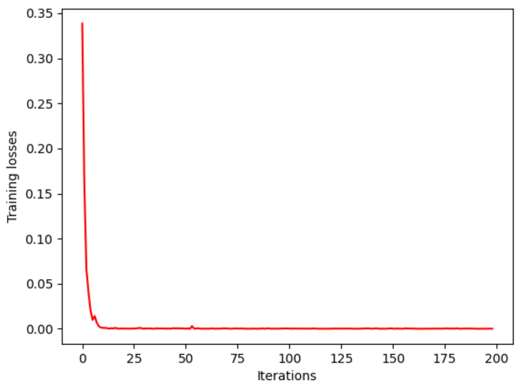


Figure 6: Decaying of ϵ in training process

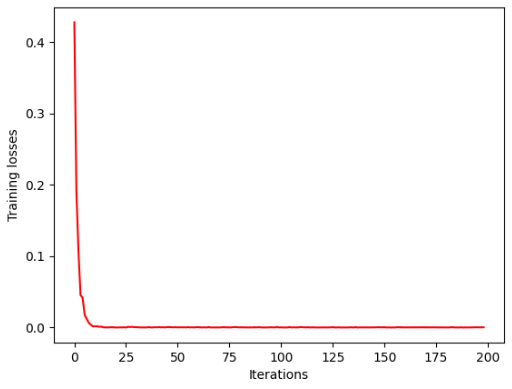
Second, it is evaluated how the losses calculated in the training process develop over training episodes. This metric measures how well the DQN is fitting the training data during the learning process. The loss curve that indicate good model behavior should be decreasing, and not include sudden spikes or jumps. The curve should plateau and stop decreasing at some point. In Figure 7, the training losses are visualized for all securities in trading environment with transaction costs. As we can see from the figure, the losses calculated over episodes for all securities are very similar to each other. They all indicate that at first the losses start from relatively high, but through learning process, the losses decrease rather quickly. The curve seems to plateau at around episode 15 which means that it reaches its plateau relative quickly. Overall, it seems that the agent learns to minimize the loss for all separate securities when iterating over episodes, and the visualization indicates effectively functioning learning process.



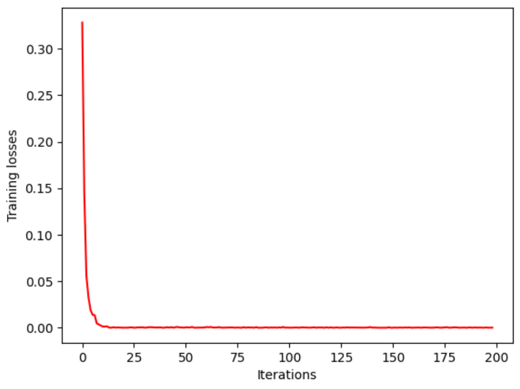
(a) Nordea



(b) Sampo



(c) Bittium



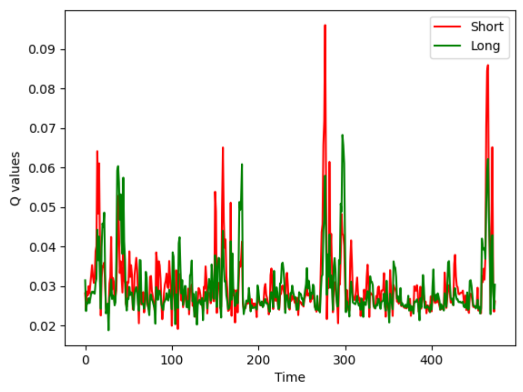
(d) OMX Helsinki PI

Figure 7: Training loss during the training process with transaction costs

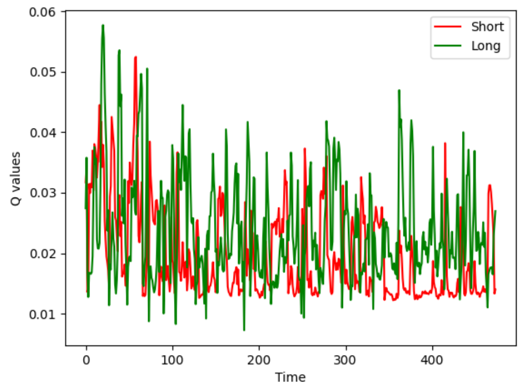
Third, to further analyze the behavior of the trading agent, Q-values in given states can be plotted for each action. Plotting Q-values for short and long actions can provide valuable insights about the agent's decision-making process. As the Q-values represent the expected cumulative reward for taking a particular action in a given state, analyzing which action the agent is more willing to take in different states help understand the behavior of the agent. In addition, it provides awareness about the distribution of Q-values. The distribution can indicate if the agent is favoring one type of action over the other. The visualization also helps to understand the learning process of the agent. As training progresses, the Q-values should converge towards the true optimal Q-values. This would indicate that the agent is learning a good approximation of the optimal policy. The Q-values should also be relative smooth and not include big sudden spikes. In Figure 8 and 9, the Q-values of short and long actions are plotted for original TDQN and the modified DQN with transaction costs. By observing the graph, it can be seen that the Q-values behave relatively similarly for all of the securities. It can be noted that the values seems to fluctuate around zero over the whole training process, but sometimes spiking upwards for both short and long actions. This high volatility in Q-values indicates instability and issues in the training process of the DQN agent. While chosen hyperparameters do affect the Q-values, it was observed that the Q-values swung similarly during the trial-and-error optimization process. These results raise the question whether financial markets are too complex for DQN, making the training process unstable.

Another observation is that the both actions receive similar values in given states. As a matter of fact, the values are often very close to each other. This could be interpreted that the difference of values between choosing short or long action is very little, meaning that there does not exist clear optimal action for specific state. This could lead to random choices of action. Naturally, this is against the idea of DQN converging to optimal policy.

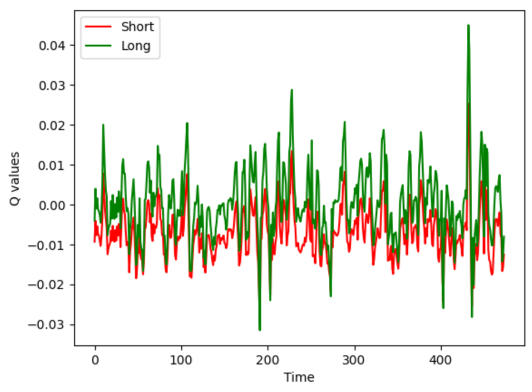
Comparing the Q-values of modified DQN to the original TDQN, it seems that while the original TDQN has also fluctuation, the introduction of stop loss and take profit levels does increase the instability in Q-values. As the levels provide additional and relative big reward when crossed, this might cause the Q-values to change rapidly. Additionally, the fact that the limits are usually crossed when the underlying price has high volatility when predicting is more difficult, may further contribute to the fluctuation. Consequently, instead of guiding the trading agent, using the trading limits within reward function may cause even more randomness to already complex trading environment.



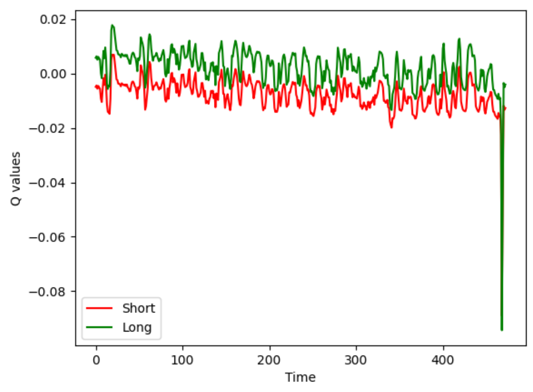
(a) Nordea



(b) Sampo

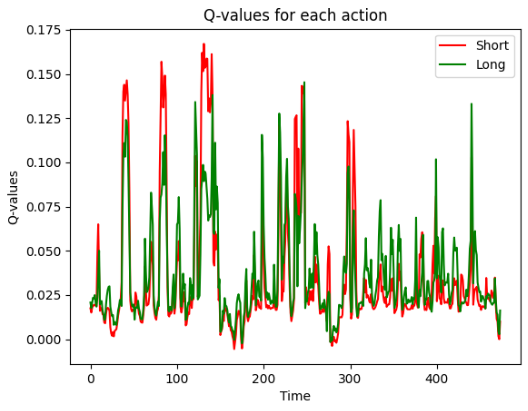


(c) Bittium

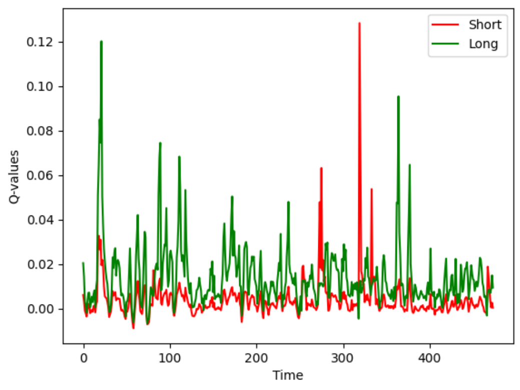


(d) OMX Helsinki PI

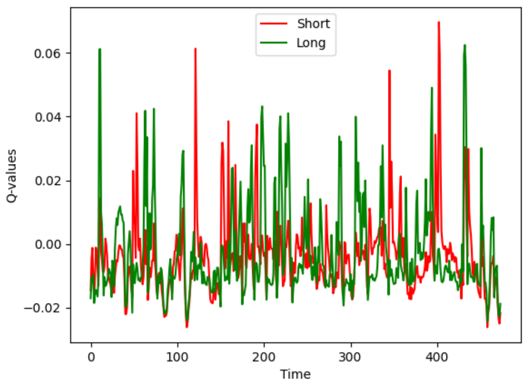
Figure 8: Q-values for each action during the training process with transaction costs for original TDQN



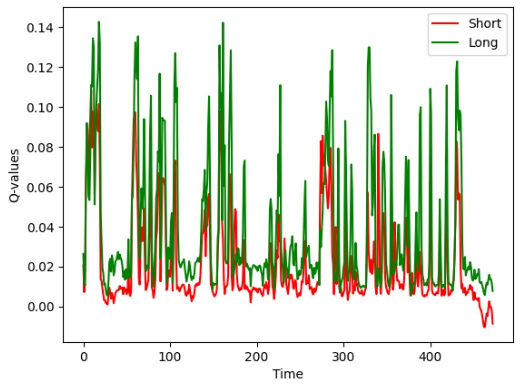
(a) Nordea



(b) Sampo



(c) Bittium



(d) OMX Helsinki PI

Figure 9: Q-values for each action during the training process with transaction costs for modified DQN

