A COMPARATIVE ANALYSIS OF RECURRENT NEURAL NETWORKS AND DEEP REINFORCEMENT LEARNING ON STOCK PRICE PREDICTION PERFORMANCE

Supervisor: Dr. Xianfang Sun

Author: Adam Tagg

Computer Science BSc

School of Computer Science & Informatics

Cardiff University

May 2024

Abstract

This project is a comparison of two deep-learning fields, Deep Reinforcement Learning and Recurrent Neural Networks. In recent years, deep learning models have been widely applied to stock market prediction and have achieved good performances. However, the majority of these deep learning-based models belong to supervised learning methods and Recurrent Neural Networks like Long Short Term Memory (LSTM). The aim of this dissertation is to design a Deep Reinforcement Learning (DRL) model which will make profitable trades in a simulated trading scenario. Inspired by the works of Yong Shi et al (2021) in their study "Stock Trading Rule Discovery with Double Deep Q-Network", this project utilises and compares both Long Short-Term Memory and Deep Q-Learning algorithms to move away from the predefined algorithm “norms” and explore the field from a different perspective.

Acknowledgements

I am immensely thankful to the teaching staff of my course, whose insights and unwavering patience were crucial in sharpening my understanding and passion for the subject. Their dedication not only educated but truly inspired me.

I owe a large amount of gratitude to my supervisor for being willing to meet with me every week and respond to any queries I had throughout the process.

I also extend my deepest gratitude to my family and friends, whose unyielding support and frequent distractions helped transform the arduous process of writing into an enjoyable experience with you all. Your encouragement and warmth made all the difference. Thank you for being part of this journey.

Table of Contents

Introduction

Companies looking to increase capital and the value of their company often turn to the issuance of stocks and shares to the public and other private equity companies. Investors around the world are therefore able to invest their wealth into these companies to make a return. Many deem this a lucrative avenue for investment, although very risky as it can lead to wealth loss if the stock prices downturn. The cause of fluctuations in stock price is commonly due to multiple factors, often unpredictable and difficult for human analysts and researchers to foresee. Therefore, the central research question of this dissertation concerns the development of such a tool: can stock prices be accurately predicted by machine learning algorithms?

In the field of deep learning, predicting stock prices to a high degree of accuracy remains an unconquered challenge that draws a substantial amount of attention from researchers. The volatile nature of the stock market makes precise prediction a difficult task. However, the financial gains on offer are enough to keep interest in this field high. Over recent years, machine learning has proved to be a powerful tool in tackling this problem. This dissertation aims to further explore the potential of machine learning in stock price prediction with a particular focus on DRL–an area that promises to enhance AI decision-making processes in stock trading.

The conventional methods of stock price prediction have predominantly utilised various forms of regression analysis, time series forecasting, and, more recently , neural network-based models such as Long Short Term Memory (LSTM) networks. However, these methods, particularly supervised learning techniques, often rely heavily on large, labelled datasets and predefined relationships which may not adequately capture the complex scope of non-linear interactions typical to financial markets. The heavy reliance by conventional methods on historical data alone makes these algorithms fall short without considering the capacity to learn and adapt strategies in real-word scenarios. Therefore, the question that this dissertation aims to answer is: can reinforcement learning live up to the proven accuracy of conventional models or even exceed them?

Aims and objectives

The primary objective of this dissertation project is to use a DRL model to predict the upcoming trend direction of a stock price. This project will utilise LSTM and DQN models to perform these predictions, the former as a control model and the latter as the evaluation model. Additionally, this project will introduce a test environment for the models to simulate trading actions close to a real-world scenario.

Given this general aim, the project can be split into more specific objectives, outlined as follows:

- Complete a **literature review** on similar studies to find avenues for exploration.
- Collect and **preprocess** the historical stock market data needed for training, validating and testing the models.
- Calculate **technical indicators** to use as new features to improve model performance.
- Develop a refined stock price movement prediction, **Deep Reinforcement Learning model** for each stock.
- Create a control model for predicting stock price movements to serve as a **performance benchmark**.
- Develop a **simulated environment** to test model predictions and track account metrics.
- Evaluate the **performance** of these models on unseen test data from a different time period, comparing their evaluation metrics and profit/loss.
- Perform an **analysis** comparing the strengths and weaknesses of each model, discussing why one might perform better than another.

Background

This chapter provides a foundational understanding of the stock market and introduces technical analysis, which is key to predicting market movements through historical data. Key economic concepts covered include the Simple Moving Average (SMA), Exponential Moving Average (EMA), Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), and Stochastic Oscillator.

The chapter also explores advanced machine learning methods such as DRL and DQNs, highlighting their potential for making price predictions. Furthermore, it details LSTM networks, and the effectiveness of its algorithms in handling time-series data for stock price forecasting.

Stock market

The stock market plays an important role in the global economy, acting as a barometer for the financial health of a country. Private equity companies rely on the stock market to raise capital for growth through the issuance of stocks (initial public offerings or IPOs) (Baker and Wurgler, 2007). The stock market comprises a complex network of financial institutions, including exchanges like the New York Stock Exchange and the NASDAQ, where securities such as stocks and bonds are bought and sold. Investors, ranging from individuals to large institutional entities, engage in trading based on a multitude of factors, including: economic indicators, company performance, and geopolitical events. Investors, therefore, are wary of the dynamics of the stock market and how they are influenced by supply and demand, investor/market sentiment and broader macroeconomic forces (Shleifer and Vishny, 1986).