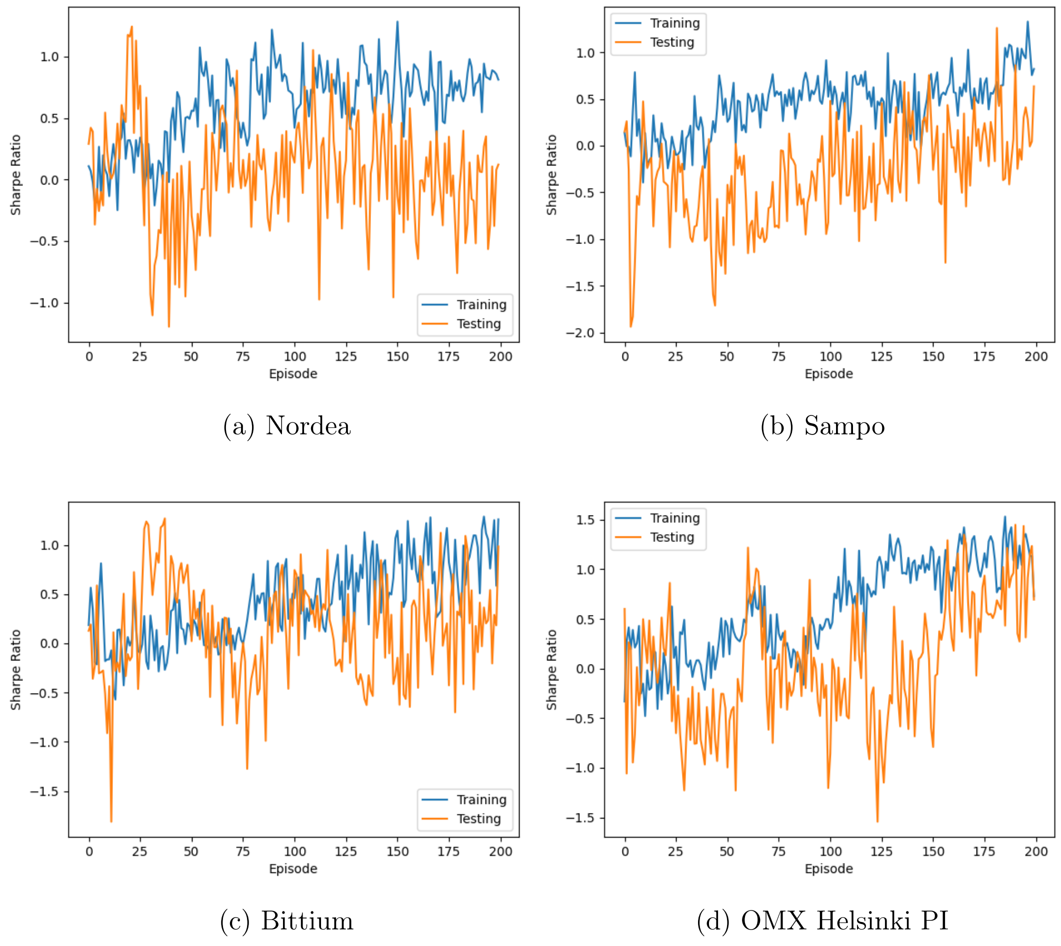
Fourth, Sharpe ratio that represents the risk-adjusted return is analyzed for both training period and testing period. To conduct this, Sharpe ratio is tracked for each episode on training and testing period. Sharpe ratio should increase gradually, as the agent learns towards the optimal policy. In addition, by comparing the Sharpe ratio between training and testing periods, it can be evaluated how well the DQN generalizes to out-of-sample data. Given that Sharpe ratio in testing period remains consistent with Sharpe ratio from training period, it suggests that the agent has learned meaningful patterns from the training data and can perform well in new, unseen environment. The comparison can also help detecting overfitting: if Sharpe ratio decreases significantly from training period to testing period, it might indicate overfitting, where the agent learns to memorize the training data rather than understanding the pattern in the data.

When an overfitted DQN is applied to out-of-sample data, it performs poorly as it has failed to learn generally in training period.

Overall, an upward trend for Sharpe ratios during training period can be detected for all securities without transaction costs in Figure 10. Despite the ratios fluctuating, the general trend is upwards, indicating that the agent is able to generate higher Sharpe ratios over training period. This signals that the agent is learning over time. However, Sharpe ratios for testing periods for all securities seem to be much more volatile than during training period. While it is normal that the results may fluctuate more for out-of-sample data, a big contrast in volatility between the periods means that the agent is overfitting and performing poorly on out-of-sample data. For Sharpe ratios of testing period, there are not similar upward trend except for Sampo (Figure 10b) in which the ratio deteriorates compared to training period, but improves over time. For other securities, the ratios are very volatile and do not show any improvement over time, although Sharpe ratio in Figure 10d for OMX Helsinki PI finishes high, close to training period Sharpe. These results imply that whilst the performance for training period is good, the DQN struggles with out-of-sample data. This shows overfitting in the DQN, despite methods used to avoid it.

Sharpe ratios for the environment with 0.2% transaction costs are illustrated in Figure 11. By analyzing the results, it can be seen that the results vary among the securities. Identical for all securities are that Sharpe ratios start very low and drops steeply to negative values in the beginning, but starts to recover afterwards. This behavior could be explained by the fact that in the beginning, the agent is exploring the environment, thus taking non-optimal actions. Over time, the agent stops exploring and starts to exploit the acquired information. This can be seen as an increase in Sharpe ratios over episodes. However, after this, the results starts to vary depending on the security. Another observation is that similar to the environment without transaction costs, Sharpe ratio for testing period is more volatile than for training period. The development for Nordea seems to be the most stable (Figure 11a). After the initial drop and recover in Sharpe ratio, the ratios seem to swing up and down slightly above zero. At around episode 120, there is a big spike downwards in Sharpe ratio for testing period. Overall, the ratios are relatively stable for Nordea, but do not indicate any upward trend over episodes. For Sampo (Figure 11b), Sharpe ratio during training period is similar to Nordea, but Sharpe ratio for testing period is more volatile and reaches lower values. Training period Sharpe ratio seems to vary the most for Bittium (Figure 11c) in which the ratio decreases slowly and then increasing back slowly. The corresponding testing ratio follows the development, although reaching lower Sharpe ratios. Finally, in Figure 11d, OMX Helsinki PI shows slight improvement in training ratio over time, but the ratio for testing period fluctuates significantly and does not perform as well as during training period.

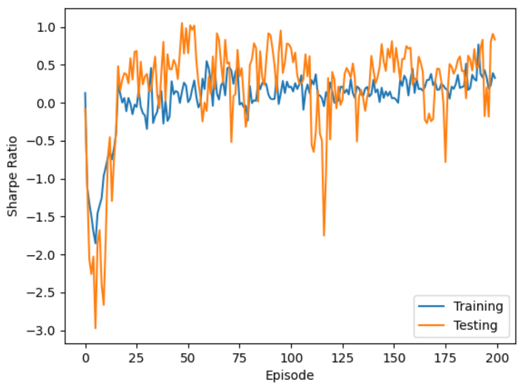
In conclusion, Sharpe ratio graphs suggest good performance for training period without transaction costs as Sharpe ratios rise incrementally, but other than that, the performance of DQN is relatively poor. Similar advancement can not be observed for other plots which show that the agent struggles to learn any exploitable information. In addition, many of the graphs suggest overfitting as the performance of Sharpe ratio is much worse compared to Sharpe ratio during training period. Although many methods were used to reduce overfitting, such as batch normalization, dropout, L2 regularisation, and double Q-network architecture, the DQN still has troubles to generalize to out-of-sample data. According to Sharpe ratio graph, the DQN does not seem to be very robust to generate stable and consistent returns.



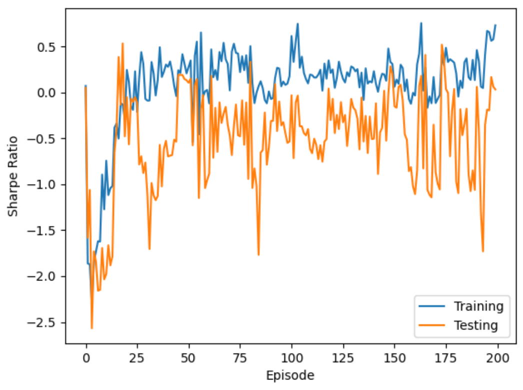
(a) Nordea (b) Sampo

(c) Bittium (d) OMX Helsinki PI

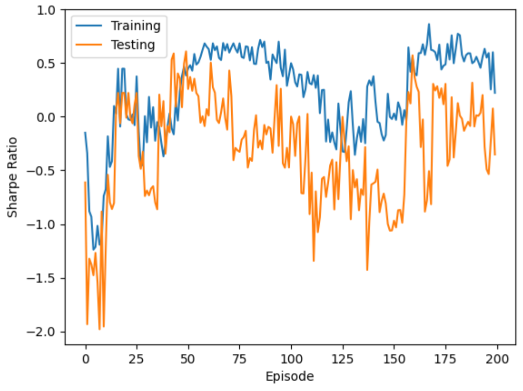
Figure 10: Sharpe ratio during training and testing episodes without transaction costs



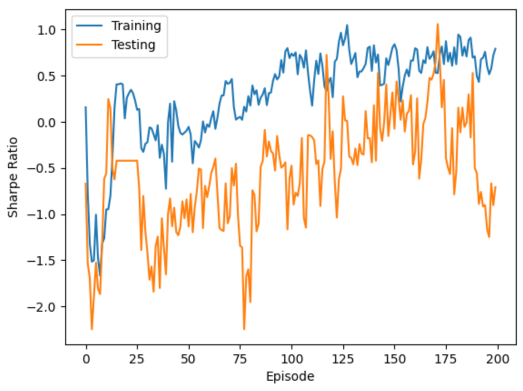
(a) Nordea



(b) Sampo



(c) Bittium



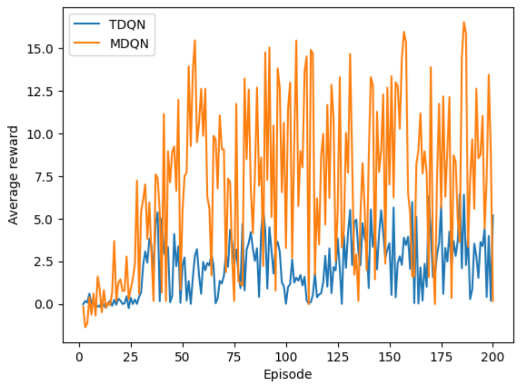
(d) OMX Helsinki PI

Figure 11: Sharpe ratio during training and testing episodes with transaction costs

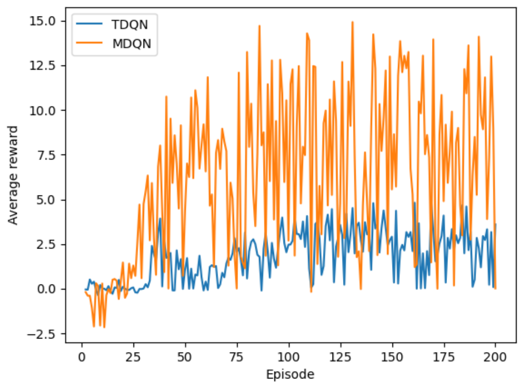
Fifth, it is studied how the average rewards develop over iterations. Reward development offers valuable insights into the agent's ability to maximize rewards in the given environment As the agent attempts to maximize rewards, increasing rewards indicate that DQN learns within the environment and performs better in the trading task over time. To track learning, average reward per episode for all training episodes are calculated. A steadily increasing average reward indicates that the DQN improves over time as it is able to generate bigger rewards (i.e., bigger trading profits) by exploiting its previous knowledge. Comparing the development of rewards for modified DQN and original TDQN by Théate and Ernst (2021) is great way to analyze how adding the trading limits affects the behavior of the trading agent. The average rewards per episode for all securities are depicted in Figure 12 without transaction costs and in Figure 13 with transaction costs of 0.2%.

The development is similar for both approaches and for all securities. First, the average rewards begin low, receiving values close to zero and negative values. When the training process advances, the average rewards start to increase gradually. This is desired development of the average rewards as it indicates that the agent is learning to generate higher and higher rewards over time. However after a certain point, the rewards start to jump up and down. It seems that the rewards settle for higher level, but incorporates extreme fluctuation between around zero value and higher values. The achieved rewards are higher for modified DQN than for original DQN, but also makes the reward fluctuation greater. This suggests similar results as in Q-values: using trading limits to provide one-off rewards in extreme cases (price rises or drops significantly) further contributes to variation in model performance. Additionally, as the trading limits are set dynamically and are not set to any fixed levels, it could create more arbitrariness to the feedback received by the trading agent. For example, given that the agent would be able to learn the meaning of trading limits, the fact that one time the limits could trigger from small drop and other time not, could confuse the trading agent to a greater degree.

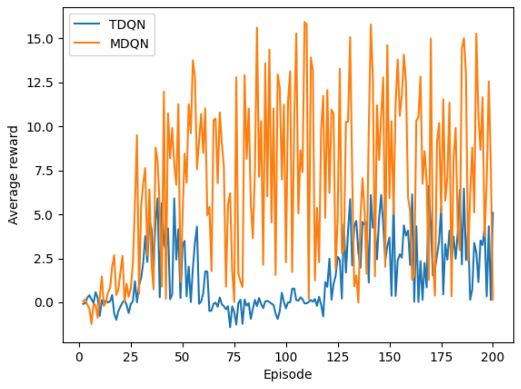
The figures indicate large instability in training process: in one episode, the agent may be able to achieve very large rewards while in another episode the rewards remain close to zero. This does not suggest that the algorithm would be robust for consistently generating excess returns. There exists some reasons for the results obtained. Firstly, the non-stationary and complexity of stock market may cause the model not to be robust enough. If random walk of stock market is presumed, the rewards signals the agent is receiving is rather arbitrary. Therefore, the agent is not able to converge to any optimal policy. This would reason the high volatility in performance as sometimes the agent performs very well and other times very poorly. Secondly, artificial neural networks are known to be prune to "catastrophic forgetting", where the neural network loses information about previously learned tasks as information about current task is incorporated (Kirkpatrick et al., 2017). In the figure, we can see that the average reward drops as if the agent forgets its previously learned information. However, while this could be the case, the jumping values up and down does not explicitly indicate that the agent would forget the information, as it jumps back up to where it was. Therefore, it is more likely that the model is just not robust enough to function well in the environment.



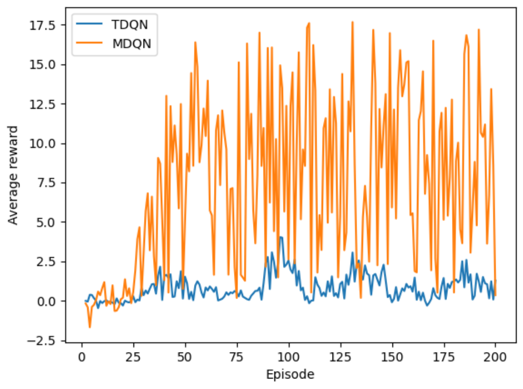
(a) Nordea



(b) Sampo

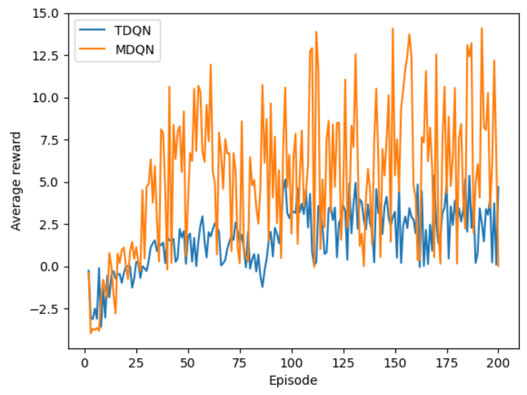


(c) Bittium

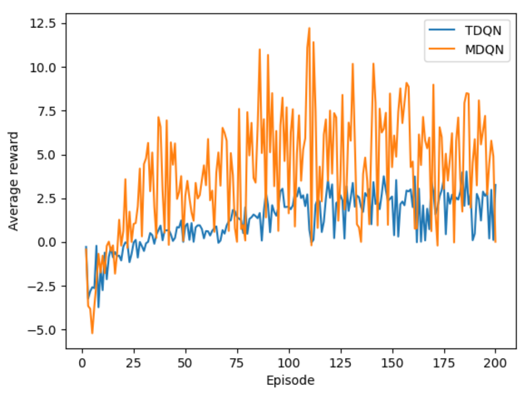


(d) OMX Helsinki PI

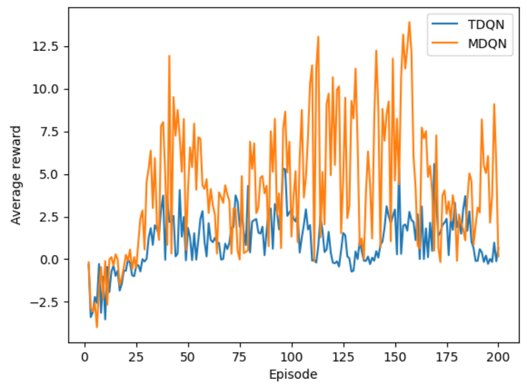
Figure 12: Average rewards over training episodes for all securities without transaction costs



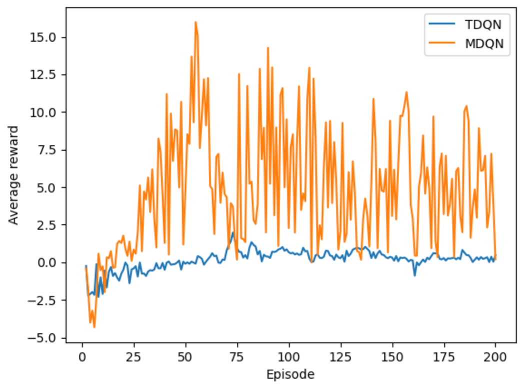
(a) Nordea



(b) Sampo



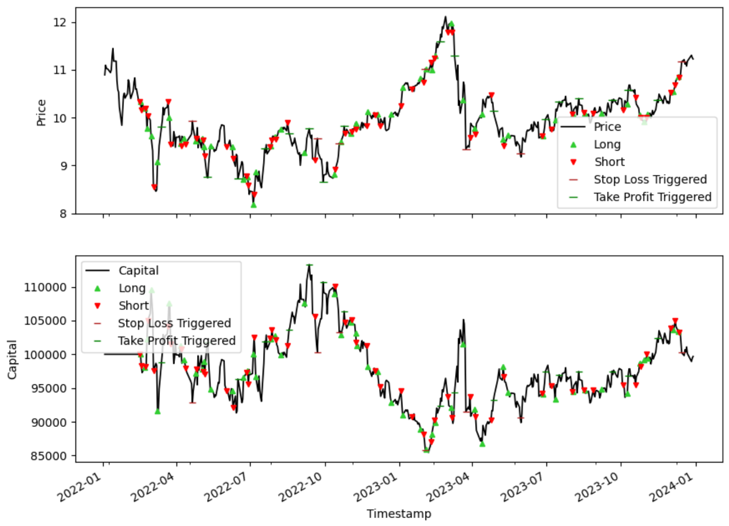
(c) Bittium



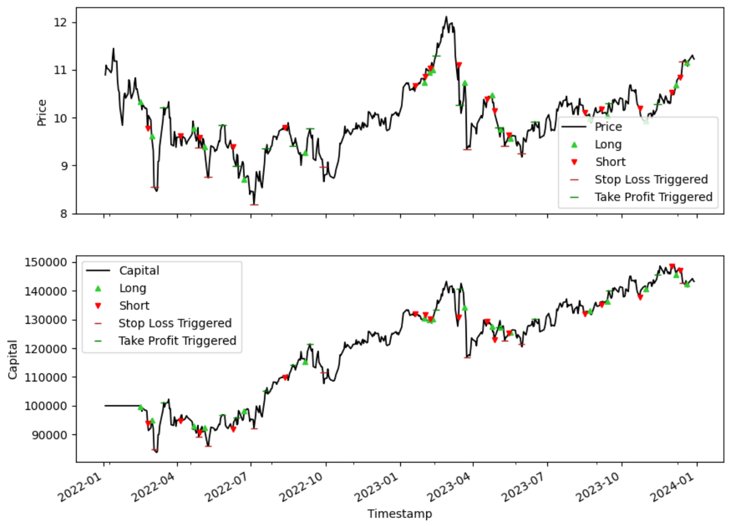
(d) OMX Helsinki PI

Figure 13: Average rewards over training episodes for all securities with 0.2% transaction costs

Next, the trading activity and performance of the proposed DQN is illustrated. The development of portfolio value is visualized together with security price development, and to examine the behavior of the DQN, the actions (long and short) and the trading limits triggered (stop loss and take profit) are also depicted in the figure. This allows to investigate what kind of actions are taken in certain states and how has the trading limits been triggered. Figures 14-17 show the results of trading activity for the securities. The trading limits does not seem to dictate when to change trading position: in many cases, the agent hold same position despite crossing stop loss or take profit level.

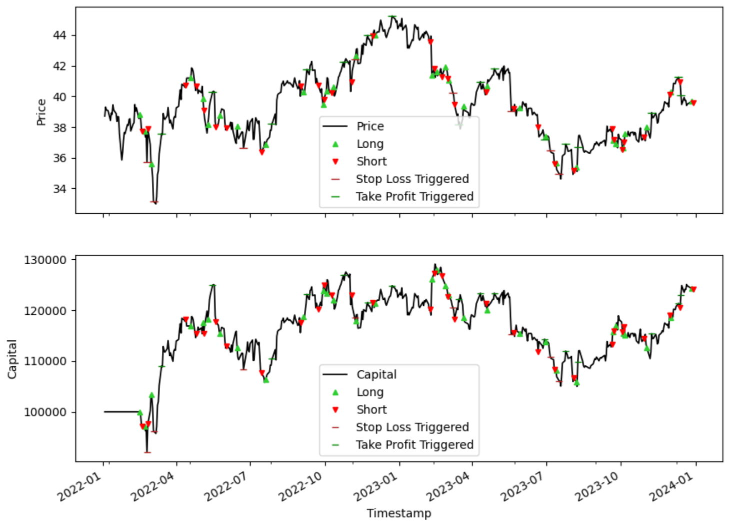


(a) Transaction costs = 0%

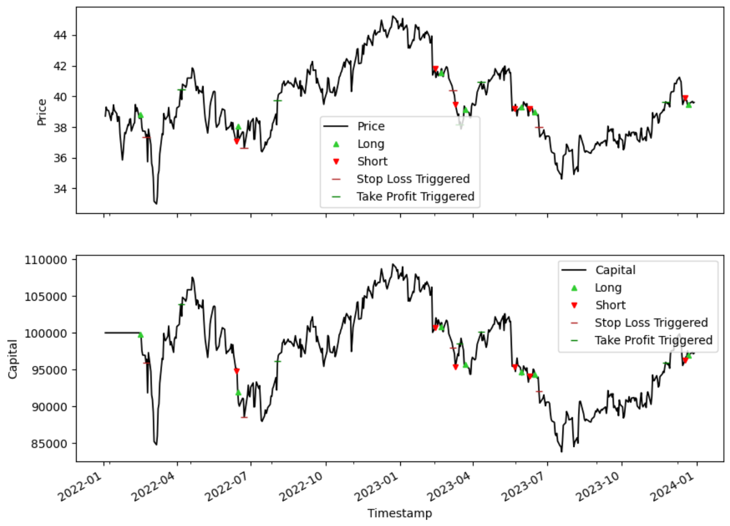


(b) Transaction costs = 0.2%

Figure 14: Trading activity for Nordea

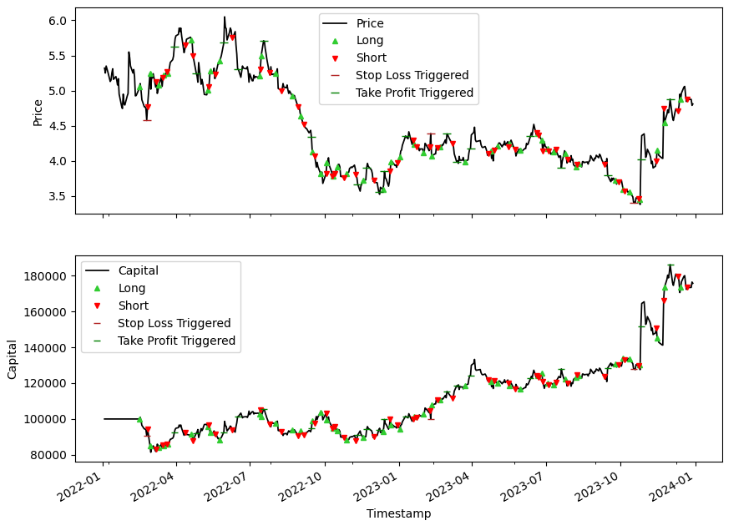


(a) Transaction costs = 0%



(b) Transaction costs = 0.2%

Figure 15: Trading activity for Sampo


(a) Transaction costs = 0%

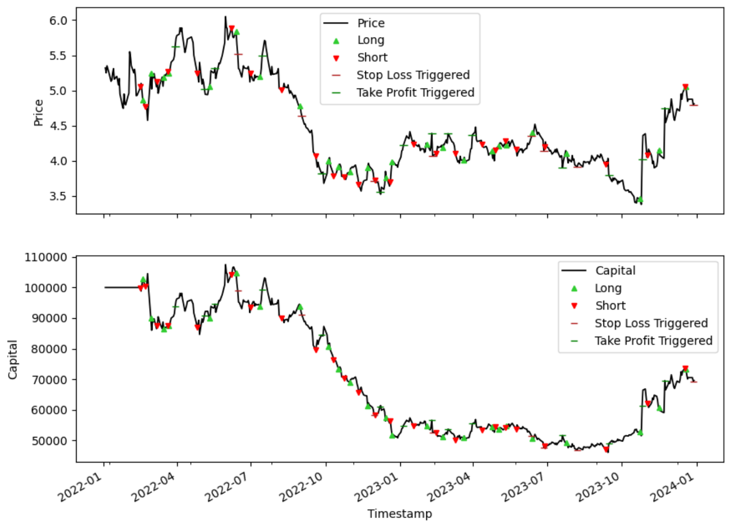
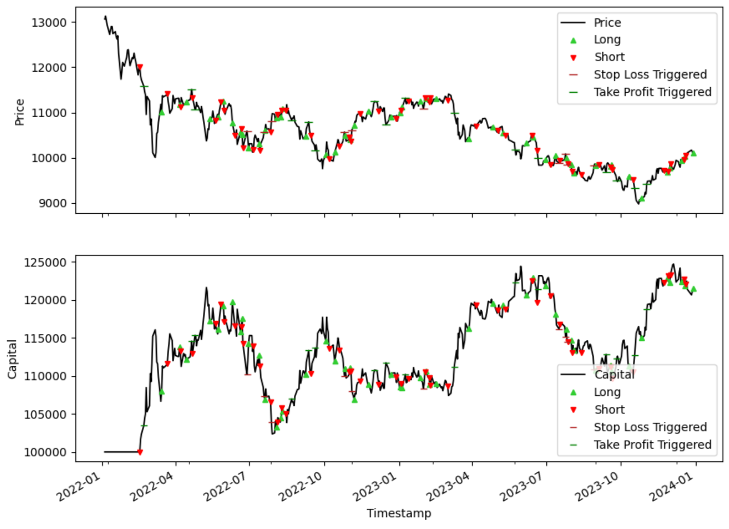