The performance of the market is often encapsulated in indexes, such as the S&P 500 or the Dow Jones Industrial Average, which aggregate the value of select groups of stocks. These indexes represent a weighted average of the performance of a specific set of companies, providing a valuable gauge of market trends and a benchmark for investment decisions (Fama and French, 1993). Understanding the intricacies of the stock market is essential for participants to make informed decisions and to navigate the fluctuations inherent in financial markets.

Technical Analysis

Technical analysis is a methodology that investors seeking to predict future market movements employ by analysing past and current market data, primarily focusing on price and volume. Unlike fundamental analysis, which delves into a company's financial statements and economic factors, technical analysis is rooted in the guideline that historical price actions and market trends can provide indicators for future price movements (CFA Institute, 2023).

Chart patterns, technical indicators, and other charting tools are often used to identify potential buying and selling opportunities. This methodology is grounded in the theory that market prices move in trends and that these trends, once established, typically continue for a foreseeable future (Investopedia, 2024). Technical indicators such as moving averages, MACD (Moving Average Convergence Divergence), RSI (Relative Strength Index), and Stochastic Oscillator allow technical analysts to assess market sentiment and investor behaviour to pinpoint potential reversals or continuation of trends in hopes of acting on their analysis to turn a profit (Corporate Finance Institute, Technical Analysis, 2020). An overview of the technical indicators used in this project are explained in the upcoming subsections.

Simple Moving Average

The Simple Moving Average (SMA) is a technical analysis indicator used to calculate the average price of a stock over a set period ‘p’. SMA smooths out price fluctuations by averaging and removing the daily

effect of price movements (Akkaynak, 2023). The benefit being that instant volume changes are prevented from creating a misleading effect on the investor (Daniswara, Widjanarko and Hikmah, 2022). SMA results provide a tool for financial analysts to use as a buy or sell signal and is calculated using the following formula (Investopedia, 2021):

$$SMA_n = \frac{P_1 + P_2 + \ldots + P_n}{n}$$

Where P_n is the price of an asset at period n and 'n' is the number of total periods.

Exponential Moving Average

The Exponential Moving Average (EMA) is a moving average that has a weighted bias towards recent data which allows the indicator value to react swiftly to price changes. Because of this, compared to the SMA, EMA follows current market movements more accurately. The EMA is calculated using the following formula (Investopedia, 2021):

$$EMA_t = (C \times \alpha) + \left(EMA_{-1} \times (1 \times \alpha)\right)$$

Where C is the current price value, α is the smoothing factor, and EMA-1 is the previous day's EMA.

Moving Average Convergence Divergence

Moving Average Convergence Divergence (MACD) is a trend-following momentum indicator to help investors identify price trends, measure trend momentum, and identify market entry points for buying or selling (Investopedia, 2021). The MACD is calculated using two different periods of EMA. The default period values are a 26 day and 12-day EMA. See equation (ibid):

$$MACD = EMA_{12} - EMA_{26}$$

Additionally, the MACD has multiple secondary components:

- The **signal line** - conventionally a 9-day EMA of the MACD line itself. The signal line is used as a trigger when it crosses the MACD as a buy and sell signal.
- The MACD **histogram** - measures the distance between the MACD line and the Signal Line, helping to identify when these crossovers are about to occur.

Relative Strength Index

The Relative Strength Index (RSI) measures the speed and change of price movements and is referred to as an oscillator in technical analysis. The RSI value can be interpreted for many signals but mainly

helps identify over-buying/selling and as a momentum shift indicator. An asset is considered over-bought when the RSI value is above 70 or oversold when below 30. The RSI is calculated using average price increase and decreases over a specified period of time; the standard period t is 14 periods. See formula below (Investopedia, 2021):

$$RS = \frac{U_t}{D_t}$$

Where U_t represents Average of t days up closes and D_t represents Average of t days down closes

$$RSI = 100 - \left(\frac{100}{1 + RS}\right)$$

Stochastic Oscillator

The Stochastic Oscillator, similarly, to the RSI is a range-bound momentum indicator for observing overbought and oversold conditions. The stochastic oscillator follows the speed or momentum of price which can foreshadow reversals because the momentum or speed of a stock's price movements changes before the price changes direction (Pruitt, 2016).The indicator consists of two lines: the %K line, which represents the current price of the security as a percentage difference between the high and low over the time period and the %D line, a 3 day moving average of the %K line. The following explains the formula of the %K line (Investopedia, 2021):

$$\%K = \left(\frac{C - L_{14}}{H_{14} - L_{14}}\right) \times 100$$

Where C is the most recent closing price, L_{14} and H_{14} are the lowest and highest prices traded over a 14-day period respectively.

Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a combination of Reinforcement Learning (RL) and Deep Learning. RL focuses on training agents to learn through trial and error, interacting with an environment and receiving rewards for desirable actions (Sutton and Barto, 2018). Deep learning works at extracting patterns from complex, high-dimensional data using artificial neural networks (LeCun, Bengio and Hinton, 2015). By combining these strengths, DRL enables agents to act with precision in environments where traditional methods struggle.

The goal of the DRL agent is to learn an optimal **policy**, a mapping from states to actions, that maximises its long-term reward (Mnih et al., 2015). For each action an agent takes in a given state, the environment provides a reward (a numerical *feedback signal)* indicating the desirability of that action. This reward is either positive for actions deemed beneficial (e.g., profiting from selling a stock) or negative for actions deemed detrimental (Sutton and Barto, 2018). See Figure 1 for an overview of a DQN model.

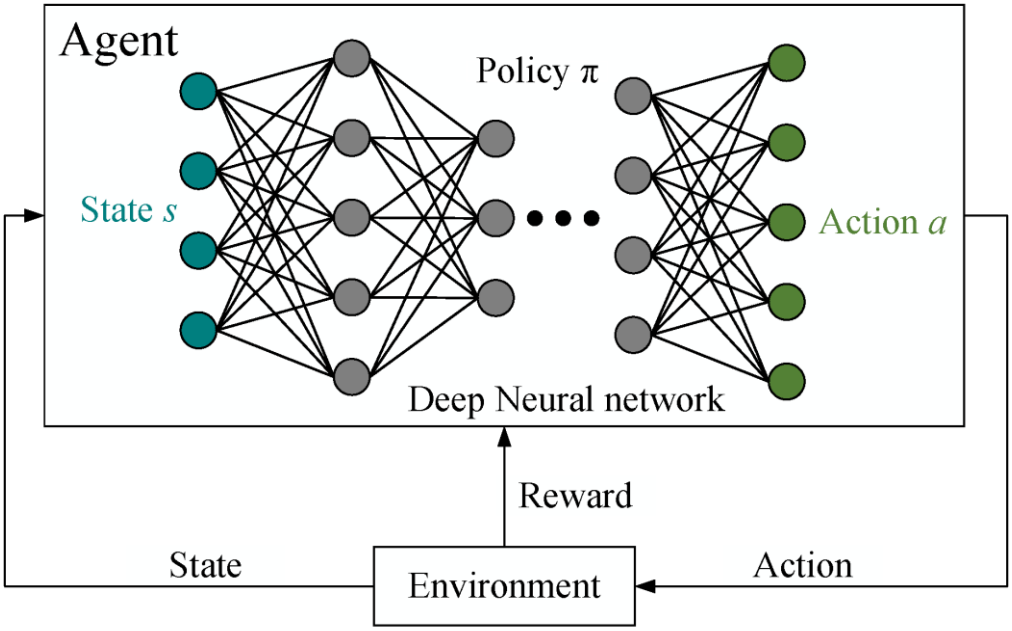


Figure 1: Deep Reinforcement Learning (Yi and Liu 2023)

Deep Q-Networks

Mnih et al. (2015) introduced Deep Q-Networks (DQNs) marking a significant breakthrough in reinforcement learning (RL). DQNs combine deep neural networks with Q-learning, a DRL algorithm, enabling agents to tackle complex environments with high-dimensional state spaces (Watkins and Dayan, 1992). This is achieved by using a neural network to approximate the Q-value function, which estimates the future reward an agent can expect from taking a specific action in a given state (Mnih et al., 2015).

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') max_{(a')} Q(s',a')$$

To address instability issues that can arise during training, DQNs utilise two key techniques: **experience replay** and **target networks** (Mnih et al., 2015). The structure of DQNs rely heavily on these two modules. Figure 2 shows the architecture of a DQN including target network and replay memory.

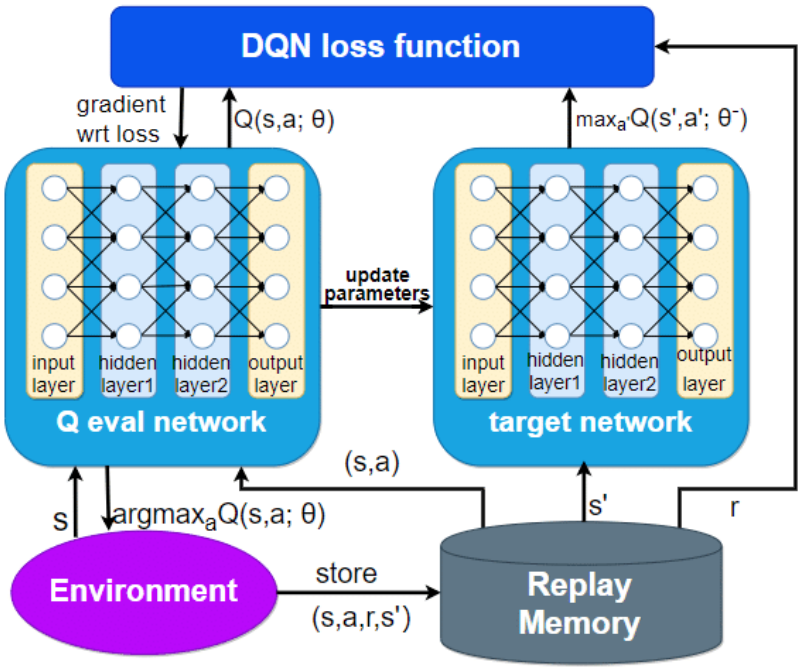


Figure 2: DQN Structure (Zhou 2019)

Experience Replay

Experience replay (also known as replay memory) allows the agent to learn from a set of past experiences, improving the training data's efficiency and stability by reusing each transition in many updates, and using uncorrelated transitions in a batch (Lin, 1992). At each time step of data collection, the transitions are added to a *replay buffer*. Then during training, instead of using just the latest transition to compute the loss and its gradient, they're computed using a smaller batch of transitions sampled from the replay buffer (TensorFlow, 2023).