(b) Transaction costs = 0.2%

Figure 16: Trading activity for Bittium


(a) Transaction costs = 0%

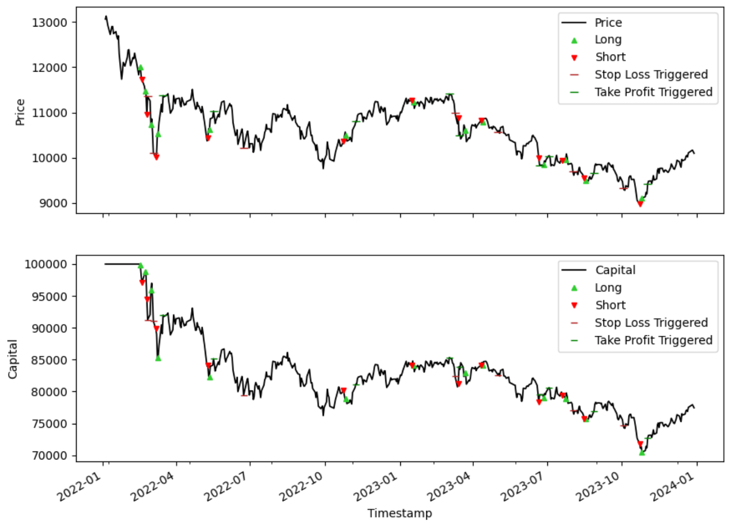

(b) Transaction costs = 0.2%

Figure 17: Trading activity for OMX Helsinki PI

In addition to depicting the trading activity, the agent behavior is further analyzed by studying how the agent trades in different market environments. In this study, two market regimes are set: low volatility regime and high volatility regime. The market is set to high volatility regime if the 14-day moving average of volatility is over 2.5%. The window is set to same as the ATR time window for the sake of uniformity. In Tables 3 and 4, statistics show how the trading behavior realizes in each regime. The regime count tells how many days of the trading period is defined as low volatility regime and how many days are set as high volatility regime. Most of the trading days belong to low volatility regime, but there are slight differences between securities. The trading position change count indicates how many trades have taken place in each regime. In some cases, only a few trades or none are executed on high volatility regime, resulting in situation where most of the statistics can not be calculated (shown as NA in the tables).

To study the behavior of trading agent in different regimes, an important statistic is the mean trade duration within each regime. By comparing this statistic, it can be evaluated if the behavior changes much in times of market stress. For example, it might be worthwhile to examine if the trading agent changes trading position in more volatile regimes. When comparing the means, it seems that the mean trade durations are relatively close to each other for Nordea (6.90 and 6.69) in an environment without transaction costs, but deviation can be detected for Bittium (8.05 and 6.10) in both environments. To evaluate the statistical significance of equal means, first the data sets are tested for normality using Shapiro-Wilk test. As the normality test indicates no normal distribution for all data sets, Mann-Whittney U test is used for equal mean test. Although the test can only be conducted on Nordea and Bittium (no transaction costs) and Bittium (transaction costs), the results indicate that on a 0.05 confidence level, there is statistical evidence that the means are not statistically different. Therefore, it can be stated that based on mean trade duration, the trading agents behavior does not change significantly between low and high volatility market regimes.

In addition to mean trade duration, there are some other observations that can be made. Firstly, the trading frequency decreases significantly when transaction costs are considered. This drop in trading position change count is expected as the agent is penalized slightly for taking an action because the transaction costs reduce the reward the agent receives. Théate and Ernst (2021) report similar results in their research. Secondly, both the mean and maximum trade duration seem to increase when comparing the two environments. This is in line with the illustration in Figures 14-17, where the trading frequency seem to decrease when transaction costs are considered. This consequently results in longer trade durations. Thirdly, with transaction costs, the standard deviations generally increase which suggests greater variability in trade durations compared to scenarios without fees.

Table 3: Statistics for trading behavior without transaction costs, duration in days

	Securities			
	Nordea	**Sampo**	**Bittium**	**OMX Helsinki PI**
Low Volatility Regime				
Regime count #	448	492	418	503
Trading position change count #	90	64	74	98
Mean trade duration	6.90	11.33	8.05	7.39
Standard deviation of trade duration	8.28	13.85	8.06	7.50
Maximum trade duration	18	68	43	43
25% quartile of trade duration	1.0	3.0	3.0	1.25
75% quartile of trade duration	10.0	13.25	10.0	8.75
High Volatility Regime				
Regime count #	56	12	86	0
Trading position change count #	13	0	20	0
Mean trade duration	6.69	NA	6.10	NA
Standard deviation of trade duration	5.41	NA	6.46	NA
Maximum trade duration	43	NA	21	NA
25% quartile of trade duration	2.0	NA	1.0	NA
75% quartile of trade duration	9.0	NA	8.0	NA
Mann-Whitney U test p-value	0.508	NA	0.164	NA

Table 4: Statistics for trading behavior with transaction costs, duration in days

	Securities			
	Nordea	**Sampo**	**Bittium**	**OMX Helsinki PI**
Low Volatility Regime				
Regime count #	453	493	422	497
Trading position change count #	34	13	41	25
Mean trade duration	20.85	55.23	15.10	26.4
Standard deviation of trade duration	27.70	78.46	11.98	38.66
Maximum trade duration	136	243	48	165
25% quartile of trade duration	6.0	7.0	6.0	2.0
75% quartile of trade duration	30.25	61.0	20.0	43.0
High Volatility Regime				
Regime count #	51	11	82	6
Trading position change count #	1	0	9	0
Mean trade duration	7.00	NA	10.67	NA
Standard deviation of trade duration	NA	NA	7.04	NA
Maximum trade duration	7	NA	27	NA
25% quartile of trade duration	7.0	NA	8.0	NA
75% quartile of trade duration	7.0	NA	11.0	NA
Mann-Whitney U test p-value	NA	NA	0.426	NA

Finally, it is informative to study statistics about the stop loss and take profit levels. In order to grasp the effect of the trading limits, descriptive statistics are presented in Table 5 for trading without transaction costs and in Table 6 for trading with trading costs. When analyzing the results for triggers in the environment without transaction costs, it can be seen that in general, take profit levels are triggered more often than stop loss levels for both positions. This results in positive net rewards for all securities, being the highest for Bittium (2.40 and 0.90) for both environments and the lowest for OMX Helsinki PI (0.90) without transaction costs and for Sampo (0.15) with transaction costs. Compared to average rewards presented earlier, the amount of additional reward seems to be proportional, not too high to solely compose reward and not too low to not have any impact. From average reward graphs, it was seen that the limits did have an impact on average reward fluctuation. Additionally, it can be observed that the frequencies are higher for long position than short position, possibly indicating that the agent holds long position more often than short position. As noted before for Sampo and OMX Helsinki PI with transaction costs, the agent preferred long position, and when changing to short position, the agent quickly switched back to long position. This explains why counts for Sampo and OMX Helsinki PI in Table 6 for short position are very low. As the both limits can be triggered during one position, this probably explains why counts of stop loss and take profit triggers for Sampo and Helsinki PI with transaction costs are very close to each other. As long position is held for long time, usually the price fluctuates so that both levels are triggered at some point.

When comparing the statistics between the environment without and with transaction costs, it seems that the frequencies decline when transaction costs are acknowledged except for long position stop loss triggers. Especially the limit triggers for short position in an environment with transaction costs are quite low. As discovered in analysis of trading activity, the trading frequency declined when transaction costs were taken into account. This most likely correlates to amount of triggering limits, as changing the position sets new stop loss and take profit levels that then can be crossed. If trading occurs scarcely, even if trading boundaries are crossed, new limits are not set until the change of trading position. Although not a sole reason, the elevated stop loss count for long position in an environment with trading fees compared to without transaction costs could be because of the agent holds long positions longer than in an environment without transaction costs.

Table 5: Descriptive statistics for stop loss and take profit limits without transaction costs ($R_{SL} = 0.15$ and $R_{TP} = 0.15$)

	Securities			
	Nordea	**Sampo**	**Bittium**	**OMX Helsinki PI**
Long Position				
Stop loss triggered #	2	7	2	3
Take profit triggered #	11	13	11	5
Total penalty from stop losses	0.30	1.05	0.30	0.45
Total reward from take profits	1.65	1.95	1.65	0.75
Short Position				
Stop loss triggered #	5	1	1	7
Take profit triggered #	6	3	8	11
Total penalty from stop losses	0.75	0.15	0.15	1.05
Total reward from take profits	0.90	0.45	1.20	1.65
Net reward from triggering limits	1.50	1.20	2.40	0.90

Table 6: Descriptive statistics for stop loss and take profit limits with transaction costs of 0.2% ($R_{SL} = 0.15$ and $R_{TP} = 0.15$)

	Securities			
	Nordea	**Sampo**	**Bittium**	**OMX Helsinki PI**
Long Position				
Stop loss triggered #	8	4	7	6
Take profit triggered #	8	4	9	7
Total penalty from stop losses	1.20	0.60	1.05	0.90
Total reward from take profits	1.20	0.60	1.35	1.05
Short Position				
Stop loss triggered #	1	0	1	1
Take profit triggered #	4	1	5	2
Total penalty from stop losses	0.15	0.00	0.15	0.15
Total reward from take profits	0.60	0.15	0.75	0.30
Net reward from triggering limits	0.45	0.15	0.90	0.30

4.2 Benchmarking

Various metrics and benchmark strategies are used to evaluate the modified DQN with trading limits used in this study during the testing period of 1st of January 2022 to 31st of December 2023. The evaluation metrics for all trading strategies are presented in Table 7 without transaction costs and in table 8 with 0.2% transaction costs. The results for strategy performances without transactions costs are analyzed first, and then performances with including transaction costs are examined afterwards.

First, it can be noted that the selected securities have performed relatively poorly during the testing period in terms of price development. In the Buy & Hold strategy, the stock is bought in the beginning and held until end, thus reflecting the statistics of underlying stock price development. For Nordea, the results show modest gain of 3.05% on cumulative return over the testing period, and similarly 2.31% for Sampo. For small cap stock Bittium, the cumulative return displays a decline of -9.59% in capital over the testing period. While the three individual stocks show relatively weak returns, the broader market index OMX Helsinki PI falls short even compared to the individual stocks. The cumulative return for OMX Helsinki PI is -20.77% for the testing period. Looking at the descriptive statistics, it can be detected that Bittium is the most volatile of the securities, with 36.85% standard deviation and maximum drawdown of 44.13%, whereas ranging from 16% to 27% and from 23% to 29% for other securities respectively.

Analyzing the Table 7 where the trading strategies' performance is assembled without transaction costs, the modified DQN suggests strong performance against the benchmark strategies except for Nordea. The cumulative return for this stock is the third lowest, only beating the original TDQN and TF. The return on modified DQN falls short slightly compared to B&H. However, for other stocks, the proposed DQN approach with trading limits shows very strong performance. Trading on Sampo returns around 10% more on cumulative return compared to B&H, although losing to MR by 4.77% in terms of cumulative returns, but outperforms even that when risk-adjusted returns are considered. The performance for original TDQN is very poor, -22.58%, whereas all other strategies report positive cumulative returns. Looking at Bittium, the modified DQN generates an astonishing cumulative returns of 75.72% with Sharpe ratio of 0.983 while the B&H returns -9.59% on cumulative returns. The trading activity for Bittium showed steady increase in trading capital, but also jumping up towards the end of period due to holding long position when the security price jumps up. Looking at the skewness for the proposed approach, it shows that the trading yields are very skewed towards the positive yields. This can also be seen from the fact that despite the trading profitability (share of profitable trades of all trades) being in line with other strategies, the cumulative returns is much higher. This means that although making profitable trades is as common as for other strategies, each trade produces much higher profits compared to benchmark strategies. The original TDQN performs well for Bittium stock likewise, producing cumulative returns of 15.40% with Sharpe ratio of 0.375. Finally, for the broader market index, the modified DQN outperforms the benchmark with cumulative return of 21.51% with Sharpe ratio of 0.695. Only TF is able to generate positive returns together with modified DQN. Although the original TDQN does not perform as well as these two strategies, as the cumulative return is −13.82%, it does perform better than the B&H strategy.

Next, the performance metrics for trading environment with transaction costs of 0.2% can be seen in Table 8. Comparing the statistics of B&H to the environment without transaction costs, it can be observed that the values are very close to each other. This is intuitive, as the shares are only bought once in the beginning of the trading period and thus the impact of transaction costs is very minimal. In contrary to previous results, the modified DQN performs relatively poorly compared to benchmark when transaction costs are taken into account, except for Nordea stock where the approach generates profits well beyond other strategies. For Nordea, the approach generates impressive cumulative returns of 43.24% with Sharpe of 0.833, surpassing B&H strategy by over 40 percentages. All of the other strategies yield returns deep in negatives, and the capital for TDQN being decayed as much as −55.78%. Despite showing promising performance on Nordea, the modified DQN underperforms B&H for all other securities. As observed earlier, the trading agent mostly held long position for Sampo and OMX Helsinki PI, sometimes switching to short position and quickly back to long position. This explains why the cumulative return is close to B&H, although being slightly worse for both securities. The performance of TDQN is in line with modified DQN for Sampo, but performing slightly better on OMX Helsinki PI where it also beats B&H strategy. Finally, the modified DQN shows weak performance for Bittium stock, producing negative cumulative return of −30.44%. This is over 20 percentages worse than B&H. The TDQN results in cumulative return of −6.24%, surpassing B&H strategy slightly.

An interesting remark can be made on Random strategy. Although showing good performance in the environment without transaction costs, and actually outperforming B&H for all securities, the performance within the environment with transaction costs is very poor (−28.93% for Nordea, −26.78% for Sampo, −56.11% for Bittium, and −55.84% for OMX Helsinki PI). This extreme change in performance could be explained by the fact that the trading activity for Random strategy is very frequent (changing position very often) which decays the portfolio capital rapidly, as 0.2% is deducted from the capital for each trade. The profits from trading activity are inadequate to cover the rapid decaying due to transaction costs, which results in weak performance.

In conclusion, the proposed approach indicates strong performance compared to benchmark strategies in an environment without transaction costs. The risk-adjusted returns were the highest for three stock out of four. Although showing good performance without transaction costs, opposing results are remarked for an environment with transaction costs. The modified DQN surpass B&H on only one of the securities, and underperforms on the other three. For Sampo and OMX Helsinki PI, the results are close to B&H strategy due to the agent holding long position most of the trading period. For Bittium, large negative returns is recorded. The overall results do not indicate consistent performance that would be able to generate excess returns. Because of results from previous section (Section 4.1) in which the outcomes suggested that the model is not very stable and struggles to exploit any meaningful information within the environment, the descriptive statistics within this section should be approached critically. The instability in DQN will certainly introduce randomness to performance metrics such as cumulative return, maximum drawdown, and profitability. Therefore, reproducibility of these results are not guaranteed.

At last, the development of each strategy for trading period is visualized in Figures 18-21 with and without transaction costs. As each strategy differs from other strategies, the development of the portfolio capitals diverge considerably, especially in the environment without transaction costs. The divergence is smaller within the environment with transaction costs. In some cases such as in Figures 19b, 20b, and 21b, TDQN or modified DQN hold long position, thus the capital development follows closely B&H strategy. As a conclusion, no strategy seem to outperform other strategies consistently, although modified DQN does finish the highest in multiple simulations.

Table 7: Performance metrics for trading strategies without transaction costs

This table describes the summary statistics for each security during the testing period from 1st of January 2022 to 31st of December 2023. The transaction costs are set to 0% for this evaluation. There are four panels, Panel A describes the statistics for Nordea stock, Panel B describes the statistics for Sampo stock, Panel C describes the statistics for Bittium stock, and Panel D describes the statistics for OMX Helsinki PI index. The metrics are defined as follows: *Cumulative return* is the gains or losses relative to original portfolio value; *Stdev* is the annualized standard deviation of the portfolio; *Sharpe Ratio* is the risk-adjusted return; *Maximum drawdown* is the largest peak-to-trough decline in the portfolio value in percentages; *Profitability* indicates the percentage of profitable trades; and *Skewness* is the distribution of trading yields.

Panel A: Descriptive Statistics for Nordea stock

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	3.05%	10.23%	−14.40%	1.85%	−6.83%	−0.33%
Stdev	26.80%	23.79%	25.44%	26.24%	25.00%	25.47%
Sharpe Ratio	0.191	0.323	−0.178	0.167	−0.016	0.121
Maximum drawdown	28.48%	18.74%	33.79%	30.10%	33.22%	24.59%
Profitability	100%	47.73%	43.89%	67.50%	44.44%	44.66%
Skewness	−0.825	0.676	−0.029	−0.532	−0.119	−0.411

Panel B: Descriptive Statistics for Sampo stock

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	2.31%	9.30%	0.89%	16.94%	−22.58%	12.17%
Stdev	21.14%	20.15%	19.99%	20.59%	19.82%	20.19%
Sharpe Ratio	0.160	0.322	0.122	0.482	−0.547	0.634
Maximum drawdown	23.48%	26.28%	32.36%	17.23%	39.01%	18.64%
Profitability	100%	53.04%	62.50%	64.29%	39.81%	45.31%
Skewness	−0.368	−0.379	−0.282	0.522	0.194	0.089

Panel C: Descriptive Statistics for Bittium

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	−9.59%	4.89%	−56.95%	−13.02%	15.40%	75.72%
Stdev	36.85%	36.23%	34.34%	35.41%	34.35%	34.57%
Sharpe Ratio	0.041	0.249	−1.049	−0.021	0.375	0.983
Maximum drawdown	44.13%	52.99%	59.47%	44.98%	36.81%	14.63%
Profitability	0%	57.84%	33.33%	53.33%	52.53%	53.19%
Skewness	2.327	−0.340	−2.001	0.643	2.275	−2.287

Panel D: Descriptive Statistics for OMX Helsinki PI

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	−20.77%	−18.89%	5.04%	−1.40%	−13.82%	21.51%
Stdev	16.52%	16.45%	16.14%	16.59%	16.17%	15.84%
Sharpe Ratio	−0.624	−0.555	0.233	0.040	−0.380	0.695
Maximum drawdown	28.94%	28.87%	17.84%	19.51%	25.51%	15.84%
Profitability	0%	50.57%	53.33%	54.55%	42.50%	47.96%
Skewness	−0.269	−0.205	0.223	−0.175	0.247	0.155

Table 8: Performance metrics for trading strategies with transaction cost of 0.2%

This table describes the summary statistics for each security during the testing period from 1st of January 2022 to 31st of December 2023. The transaction costs are set to 0.2% for this evaluation. There are four panels, Panel A describes the statistics for Nordea stock, Panel B describes the statistics for Sampo stock, Panel C describes the statistics for Bittium stock, and Panel D describes the statistics for OMX Helsinki PI index. The metrics are defined as follows: *Cumulative return* is the gains or losses relative to original portfolio value; *Stdev* is the annualized standard deviation of the portfolio; *Sharpe Ratio* is the risk-adjusted return; *Maximum drawdown* is the largest peak-to-trough decline in the portfolio value in percentages; *Profitability* indicates the percentage of profitable trades; and *Skewness* is the distribution of trading yields.

Panel A: Descriptive Statistics for Nordea stock

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	3.05%	−28.93%	−10.33%	−13.04%	−55.78%	43.24%
Stdev	26.80%	24.26%	24.74%	26.17%	25.66%	25.49%
Sharpe Ratio	0.187	−0.583	−0.097	−0.135	−1.460	0.833
Maximum drawdown	28.48%	52.12%	32.88%	34.21%	57.51%	18.39%
Profitability	100%	40.14%	40.91%	57.50%	32.50%	60.00%
Skewness	−0.824	0.734	0.284	−0.522	−0.266	−0.529

Panel B: Descriptive Statistics for Sampo stock

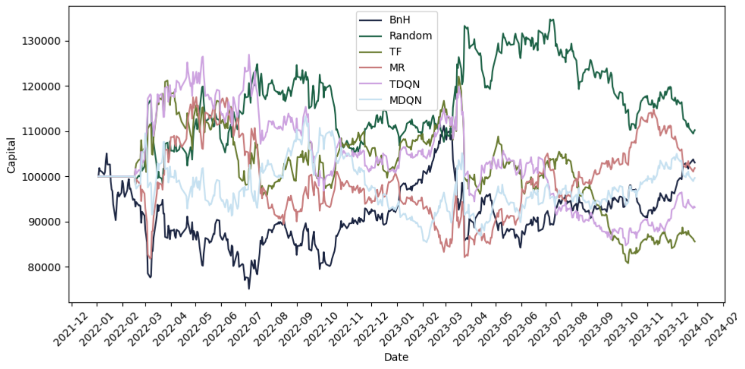
Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	2.31%	−26.78%	−16.45%	−10.43%	−2.20%	−2.77%
Stdev	21.14%	20.27%	19.99%	20.81%	19.89%	20.28%
Sharpe Ratio	0.155	−0.667	−0.349	−0.161	0.043	0.032
Maximum drawdown	23.48%	38.56%	36.96%	24.49%	19.98%	23.37%
Profitability	100%	47.10%	55.00%	55.00%	50.00%	53.85%
Skewness	−0.367	−0.568	−0.584	0.329	−0.127	−0.381

Panel C: Descriptive Statistics for Bittium

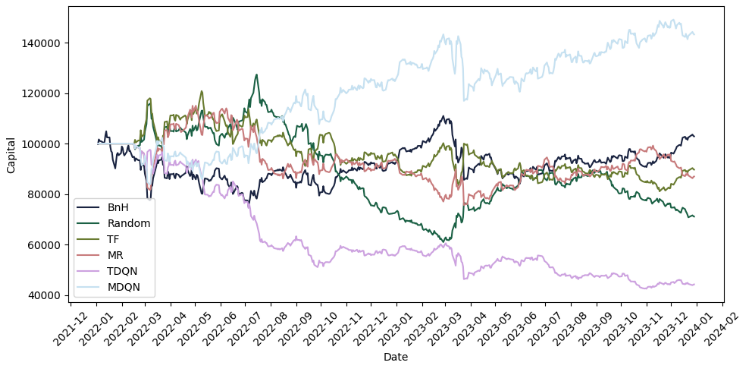
Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	−9.59%	−56.11%	−62.27%	−53.18%	−6.24%	−30.44%
Stdev	36.85%	35.25%	34.22%	35.94%	34.83%	34.77%
Sharpe Ratio	0.039	−0.989	−1.246	−0.876	0.076	−0.353
Maximum drawdown	44.13%	59.03%	64.40%	62.33%	46.42%	57.19%
Profitability	0%	37.80%	33.33%	43.14%	63.64%	38.00%
Skewness	2.328	−0.326	−2.018	0.653	2.476	2.122

Panel D: Descriptive Statistics for OMX Helsinki PI

Metric	B&H	Random	TF	MR	TDQN	DQN with limits
Cumulative return	−20.81%	−55.84%	−6.22%	−29.98%	−17.64%	−22.55%
Stdev	16.55%	16.05%	15.73%	16.88%	16.77%	16.23%
Sharpe Ratio	−0.629	−2.470	−0.126	−0.973	−0.496	−0.708
Maximum drawdown	29.00%	62.36%	23.28%	30.47%	24.48%	29.48%
Profitability	0%	36.60%	37.50%	36.00%	0.00%	36.00%
Skewness	−0.268	0.266	0.053	−0.084	−0.139	−0.114