Target Networks

Target networks, introduce a separate network with the same architecture as the main Q-network but with periodically updated weights. This helps stabilise the learning process and prevents the Q-values from overestimating (Mnih et al., 2015).

Long Short-Term Memory

Long Short-Term Memory Networks, known as LSTMs, were introduced by Hochreiter & Schmidhuber in 1997. They are a specialised type of Recurrent Neural Network (RNN) designed to address the problem of learning long-term dependencies as traditional RNNs struggle to capture long-distance relationships within the data due to issues such as vanishing and exploding gradients (Hochreiter and Schmidhuber, 1997). LSTMs have shown great success in a variety of sequence learning tasks due to their ability to capture long-range dependencies in sequence data, and as such are well-suited to time series problems.

An LSTM unit is composed of a cell state and three regulatory gates: the input gate, the forget gate, and the output gate.

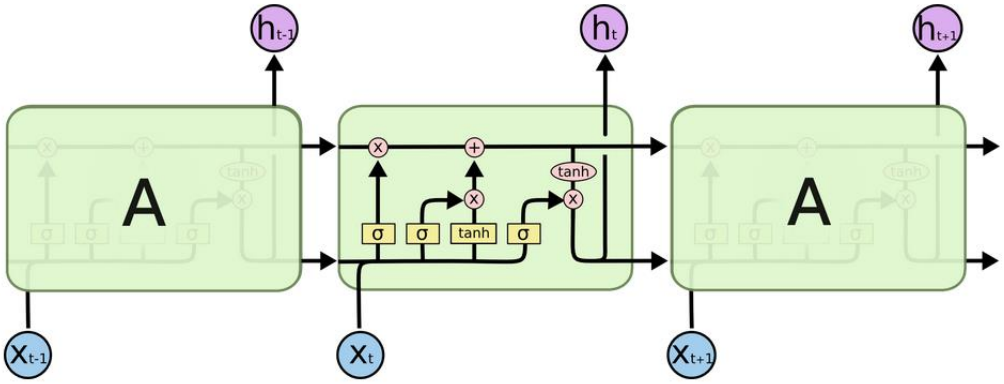


Figure 3: LSTM Architecture (Olah 2015)

Cell State ($C_t/C_{(t-1)}$):

The cell state, seen in figure 3 (represented as A), acts as the memory of the LSTM, carrying relevant information throughout the processing of the sequence. It gets updated or maintained at every time step based on the inputs from the forget and input gates.

Forget Gate (f_t):

The forget gate decides what information should be thrown away from the previous cell states. It processes the current input x_t and the previous output/hidden state h_{t-1} and determines a number between 0 and 1 for each number in the cell state $C_{(t-1)}$. A value of 1 represents that full data should be kept while a value of 0 means that the data should be discarded entirely. The forget gate is calculated using the following formula (Khalil et al., 2019):

$$f_t = \sigma\left(W_f \cdot \left[h_{(t-1)}, x_t\right] + b_f\right)$$

Where σ is the sigmoid activation function, W_f is a weight (hyper-parameter) for the forget gate h_{t-1} is the last hidden state, and b_f is an added bias.

Input Gate (i_t):

The input gate calculates the amount of new information to be added to the cell state. It first calculates a candidate vector i_t, which is the result of the current input x_t and the previous hidden state $h_{(t-1)}$. The candidate vector may contribute new information to the cell state depending on its value. To keep the value between a range of -1 and 1, a tanh function is used on a sigmoid layer. See the following for the associated formulas (Khalil et al., 2019):

$$i_t = \sigma\left(W_i \cdot \left[h_{(t-1)}, x_t\right] + b_i\right)$$

$$\tilde{C}_t = tanh\left(W_C \cdot \left[h_{(t-1)}, x_t\right] + b_C\right)$$

Where W_t is a weight (hyper-parameter) for the input gate, b_t and b_C are biases, $\acute{C}_t$ is the candidate values after tanh normalisation.

Output Gate (o_t):

The output gate controls the extent to which the value in the cell state is used to compute the output activation of the LSTM unit. The output is determined using the previous hidden state, the updated cell state and the input data. Employing sigmoid activation again, the cell state is passed through the tanh function to normalise values to be between -1 and 1. The normalised value is multiplied by the output of the sigmoid gate and therefore only the determined parts are allowed through, creating network stability over time. The formulas for the output gate are detailed below (Khalil et al 2019, p. 1886):

$$o_t = \sigma\left(W_o \cdot \left[h_{(t-1)}, x_t\right] + b_o\right)$$

$$C_t = f_t * C_{(t-1)} + i_t * \tilde{C}_t$$

$$h_t = o_t tanh(C_t)$$

Where W_o is a weight for the output gate, b_o is an added bias, and C_t is the final memory state.

Related Work

Throughout recent times, using computers for forecasting stock prices has risen in significance with leading computer scientists and economists, assisted by deep learning, collaborating in an effort to solve this highly problematic task. This chapter covers a review of the existing literature revealing numerous comprehensive attempts at predicting stock prices or their trend movements. This review focuses on research that uses reinforcement learning algorithms as their focus model in an effort to highlight a more niche area of research on this topic. Furthermore, this review identifies areas of stock price prediction where classic assumptions fail and their causes.

In their study, Kabbani and Duman (2022)apply DRL for automating trading strategies in the stock market. The model aims to maximise financial gains through allowing the model to buy, hold, or sell stocks, based on technical indicators such as price momentum and trading volume. Validated using data from the S&P 500 index over five years (2015-2020), the DRL model outperforms traditional trading methods, including the Buy and Hold strategy, by delivering superior Return on Investment (ROI). This result highlights the DRL model's effectiveness in navigating the complex market environment and emphasises its potential for transforming automated trading strategies. Using a Partially Observed Markov Decision Process (POMDP) and the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, the model achieved a 2.68 Sharpe Ratio on test data, demonstrating an advantage for DRL's in strategic financial decision-making.

Furthering research into DRL solutions for stock price forecasting, a study by Shi et al. (2021) applies a double DQN model on American and Chinese stock market data throughout a 3-year training-period from 2015 to 2018- and 4-month testing period. The results of this work show that the deep learning architecture with Convolutional Neural Network (CNN) layers in a policy network are able to extract latent dependency in the stock data. The experiment also indicates the proposed Double DQN reduces overestimation and improves the stability of the model in comparison to its LSTM counterpart. The study not only confirms the robustness of DRL for high-frequency trading but also highlights the differential impacts of market-specific regulations on the performance of trading algorithms. For instance, the prohibition of short selling in the Shanghai Stock Exchange (SSE) during the testing phase imposed unique constraints, yet the Double DQN model adjusted effectively, outperforming Support Vector Machine (SVM) in both accuracy and return rates. Similarly, in the NASDAQ settings, the double DQN allowed for more flexible trading strategies like short selling, the model demonstrated significant superiority over traditional methods by still managing high accuracy and better risk management for volatile stocks. This indicates the adaptability of the Double DQN model to diverse regulatory environments and its potential as a reliable tool in price forecasting.

A performance analysis of regression-based machine learning techniques aimed at predicting stock market movements elucidates the heavy reliance of supervised machine learning models on historical stock data, a methodology that, while traditional, is plagued with constraints. The paper by Nti, Adekoya and Weyori (2021) reveals that despite using sophisticated algorithms such as Linear Regression, Ridge Regression, and Support Vector Regression (SVR), models exhibit a notable variance in performance metrics—highlighting inherent limitations in capturing the unpredictable dynamics of stock markets solely through historical data. The study showcases comparative accuracies across different models: Linear Regression achieving a median accuracy of 55%, Ridge Regression slightly higher at 58%, and SVR leading with 62% accuracy over a testing period encompassing data from 2010 to 2018. These figures underline the dependency dilemma—where past data's over-utilization potentially leads to models that inadequately forecast future market trends due to their inherent inability to account for

unforeseen market influences, highlighting a crucial gap in predictive precision that current methodologies fail to bridge.

In a separate paper "Performance Analysis of Regression-Based Machine Learning Techniques for Prediction of Stock Market Movement" by Sakhare and Sagari (2019), the effectiveness of regression-based machine learning methods in predicting stock market trends is explored. The authors highlight how heavily supervised models depend on historical data. They point out the significant limitation of such a dependency, emphasising that past data might not always accurately forecast future market behaviours, which could lead to errors in predictions. Through their analysis, they attempt to determine the most accurate regression method for stock market prediction by comparing Linear Regression, Polynomial Regression, and SVR, applying these techniques to the S&P 500 index data. Specifically, the research presents the following findings:

- Linear Regression yielded an average standard deviation of 29.70 and a Mean Squared Error (MSE) of 0.6675.
- Polynomial Regression showed improvements with an average standard deviation of 25.59 and an MSE of 0.5537.
- SVR outperformed the other techniques significantly, presenting an average standard deviation of 5.694 and an MSE of 0.1429.

SVR emerged as the most effective method among those tested, demonstrating lower error rates and greater reliability in forecasting stock market trends.

The above review gleams that the growing use of DRL, tested in varied market conditions including top indices like S&P 500 and cross-market setups such as NASDAQ and SSE, sometimes outperform traditional models by adapting well to regulatory differences and extracting deeper market dependencies. Despite the capabilities of regression methods and DRL, their reliability is stunted by over-dependence on historical data, which often fails to predict future market movements accurately.