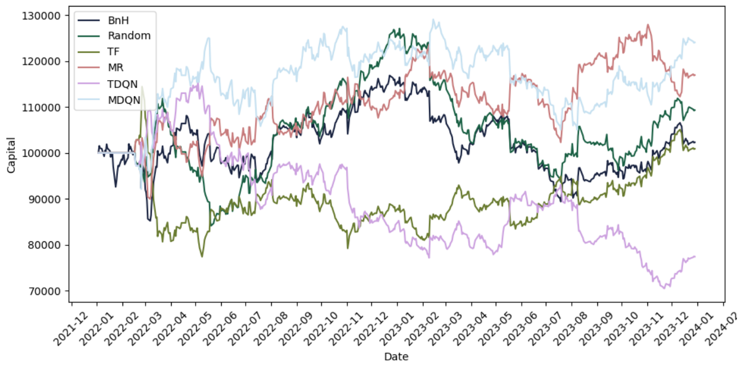


(a) Transaction costs = 0%

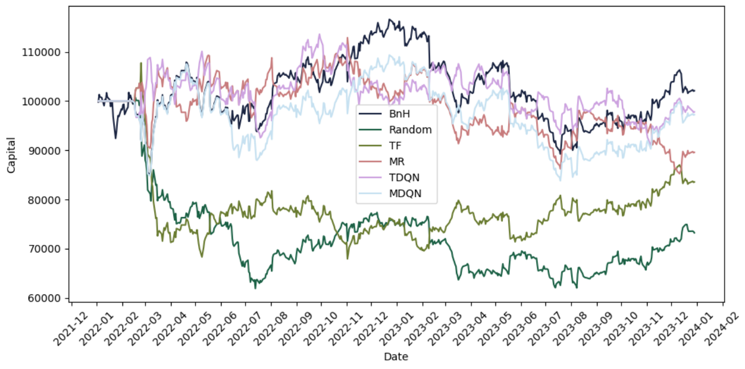


(b) Transaction costs = 0.2%

Figure 18: Portfolio development for Nordea

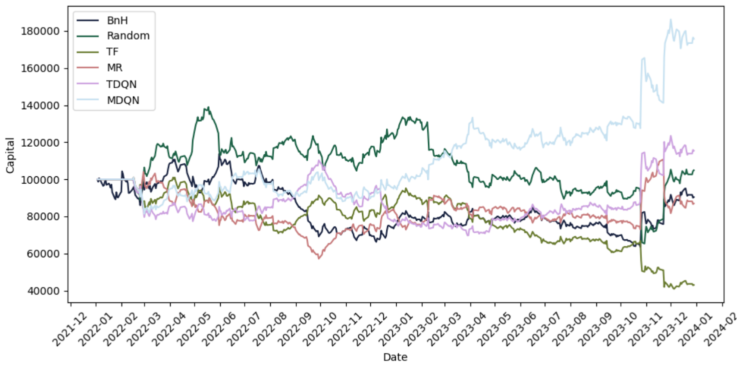


(a) Transaction costs = 0%

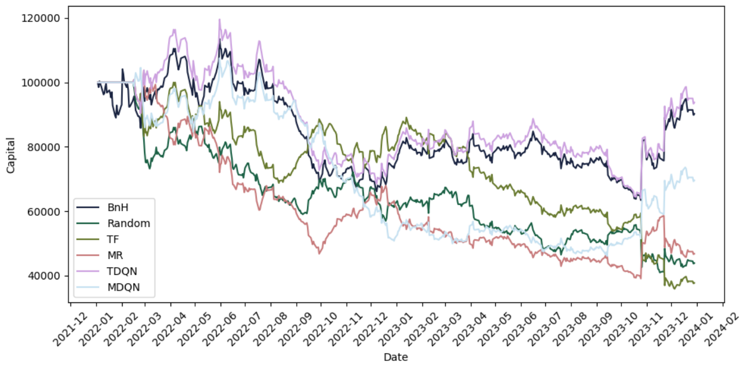


(b) Transaction costs = 0.2%

Figure 19: Portfolio development for Sampo

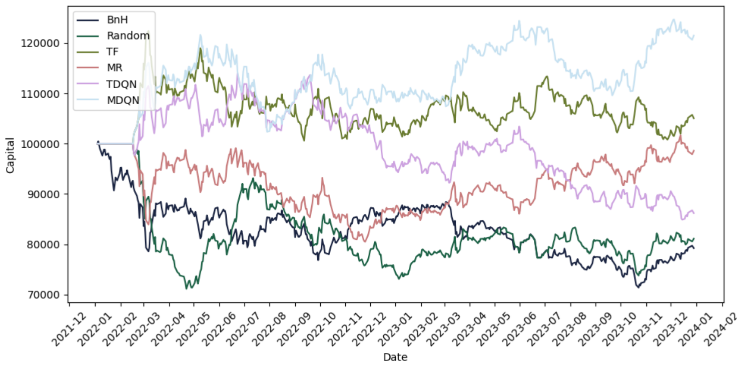


(a) Transaction costs = 0%

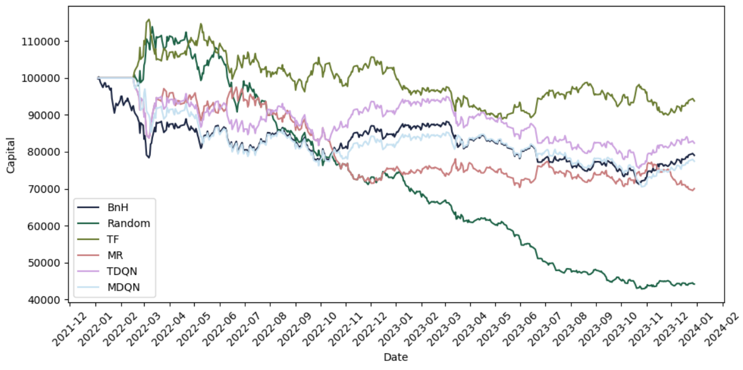


(b) Transaction costs = 0.2%

Figure 20: Portfolio development for Bittium



(a) Transaction costs = 0%



(b) Transaction costs = 0.2%

Figure 21: Portfolio development for OMX Helsinki PI

5 CONCLUSIONS

The first objective of this study was to examine whether novel DQN algorithm with trading limits can be utilized for effective AT strategy that can surpass benchmark strategies on Finnish stock market. The analysis was conducted on three individual stocks (Nordea, Sampo, and Bittium) and broader Finnish stock market index (OMX Helsinki PI) between the beginning of 2022 and the end of 2023. The algorithm introduced by Théate and Ernst (2021) was developed further by incorporating trading boundaries for customized reinforcement learning feedback mechanism. The additional stop loss and take profit limits were designed for further guidance of the trading agent, giving extra penalty for trades that crossed stop loss level, and extra reward when reaching take profit goal. The second objective was to examine how the introduction of these trading limits affected the behavior of trading agent, comparing the results to initial TDQN created by Théate and Ernst (2021).

To address the first research question whether trading algorithm is able to consistently generate excess returns on Finnish stock market, the model performance was first analyzed. The results indicated that the model was quite unstable, having large fluctuations in Q-values and average rewards. Although Sharpe ratios showed promising development for training period within environment without transaction costs, in other cases the ratios seemed to stagnate and vary significantly. This behavior does not indicate that the agent would be efficiently learning to increase Sharpe over time. Overall, these results tell that the model is relatively unstable, and thus consistent excess returns are not possible.

The proposed approach was benchmarked against various trading strategies. The randomness of the model reflected on benchmarking evaluation. Although the modified DQN outperformed benchmark strategies in multiple scenarios, the cumulative returns suggest relatively high variation as in some simulations the strategy yields returns much higher than other strategies, but in some simulations underperforms others significantly. These outcomes show that the proposed DQN did perform well under these circumstances, but the reproducibility and consistency are deficient for the strategy.

To assess the second research question, descriptive statistics were calculated for trading limits and model statistics were plotted to compare modified DQN to the original TDQN. In the descriptive statistics tables (Table 5 and 6), limit trigger counts and rewards acquired from triggers were presented. It was observed that overall, the trading limits provided a little bit of additional positive net reward for all simulations. In addition, the Q-values and average rewards over training period were set against the results from the original TDQN. Instead of guiding the agent towards finding an optimal policy, the trading boundaries seemed to introduce more fluctuation and randomness to the model. As the trading limits are usually triggered in volatile price changes in security price, which in itself create variation in rewards, providing additional one-off rewards at these times magnifies the phenomenon. It was also pointed out that as the trading limits are set dynamically using ATR, the changing limit ranges might add to the complexity of the model. Overall, using the trading limits supported the agent to acquire higher rewards, but at the cost of model stability.

By analyzing model performance, it was noted that the performance of model does not converge to optimum, but has relatively much fluctuation. A possible reason could be that the stock market is just too complex for the agent to learn, and includes a lot of noise in the data. Therefore, the agent might not be able to master the complexity of stock market. Additionally, it is not evident that historical price movements can be exploited to predict the subsequent prices, meaning that although the agent would be able to master the trading environment, it would be futile as this information would bring no advantages. This ideology supports Random Walk Theory which was explained in Section 2.1.2. However, some signs of learning were recorded, as Sharpe ratio during training period in the environment without transaction did increase gradually.

The instability of the proposed model does bring up the discussion whether DQN is suitable for stock market environment. While it is a fact that DQN has demonstrated good performance, for example in video games, and rewarding the agent based on portfolio gains is intuitive method to guide the agent towards ever increasing profits, the reality remains harsh in terms of stock trading. The complexity and noisiness of the environment makes the DQN environment unique from many other domains of application. The stationary environment introduces stationary to rewarding also, which hinders the consistency of reward signal the agent is receiving. Further research could be conducted on stabilizing the model performance, if possible.

For model architecture, whereas the newly introduced trading boundaries do provide additional reward for the DQN agent, a more restricting framework could be established. Instead of giving additional reward, triggering trading boundaries could force the agent to give up its current trading position (long or short) and return to neutral position, similarly to the usual real-world approach. This was not implemented in this study, as the action space was mapped to two possible options, going long or short with all resources, and it was not desired to force the agent to take opposite trading position. Using trading framework like this could enhance the risk management of DQN trading strategy, and resulting in different outcomes.

In conclusion, DQN is a novel algorithm that has seen wide adoption among finance literature recently. Many practitioners have indicated positive results, but the dynamic and complex nature of financial markets hinders the application of DQN just like any other prediction model. The results of this study indicate that while some improvement on agent's behavior was discovered, the financial markets exhibits an extremely difficult environment for DQN where consistent learning is vital.

REFERENCES

Ang, A. – Bekaert, G. (2007) Stock return predictability: Is it there? *The Review of Financial Studies*, vol. 20 (3), 651–707.

Ansari, Y., Yasmin, S., Naz, S., Zaffar, H., Ali, Z., Moon, J. – Rho, S. (2022) A deep reinforcement learning-based decision support system for automated stock market trading. *IEEE Access*, vol. 10, 127469–127501.

Awad, A. L., Elkaffas, S. M. – Fakhr, M. W. (2023) Stock market prediction using deep reinforcement learning. *Applied System Innovation*, vol. 6 (6), 106.

Bachelier, L. (1900) Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, vol. 17, 21–86.

Bellman, R. (1957) A markovian decision process. *Journal of Mathematics and Mechanics*, vol. 6 (5), 679–684, URL: `http://www.jstor.org/stable/24900506`.

Bertoluzzo, F. – Corazza, M. (2012) Testing different reinforcement learning configurations for financial trading: Introduction and applications. *Procedia Economics and Finance*, vol. 3, 68–77.

Biais, B., Foucault, T. – Moinas, S. (2011) Equilibrium high frequency trading. In *Proceedings from the fifth annual Paul Woolley Centre conference, London School of Economics.*

Black, F. (1986) Noise. *The journal of finance*, vol. 41 (3), 528–543.

Bodie, Z. (1976) Common stocks as a hedge against inflation. *The Journal of Finance*, vol. 31 (2), 459–470, URL: `http://www.jstor.org/stable/2326617`.

Brim, A. – Flann, N. S. (2022) Deep reinforcement learning stock market trading, utilizing a cnn with candlestick images. *Plos one*, vol. 17 (2), e0263181.

Brogaard, J., Hendershott, T. – Riordan, R. (2014) High-frequency trading and price discovery. *The Review of Financial Studies*, vol. 27 (8), 2267–2306.

Campbell, J. Y. – Shiller, R. J. (1998) Valuation ratios and the long-run stock market outlook.

Carapuço, J., Neves, R. – Horta, N. (2018) Reinforcement learning applied to forex trading. *Applied Soft Computing*, vol. 73, 783–794, URL: `https://www.sciencedirect.com/science/article/pii/S1568494618305349`.

Carta, S., Ferreira, A., Podda, A. S., Reforgiato Recupero, D. – Sanna, A. (2021) Multi-dqn: An ensemble of deep q-learning agents for stock market forecasting. *Expert Systems with Applications*, vol. 164, 113820, URL: `https://www.sciencedirect.com/science/article/pii/S0957417420306321`.

Cartea, Á., Jaimungal, S. – Sánchez-Betancourt, L. (2021) Deep reinforcement learning for algorithmic trading. *Available at SSRN 3812473*.

Casqueiro, P. X. – Rodrigues, A. J. (2006) Neuro-dynamic trading methods. *European journal of operational research*, vol. 175 (3), 1400–1412.

Chaboud, A. P., Chiquoine, B., Hjalmarsson, E. – Vega, C. (2014) Rise of the machines: Algorithmic trading in the foreign exchange market. *The Journal of Finance*, vol. 69 (5), 2045–2084, URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.12186`.

Chakole, J. – Kurhekar, M. (2020) Trend following deep q-learning strategy for stock trading. *Expert Systems*, vol. 37 (4), e12514, URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.12514`.

Chen, L. – Gao, Q. (2019) Application of deep reinforcement learning on automated stock trading. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 29–33.

Chen, X., Wang, Q., Yuxin, L., Hu, C., Wang, C. – Yan, Q. (2023) Stock price forecast based on dueling deep recurrent q-network. In *2023 IEEE 6th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, 1091–1096.

Clements, W. R., Delft, B. V., Robaglia, B.-M., Slaoui, R. B. – Toth, S. (2020) Estimating risk and uncertainty in deep reinforcement learning.

Coggan, M. (2004) Exploration and exploitation in reinforcement learning. *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*.

Cootner, P. H. (1967) *The random character of stock market prices*. The MIT Press.

Cui, K., Hao, R., Huang, Y., Li, J. – Song, Y. (2023) A novel convolutional neural networks for stock trading based on ddqn algorithm. *IEEE Access*, vol. 11, 32308–32318.

Dang, Q.-V. (2019) Reinforcement learning in stock trading. In *International conference on computer science, applied mathematics and applications*, 311–322, Springer.

De Bondt, W. F. – Thaler, R. (1985) Does the stock market overreact? *The Journal of finance*, vol. 40 (3), 793–805.

De Long, J. B., Shleifer, A., Summers, L. H. – Waldmann, R. J. (1990) Noise trader risk in financial markets. *Journal of political Economy*, vol. 98 (4), 703–738.

Fama, E. F. (1965a) The behavior of stock-market prices. *The Journal of Business*, vol. 38 (1), 34–105, URL: `http://www.jstor.org/stable/2350752`.

Fama, E. F. (1965b) Random walks in stock market prices. *Financial Analysts Journal*, vol. 21 (5), 55–59, URL: `http://www.jstor.org/stable/4469865`.

Fama, E. F. (1970) Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, vol. 25 (2), 383–417, URL: `http://www.jstor.org/stable/2325486`.

Fama, E. F. (1981) Stock returns, real activity, inflation, and money. *The American Economic Review*, vol. 71 (4), 545–565, URL: `http://www.jstor.org/stable/1806180`.

Fama, E. F. (1991) Efficient capital markets: Ii. *The journal of finance*, vol. 46 (5), 1575–1617.

Fama, E. F. – French, K. R. (1988a) Dividend yields and expected stock returns. *Journal of financial economics*, vol. 22 (1), 3–25.

Fama, E. F. – French, K. R. (1988b) Permanent and temporary components of stock prices. *Journal of political Economy*, vol. 96 (2), 246–273.

Fama, E. F. – French, K. R. (1993) Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, vol. 33 (1), 3–56, URL: `https://www.sciencedirect.com/science/article/pii/0304405X93900235`.

French, K. R. (1980) Stock returns and the weekend effect. *Journal of financial economics*, vol. 8 (1), 55–69.

French, K. R., Schwert, G. – Stambaugh, R. F. (1987) Expected stock returns and volatility. *Journal of Financial Economics*, vol. 19 (1), 3–29, URL: `https://www.sciencedirect.com/science/article/pii/0304405X87900262`.

Gao, X. – Chan, L. (2000) An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *Proceedings of the international conference on neural information processing*, 832–837.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z. – Su, J. (2020) Application of deep q-network in portfolio management. In *2020 5th IEEE International Conference on Big Data Analytics (ICBDA)*, 268–275, IEEE.

Gibbs, S. (2014) Google buys uk artificial intelligence startup deepmind for £400m. URL: `https://www.theguardian.com/technology/2014/jan/27/google-acquires-uk-artificial-intelligence-startup-deepmind`.

Goldstein, M. A., Kumar, P. – Graves, F. C. (2014) Computerized and high-frequency trading. *Financial Review*, vol. 49 (2), 177–202, URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/fire.12031`.

Hao, L., Wang, B., Lu, Z. – Hu, K. (2022) Application of deep reinforcement learning in financial quantitative trading. In *2022 4th International Conference on Communications, Information System and Computer Engineering (CISCE)*, 466–471.

Hasselt, H. (2010) Double q-learning. *Advances in neural information processing systems*, vol. 23.

He, Y., Yang, Y., Li, Y. – Sun, P. (2022) A novel deep reinforcement learning-based automatic stock trading method and a case study. In *2022 IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain Beyond (iGETblockchain)*, 1–6.

Hendershott, T., Jones, C. M. – Menkveld, A. J. (2011) Does algorithmic trading improve liquidity? *The Journal of Finance*, vol. 66 (1), 1–33, URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.2010.01624.x`.

Hirsa, A., Osterrieder, J., Hadji-Misheva, B. – Posth, J.-A. (2021) Deep reinforcement learning on a multi-asset environment for trading.

Hu, G. (2023) Advancing algorithmic trading: A multi-technique enhancement of deep q-network models.

Hu, J. (2022) Intelligent decisionmaking system through lstm prediction model and dqn algorithm. In *2022 IEEE 2nd International Conference on Data Science and Computer Application (ICDSCA)*, 958–965.

Huang, C. Y. (2018) Financial trading as a game: A deep reinforcement learning approach.

Huang, Y., Lu, X., Zhou, C. – Song, Y. (2023) Dade-dqn: Dual action and dual environment deep q-network for enhancing stock trading strategy. *Mathematics*, vol. 11 (17), URL: `https://www.mdpi.com/2227-7390/11/17/3626`.

Ioannidis, J. P. (2005) Why most published research findings are false. *PLoS medicine*, vol. 2 (8), e124.

Ioffe, S. – Szegedy, C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456, pmlr.

Jaffe, J. F. – Mandelker, G. (1976) The "fisher effect" for risky assets: An empirical investigation. *The Journal of Finance*, vol. 31 (2), 447–458, URL: `http://www.jstor.org/stable/2326616`.

Jegadeesh, N. – Titman, S. (1993) Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, vol. 48 (1), 65–91, URL: `http://www.jstor.org/stable/2328882`.

Jiang, L., Huang, H. – Ding, Z. (2019) Path planning for intelligent robots based on deep q-learning with experience replay and heuristic knowledge. *IEEE/CAA Journal of Automatica Sinica*, vol. 7 (4), 1179–1189.

Jin, O. – El-Saawy, H. (2016) Portfolio management using reinforcement learning. *Stanford University.*

Jovanovic, F. – Le Gall, P. M. (2001) Does god practice a random walk? the financial physics of a nineteenth-century forerunner, jules regnault. *European journal of the history of economic thought*, vol. 8 (3), 332–362.

Kahneman, D. – Tversky, A. (1979) Prospect theory: An analysis of decision under risk. *Econometrica*, vol. 47 (2), 263–291, URL: `http://www.jstor.org/stable/1914185`.

Keim, D. B. (1983) Size-related anomalies and stock return seasonality: Further empirical evidence. *Journal of financial economics*, vol. 12 (1), 13–32.

Khan, S. (n.d.) Chain rule. URL: `https://www.khanacademy.org/math/ap-calculus-ab/ab-differentiation-2-new/ab-3-1a/v/chain-rule-introduction`.

Khemlichi, F., Chougrad, H., Khamlichi, Y. I., El Boushaki, A. – Ali, S. E. B. (2021) Deep deterministic policy gradient for portfolio management. In *2020 6th IEEE Congress on Information Science and Technology (CiSt)*, 424–429, IEEE.

Khowaja, S. A. – Khuwaja, P. (2021) Q-learning and lstm based deep active learning strategy for malware defense in industrial iot applications. *Multimedia Tools and Applications*, vol. 80 (10), 14637–14663.

Kim, S.-H., Park, D.-Y. – Lee, K.-H. (2022) Hybrid deep reinforcement learning for pairs trading. *Applied Sciences*, vol. 12 (3), URL: https://www.mdpi.com/2076-3417/12/3/944.

Kim, T. – Kim, H. Y. (2019) Optimizing the pairs-trading strategy using deep reinforcement learning with trading and stop-loss boundaries. *Complexity*, vol. 2019, 1–20.

Kingma, D. P. – Ba, J. (2014) Adam: A method for stochastic optimization.

Kirilenko, A., Kyle, A. S., Samadi, M. – Tuzun, T. (2017) The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, vol. 72 (3), 967–998, URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.12498.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A. et al. (2017) Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, vol. 114 (13), 3521–3526.

Kumar, S., Vishal, M. – Ravi, V. (2022) Explainable reinforcement learning on financial stock trading using shap.

Lazov, B. (2023) A deep reinforcement learning trader without offline training.

LeCun, Y., Bengio, Y. – Hinton, G. (2015) Deep learning. *Nature*, vol. 521 (7553), 436–444, URL: https://doi.org/10.1038/nature14539.

Lee, J. W., Park, J., O, J., Lee, J. – Hong, E. (2007) A multiagent approach to *q*-learning for daily stock trading. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37 (6), 864–877.

Li, Y., Zheng, W. – Zheng, Z. (2019) Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access*, vol. 7, 108014–108022.

Li, Y., Zhou, P., Li, F. – Yang, X. (2021) An improved reinforcement learning model based on sentiment analysis.

Liang, Z., Chen, H., Zhu, J., Jiang, K. – Li, Y. (2018) Adversarial deep reinforcement learning in portfolio management.

Lintner, J. (1975) Inflation and security returns. *The journal of finance*, vol. 30 (2), 259–280.

Liu, R. – Zou, J. (2018) The effects of memory replay in reinforcement learning. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 478–485.

Liu, Y., Liu, Q., Zhao, H., Pan, Z. – Liu, C. (2020) Adaptive quantitative trading: An imitative deep reinforcement learning approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34 (02), 2128–2135, URL: `https://ojs.aaai.org/index.php/AAAI/article/view/5587`.

Lo, A. W. – MacKinlay, A. C. (1988) Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test. *The Review of Financial Studies*, vol. 1 (1), 41–66, URL: `https://doi.org/10.1093/rfs/1.1.41`.

Lo, A. W. – MacKinlay, A. C. (1999) *A non-random walk down wall street.* Princeton University Press.

Lo, A. W., Mamaysky, H. – Wang, J. (2000) Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The journal of finance*, vol. 55 (4), 1705–1765.

Lucarelli, G. – Borrotti, M. (2020) A deep q-learning portfolio management framework for the cryptocurrency market. *Neural Computing and Applications*, vol. 32, 17229–17244.

Luo, Y. – Duan, Z. (2023) Agent performing autonomous stock trading under good and bad situations.

Malkiel, B. G. (1973) *A random walk down wall street.* Norton.

Malkiel, B. G. (2003) The efficient market hypothesis and its critics. *Journal of economic perspectives*, vol. 17 (1), 59–82.

Mandelbrot, B. (1966) Forecasts of future prices, unbiased markets, and" martingale" models. *The Journal of Business*, vol. 39 (1), 242–255.

Massahi, M. – Mahootchi, M. (2024) A deep q-learning based algorithmic trading system for commodity futures markets. *Expert Systems with Applications*, vol. 237, 121711, URL: `https://www.sciencedirect.com/science/article/pii/S0957417423022133`.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015) Human-level control through deep reinforcement learning. *nature*, vol. 518 (7540), 529–533.

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

Moody, J., Wu, L., Liao, Y. – Saffell, M. (1998) Performance functions and reinforcement learning for trading systems and portfolios. *Journal of forecasting*, vol. 17 (5-6), 441–470.