Methodology

This chapter outlines the procedures and methods used to predict and execute profitable trades in the stock market, focusing on the application of machine learning techniques. Each step in this chapter is tied to the overarching goal of creating a profitable DRL trading model.

Overall Approach

1. **Stock Selection**
    - **Objective**: To ensure robustness and generalisability across various market conditions, stocks from different sectors should be selected. This choice allows the model to handle diverse economic conditions and enhances the relevance of the findings.
    - **Process**: Stocks are chosen based on liquidity, market capitalization, and representation across industrial sectors. This diversity helps in training more adaptable and resilient models.

2. **Data Collection**
    - **Objective**: To gather a comprehensive dataset that captures a wide range of market behaviours over economic cycles.
    - **Process**: Utilising 'yfinance', a Python API for Yahoo! Finance, historical stock data spanning two years (2021-2023) is collected for training. An additional six months (January to June 2023) is included for testing and validation to evaluate the models under near-current market conditions.

3. **Data Preprocessing**
    - **Objective**: To refine the dataset ensuring cleanliness and relevance, which aids in effective model training and testing.
    - **Process**: The dataset undergoes a cleaning process to remove any incomplete, incorrect, or irrelevant data that could distort model training results.

4. **Feature Engineering / Data Sampling**
    - **Objective**: To enrich the dataset with features that are likely to have significant predictive power regarding stock price movements.
    - **Process**: New features, such as the Simple Moving Average (SMA) and other technical indicators, are computed and added to enhance the dataset. These features are selected based on their historical performance in technical analysis scenarios.

5. **Test Environment Setup**
    - **Objective**: To create a simulated market environment where models can be safely tested and tuned without incurring real-world financial risks.
    - **Process**: A virtual trading environment is configured to mimic real market conditions as closely as possible, providing a robust stage for assessing the performance of the trading strategies developed by the LSTM and DQN models.

6. **Model Development and Training**
    - **Objective**: To construct and fine-tune models that can reliably predict market movements and execute trades with high accuracy.

- **Process**: The LSTM model is set up to predict future stock prices based on time-series data, while the DQN model is designed to learn optimal trading strategies. Both models are trained using the previously prepared and enhanced data.

7. **Performance Evaluation**

- **Objective**: To assess each model's effectiveness using a series of performance metrics that reflect both trading profitability and machine learning accuracy.
- **Process**: After training, the models are evaluated based on achieved profit levels and metrics such as accuracy, recall, precision, F-score, Q-value, and loss. These metrics offer insights into the effectiveness of the models and guide further optimizations.

Implementation

This chapter outlines the implementation of this study on predicting stock price movements using machine learning models and prepares the ground for discussing key results derived from the experimental findings. Starting with an explanation of the development environment and subsequent sections focusing on data collection, preprocessing, and the detailed architecture of both discussed models, to explain each models' configurations. The chapter then explores the simulated environment designed for the models, enabling the DQN agents to perform actions and LSTM in simulating its predictions. Lastly, it presents the dual evaluation strategies—covering traditional metrics and profit-back testing for further comparative analysis in the results section.

Development Environment

For the development of this project Google Colab (an online web-browser based platform) was used for writing and executing code. Google Colab is well suited for developing machine learning solutions, able to execute .ipynb (Jupyter Notebook) files in the cloud, removing local hardware limitations and alleviating time constraints, thus enabling the development cycle to be more efficient. The exact hardware specifications for the version of Google Colab used in this project can be found in appendix A.

Python was selected as the primary programming language for this project for one main reason; Python has a large community-based catalogue of code libraries, which significantly simplifies the implementation process especially for machine learning solutions. Libraries such as Pandas for data manipulation, NumPy for numerical calculations, and Matplotlib for data visualisation, offer powerful, and efficient solutions that are well suited for handling the speed and complexity requirements of machine learning tasks. Furthermore, these libraries are well optimised and have gone through many versions and updates throughout the years meaning that they would far exceed the performance of any code I write to achieve the same result. The python libraries used in this project are detailed in Table 1.

Table 1: Python Libraries

Library	Version	Description
NumPy	1.25.2	NumPy is a Python library that provides a multidimensional array object, various derived objects, and an assortment of routines for fast operations on arrays. (NumPy, 2024)
Pandas	1.5.3	Pandas is an open source data analysis and manipulation tool, built on top of the Python programming language. (Pandas, 2024)
PyTorch	2.2.1	PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.(PyTorch, 2024)
yfinance	0.2.37	yfinance offers a threaded and Pythonic way to download market data from Yahoo! Finance. (Aroussi, 2024)
ta	0.11.0	Technical Analysis library for financial time series datasets. Built on the Pandas library.(Padial (Bukosabino), 2023)

Matplotlib	3.7.1	Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.(Droettboom, 2024)
tqdm	4.66.2	Tdqm is a library to make your loops show a smart progress meter. (tqdm, 2024)

Data collection

As mentioned previously in the methodology, for training the machine learning models to predict stock price movements, historical stock price data from the U.S. stock market were used. This historical data facilitates the computation of technical indicators to enhance model performance. The data used in this project was collected from 5 major sectors to cover a wide range of stocks; Technology, Finance, Energy, Consumer goods and Healthcare. All data was sourced from Yahoo! Finance because its API allowed direct integration of the data into the Google Colab session. Table 2 gives the information of the chosen stocks and reasoning for doing so:

Table 2: Stock Selections

Name	Symbol	Sector	Reasoning
Apple Inc.	APPL	Technology	Apple, being one of the largest companies in the world by market cap, represents the tech sector and offers a versatile dataset due to its high trading volume and active analyst coverage. Its stock price is influenced by a wide range of factors, including product launches, global supply chain conditions, and broader tech industry trends.
JPMorgan Chase & Co.	JPM	Finance	As a leading global financial services firm, JPMorgan Chase is a good representative of the banking industry and the broader financial sector. Banking stocks are highly sensitive to interest rate changes, regulatory policies, and economic indicators, adding a different set of variables for the DQN to learn and predict.
Exxon Mobil Corporation	XOM	Energy	Exxon Mobil, one of the world's largest publicly traded oil and gas companies, is representative of the energy sector, which is closely tied to fluctuating commodity prices, geopolitical tensions, and changes in global energy policies. This introduces a different volatility pattern and external dependencies to model.
The Coca-Cola Company	KO	Consumer-Goods	Coca-Cola is a leading global beverage company, reflective of the consumer goods sector. This sector is often considered more stable and can provide a contrast to more volatile tech or energy stocks. Consumer goods companies can be impacted by different factors, including changes in consumer trends, global economic health, and commodity costs.
Pfizer Inc.	PFE	Healthcare	Representing the healthcare sector, Pfizer is one of the world's largest pharmaceutical companies. The

			healthcare industry can be influenced by product pipelines, regulatory approvals, and broader healthcare policy changes. Especially in recent years, it also factors in global health emergencies, making it a dynamic sector for prediction.

Data Preprocessing

Handling Missing Values

The dataset may contain missing or "null" values, which can negatively affect model performance or stop it from training at all. Some null values appear after data collection due to incomplete data sets or more often appear in the columns of calculated technical indicators when there is not enough data for computation. For instance, a 14-day EMA will have null values for the first 13 days, as there are not enough prior days to calculate the average. Consequently, rows with these missing values are typically found at the beginning of a stock's period are removed.

Feature engineering – Technical Indicators

Feature engineering is essential for enhancing the predictive power and accuracy in machine learning models, as it transforms raw data into features with greater relevance and impact on the classification or prediction tasks (Rawat and Khemchandani, 2016). Therefore, utilising the advantages of feature engineering is necessary in my project to push the performance of the trained models.

The most common features available from raw stock data and commodity prices are calculating technical indicators, therefore these are the main features that are used. Using multiple technical indicators in the same dataset can enhance the accuracy of market analysis, therefore, this project utilised five separate technical indicators over three different types. To calculate the indicators , the 'ta' (technical analysis) Python library was used. Each indicator has unique configuration parameters, and some technical indicators produce multiple lines, and each line represents different information. For easier understanding, the technical indicators and their parameters are shown in Table 3.

Table 3: Technical Indicator Parameters

Indicator	Parameters
Simple Moving Average (SMA)	Period = 20
Exponential Moving Average (EMA)	Period = 20
Moving Average Convergence Divergence (MACD)	Fast = 12 Slow = 26 Signal = 9
Relative Strength Index (RSI)	N/A
Stochastic Oscillator	N/A

Feature Scaling

Feature scaling is usually necessary when working with indicators such as SMA, EMA, and MACD as they are dependent on the price levels of the asset being analysed. However, feature scaling is not strictly necessary here as the models are trained on only one stock at a time (see model selection) and RSI and Stochastic Oscillators are inherently normalised.

Data Splitting

The gathered dataset is divided into three groups. The first period for training both models, the second period for validating the LSTM model during training and the last period is for testing both model's performances. *Note that the second period is not used by the DQN as RL agent validation is achieved by assessing how the trained agent performs on its intended task*

See code excerpt below for visualisation:

```
training_data = data['2021-01-01':'2023-01-01']
validation_data = data['2023-01-02':'2023-03-01']
predicitve_data = data['2023-03-02':'2023-06-01']
```

Trading Environment

Letting DRL agents perform actions inside of a simulated environment is important for assessing the performance of a given agent. Using python, a trading environment class has been implemented to enable the DQN agent to take actions and earn a reward. It has a customizable initial cash balance and allows performing purchasing actions such as buy, sell and hold for each day of stock data. The following subsections provide insight into how the environment and its functions operate.

Initialization

The environment is initialised with the collected stock price data in a pandas DataFrame format. The default starting cash balance is set at a default $12,000, although customisable. The primary attributes upon initialization include:

- `stock_price_history`: A DataFrame housing the stock data.
- `initial_balance`: The initial cash reserve for trading.
- `n_step`: Total number of trading steps or days for which the data is provided.

The initial balance default of $12,000 was chosen because the median stock market holdings for families earning an average income of $35,000 to $52,999 is $12,000 (Pino, 2024).

Reset Function

Upon resetting, the environment

1. Sets the current step to zero,
2. Resets the account balance to its initial state,
3. Clears any holdings of stocks,
4. Maintains a record of actions and the account value through the trading period.