Nagy, P., Calliess, J.-P. – Zohren, S. (2023) Asynchronous deep double dueling q-learning for trading-signal execution in limit order book markets. *Frontiers in Artificial Intelligence*, vol. 6, URL: `https://www.frontiersin.org/articles/10.3389/frai.2023.1151003`.

Nasdaq (n.d.) Overview for omxhpi. URL: `https://indexes.nasdaqomx.com/Index/Overview/OMXHPI`.

Nelson, C. R. (1976) Inflation and rates of return on common stocks. *The Journal of Finance*, vol. 31 (2), 471–483, URL: `http://www.jstor.org/stable/2326618`.

Neuneier, R. (1995) Optimal asset allocation using adaptive dynamic programming. *Advances in neural information processing systems*, vol. 8.

Neuneier, R. (1997) Enhancing q-learning for optimal asset allocation. *Advances in neural information processing systems*, vol. 10.

Ng, A. (n.d.) URL: `https://cs230.stanford.edu/section/4/`.

Nicholson, S. F. (1960) Price-earnings ratios. *Financial Analysts Journal*, 43–45.

Ning, B., Lin, F. H. T. – Jaimungal, S. (2021) Double deep q-learning for optimal execution. *Applied Mathematical Finance*, vol. 28 (4), 361–380.

Niu, H., Li, S. – Li, J. (2022) Metatrader: An reinforcement learning approach integrating diverse policies for portfolio optimization. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, 1573–1583, Association for Computing Machinery, New York, NY, USA, URL: `https://doi.org/10.1145/3511808.3557363`.

O, J., Lee, J., Lee, J. W. – Zhang, B.-T. (2006) Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences*, vol. 176 (15), 2121–2147, URL: `https://www.sciencedirect.com/science/article/pii/S0020025505003166`.

OpenAI (n.d.) URL: `https://spinningup.openai.com/en/latest/algorithms/ddpg.html`.

Otabek, S. – Choi, J. (2024) Multi-level deep q-networks for bitcoin trading strategies. *Scientific Reports*, vol. 14 (1), 771.

Park, H., Sim, M. K. – Choi, D. G. (2020) An intelligent financial portfolio trading strategy using deep q-learning. *Expert Systems with Applications*, vol. 158, 113573, URL: `https://www.sciencedirect.com/science/article/pii/S0957417420303973`.

Parkavi, A., J., S. B. – Sini Anna, A. (2022) Automated stock trading using deep reinforcement learning and portfolio optimization. *Webology*, vol. 19 (2).

Pesaran, M. H. – Timmermann, A. (1995) Predictability of stock returns: Robustness and economic significance. *The Journal of Finance*, vol. 50 (4), 1201–1228.

Pichai, S. (2023) Google deepmind: Bringing together two world-class ai teams. URL: `https://blog.google/technology/ai/april-ai-update/`.

Pigorsch, U. – Schäfer, S. (2022) High-dimensional stock portfolio trading with deep reinforcement learning. In *2022 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, 1–8.

Poterba, J. M. – Summers, L. H. (1988) Mean reversion in stock prices: Evidence and implications. *Journal of financial economics*, vol. 22 (1), 27–59.

Qiu, Y., Du, C., Zhang, Y., Qiu, Z. – Yang, Y. (2022) Trading strategy based on dqn and dynamic programming. In *2022 3rd International Conference on Computer*

Vision, Image and Deep Learning International Conference on Computer Engineering and Applications (CVIDL ICCEA), 334–338.

Ramaswamy, A. – Hüllermeier, E. (2022) Deep q-learning: Theoretical insights from an asymptotic analysis. *IEEE Transactions on Artificial Intelligence*, vol. 3 (2), 139–151.

Samuelson, P. A. (1965) Proof that properly anticipated prices fluctuate randomly. *IMR-INDUSTRIAL MANAGEMENT REVIEW*, vol. 6 (2), 41–49.

Sarkar, S. (2023) Quantitative trading using deep q learning. *International Journal for Research in Applied Science and Engineering Technology*, vol. 11 (4), 731â€“738, URL: `http://dx.doi.org/10.22214/ijraset.2023.50170`.

Sarker, I. H. (2021) Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, vol. 2 (6), 420, URL: `https://doi.org/10.1007/s42979-021-00815-1`.

Schaul, T., Quan, J., Antonoglou, I. – Silver, D. (2016) Prioritized experience replay.

Shah, R., Tambe, A., Bhatt, T. – Rote, U. (2020) Real-time stock market forecasting using ensemble deep learning and rainbow dqn. In *Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST)*.

Shiller, R. J. (2000) *Irrational exuberance.* Princeton University Press.

Shin, H.-G., Ra, I. – Choi, Y.-H. (2019) A deep multimodal reinforcement learning system combined with cnn and lstm for stock trading. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 7–11.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. – Riedmiller, M. (2014) Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, eds. E. P. Xing – T. Jebara, vol. 32 of *Proceedings of Machine Learning Research*, 387–395, PMLR, Bejing, China, URL: `https://proceedings.mlr.press/v32/silver14.html`.

Suliman, U., van Zyl, T. L. – Paskaramoorthy, A. (2022) Cryptocurrency trading agent using deep reinforcement learning. In *2022 9th International Conference on Soft Computing Machine Intelligence (ISCMI)*, 6–10.

Sutton, R. S., Barto, A. G. et al. (1998) *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge.

Thakkar, A. – Chaudhari, K. (2021) A comprehensive survey on deep neural networks for stock market: The need, challenges, and future directions. *Expert Systems with Applications*, vol. 177, 114800, URL: `https://www.sciencedirect.com/science/article/pii/S0957417421002414`.

Théate, T. – Ernst, D. (2021) An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, vol. 173, 114632, URL: `https://www.sciencedirect.com/science/article/pii/S0957417421000737`.

Tran, M., Pham-Hi, D. – Bui, M. (2023) Optimizing automated trading systems with deep reinforcement learning. *Algorithms*, vol. 16 (1), 23.

Wang, J., Jing, F. – He, M. (2023) Stock trading strategy of reinforcement learning driven by turning point classification. *Neural Processing Letters*, vol. 55 (3), 3489–3508.

Wang, R., Wei, H., An, B., Feng, Z. – Yao, J. (2021) Deep stock trading: A hierarchical reinforcement learning framework for portfolio optimization and order execution.

Wang, Y. – Yan, G. (2021) Survey on the application of deep learning in algorithmic trading. *Data Science in Finance and Economics*, vol. 1 (4), 345–361.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. – Freitas, N. (2016) Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, 1995–2003, PMLR.

Watkins, C. (1989) Learning from delayed rewards.

Watkins, C. – Dayan, P. (1992) Q-learning. *Machine learning*, vol. 8, 279–292.

Welles, W. j. J. (1978) *New Concepts in Technical Trading Systems.* Trend Research.

Wen, W., Yuan, Y. – Yang, J. (2021) Reinforcement learning for options trading. *Applied Sciences*, vol. 11 (23), URL: `https://www.mdpi.com/2076-3417/11/23/11208`.

Wood, T. (2020) Sigmoid function. URL: `https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function`.

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V. – Fujita, H. (2020) Adaptive stock trading strategies with deep reinforcement learning methods. *Information*

Sciences, vol. 538, 142–158, URL: `https://www.sciencedirect.com/science/article/pii/S0020025520304692`.

Ye, J., Li, X. – Wang, Y. (2022) A multi-agent deep reinforcement learning framework for vwap strategy optimization. In *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Zahavy, T., Ben-Zrihem, N. – Mannor, S. (2016) Graying the black box: Understanding dqns. In *Proceedings of The 33rd International Conference on Machine Learning*, eds. M. F. Balcan – K. Q. Weinberger, vol. 48 of *Proceedings of Machine Learning Research*, 1899–1908, PMLR, New York, New York, USA, URL: `https://proceedings.mlr.press/v48/zahavy16.html`.

Zhang, H., Jiang, Z. – Su, J. (2021) A deep deterministic policy gradient-based strategy for stocks portfolio management. In *2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*, 230–238.

Zhang, J., Lei, Y. et al. (2022) Deep reinforcement learning for stock prediction. *scientific programming*, vol. 2022.

Zhang, W., Yang, G., Lin, Y., Ji, C. – Gupta, M. M. (2018a) On definition of deep learning. In *2018 World Automation Congress (WAC)*, 1–5.

Zhang, Y., Sun, P., Yin, Y., Lin, L. – Wang, X. (2018b) Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning. In *2018 IEEE intelligent vehicles symposium (IV)*, 1251–1256, IEEE.

Zhou, P. – Tang, J. (2021) Improved method of stock trading under reinforcement learning based on drqn and sentiment indicators arbr.

Zhu, T. – Zhu, W. (2022) Quantitative trading through random perturbation q-network with nonlinear transaction costs. *Stats*, vol. 5 (2), 546–560, URL: `https://www.mdpi.com/2571-905X/5/2/33`.

APPENDIX I

Listing 1: Python code for setting up the trading limits

```
#Function for calculating ATR based on past 14 trading days
def calculate_atr(self):
    true_range = []
    for i in range(14):
        #Calculate daily true range
        true_range_daily = max(self.data['High'][self.t-i] - self.data['Low'][self.t-i
            ], abs(self.data['High'][self.t-i] - self.data['Close'][self.t-i-1]), abs(
            self.data['Low'][self.t-i] - self.data['Close'][self.t-i-1]))
        #Add to list
        true_range.append(true_range_daily)
    #Calculate the rolling ATR
    atr = sum(true_range)/len(true_range)
    return atr

#Function for updating the stop loss and take profit levels for long position
def update_limits_long(self):
    #Reset the stop loss / take profit boolean
    self.stoploss_triggered = False
    self.takeprofit_triggered = False

    #Call calculate atr function
    atr = self.calculate_atr()

    #Set the multiplier for ATR
    multiplier = 1.5

    #Calculate stop loss and take profit levels
    self.stop_loss = self.data['Close'][self.t] - (atr * multiplier)
    self.take_profit = self.data['Close'][self.t] + (atr * multiplier)
    return self.stop_loss, self.take_profit


#Function for updating the stop loss and take profit levels for short position
def update_limits_short(self):
    #Reset the stop loss / ake profit boolean
    self.stoploss_triggered = False
    self.takeprofit_triggered = False

    #Call calculate atr function
    atr = self.calculate_atr()

    #Set the multiplier for ATR
    multiplier = 1.5

    #Calculate stop loss and take profit levels
    self.stop_loss = self.data['Close'][self.t] + (atr * multiplier)
    self.take_profit = self.data['Close'][self.t] - (atr * multiplier)
    return self.stop_loss, self.take_profit
```

APPENDIX II

Listing 2: Python code for modified reward function

```
#Set the default reward (returns)
default_reward = self.data['Returns'][t]

#Choose the reward based on position
#If agent is long position
if self.data['Position'][t] == 1:
    #Check if close price is below stop loss level and the stop loss has not been
        triggered yet
    if self.data['Close'][t] <= self.stop_loss and self.stoploss_triggered == False:
        reward = default_reward-self.stoploss_penalty
        self.stoploss_triggered = True
        self.data['StopLossTriggeredLong'][t] = True

    #Check if close price is above take profit level and the take profit has not been
        triggered yet
    elif self.data['Close'][t] >= self.take_profit and self.takeprofit_triggered ==
        False:
        reward = default_reward+self.takeprofit_reward
        self.takeprofit_triggered = True
        self.data['TakeProfitTriggeredLong'][t] = True

    #If any of the levels are not triggered, give default reward
    else:
        reward = default_reward

#If agent is short position
elif self.data['Position'][t] == -1:
    #Check if close price is above stop loss level and the stop loss has not been
        triggered yet
    if self.data['Close'][t] >= self.stop_loss and self.stoploss_triggered == False:
        reward = default_reward-self.stoploss_penalty
        self.stoploss_triggered = True
        self.data['StopLossTriggeredShort'][t] = True

    #Check if close price is below take profit level and the take profit has not been
        triggered yet
    elif self.data['Close'][t] <= self.take_profit and self.takeprofit_triggered ==
        False:
        reward = default_reward+self.takeprofit_reward
        self.takeprofit_triggered = True
        self.data['TakeProfitTriggeredShort'][t] = True

    #If any of the levels are not triggered, give default reward
    else:
        reward = default_reward

#If agent does not have short or long position (such as beginning of simulation), give
    default reward
else:
    reward = default_reward
```