Step Function

This is the main function for simulating trading actions in the environment. Each trading action received by the step function leads to:

- An update in stock shares held, based on the action taken (buy or sell).
- An update in account balance as shares are bought or sold.
- A re-calculation of total asset value (cash + value of held shares).
- Progression to the next day in the stock price data.
- Computation of the immediate **reward** as the difference in total asset value pre and post-action.
- Indication of whether the end of data has been reached, which terminates the episode.

Action Handling

Three possible actions can be handled:

- `0 (Buy)`: Uses all available balance to buy stocks at the current price.
- `1 (Hold)`: No change in the holdings or account balance.
- `2 (Sell)`: Sells all held stocks at the current price, adding proceeds to the balance.

To map the LSTM models price prediction into the trading environment's action space a simple case statement is employed. See pseudocode below:

```
if next_price > current_price

     buy signal

else if  next_price < current_price:

     sell signal

else
```

Model Selection – DQN

DQN has been chosen as the primary reinforcement learning algorithm for predicting stock prices due to several unique advantages it offers in the context of financial markets. Central to its selection is the ability of DQN to effectively handle high-dimensional state spaces, a common characteristic of stock trading environments where numerous indicators (e.g., SMA, EMA, MACD, RSI, STOCH) must be simultaneously considered. The advantages of the DQN model are highlighted below:

DQN combines Q-learning with deep neural networks, enabling the model to learn the optimal action-selection policy in a sequential decision-making framework. This **capability to learn optimal actions** is crucial for the trading environment where each action's impact evolves over time and depends complexly on the market state. Financial markets are predominantly non-linear and sometimes demonstrate abrupt changes. DQNs are adept at **identifying and adapting to non-linear relationships** within data, helping to capture complex patterns and anomalies that other models might miss. Lastly**,** the use of **experience replay** in DQNs allows the model to remember and reuse past experiences at random similarly to LSTM. This method is beneficial in stock trading where older data might still hold relevance due to cyclical trends.

Model Selection – LSTM

LSTM networks have been selected as the control or comparison model due to their proven proficiency in time-series forecasting, which is directly applicable to stock price predictions (Fischer and Krauss, 2018). The advantages of the LSTM model are highlighted below:

LSTMs are specifically designed to address issues like short-term memory in traditional RNNs, making them highly capable for forecasting tasks where understanding long-term dependencies and **handling time-series data** is crucial—characteristic of stock price movements. Furthermore, since LSTMs are well-established in financial forecasting, they **provide a good benchmark** to evaluate the novelty and efficiency of reinforcement-based approaches like DQN in trading. Determining where DQN provides additional benefits over traditional LSTM can highlight the specific contributions of reinforcement learning in financial contexts.

This dual-model approach utilising both DQN and LSTM allows for a comprehensive analysis of performance in stock price prediction for active trading. While DQN offers a strategic advantage through its policy-based learning and adaptability to the environment's dynamics, LSTM serves as a robust benchmark with its strong suit in capturing time-related dependencies. Comparing these methodologies helps illuminate their respective strengths and limitations in predicting stock prices and assisting in developing more refined models for trading.

Model Architecture – DQN

The architecture of the DQN is implemented using PyTorch. The network starts with an input layer designed to receive an array matching the number of financial indicators used. Following this, the network's configuration includes several layers.

The first layer is a fully connected (Linear) layer with 128 neurons, utilising the Rectified Linear Unit (ReLU) activation function to introduce non-linearity, essential for learning complex patterns in the data. Subsequently, another fully connected layer with 64 neurons continues to abstract and compress the learned features, also employing the ReLU activation function. The output layer consists of a fully connected layer with a size equal to `action_dim` (3 in this case, representing buy, hold, sell actions), which directly ties the neural network outputs to actionable trading decisions.
To achieve stable learning and prevent the network from over-fitting to the noisy financial data, several strategic choices were made in designing this architecture. Upon setting up the network structure of the DQN, the system requires compilation and optimization setup, enabling it to learn effectively from the simulated trading environment. The Adam optimizer was selected due to its adaptiveness in learning rates, making it highly suitable for the model given the dynamic nature of stock prices and the necessity for quick adaptation. The model utilises the Smooth L1 loss, a less sensitive loss function to outliers, which benefits the training process where prediction error variance is high, typical in financial applications.

The DQN model is configured to train through a series of episodes, with each episode representing a complete sequence from the start to the end of the available historical data. Training proceeds for a defined number of `episodes`, iterating over the trading data and using the gains from actions to inform subsequent decisions. Furthermore, the training leverages an epsilon-greedy strategy to balance the exploration of new trading actions and the exploitation of previously learned strategies. The value of epsilon decays exponentially to allow more exploitation as the model’s performance improves over episodes.

To combat issues related to correlated time-series data and improve learning stability, the model utilises replay memory, which stores previous states, the actions taken, and their outcomes. This allows the model to learn from a set of historical data points, smoothing over local fluctuations and enabling broader generalisation. A target network is also used, where the policy network’s weights are periodically copied to the target network, ensuring stable targets during temporal difference learning. See figure 4 below for a full visual description of the model architecture.

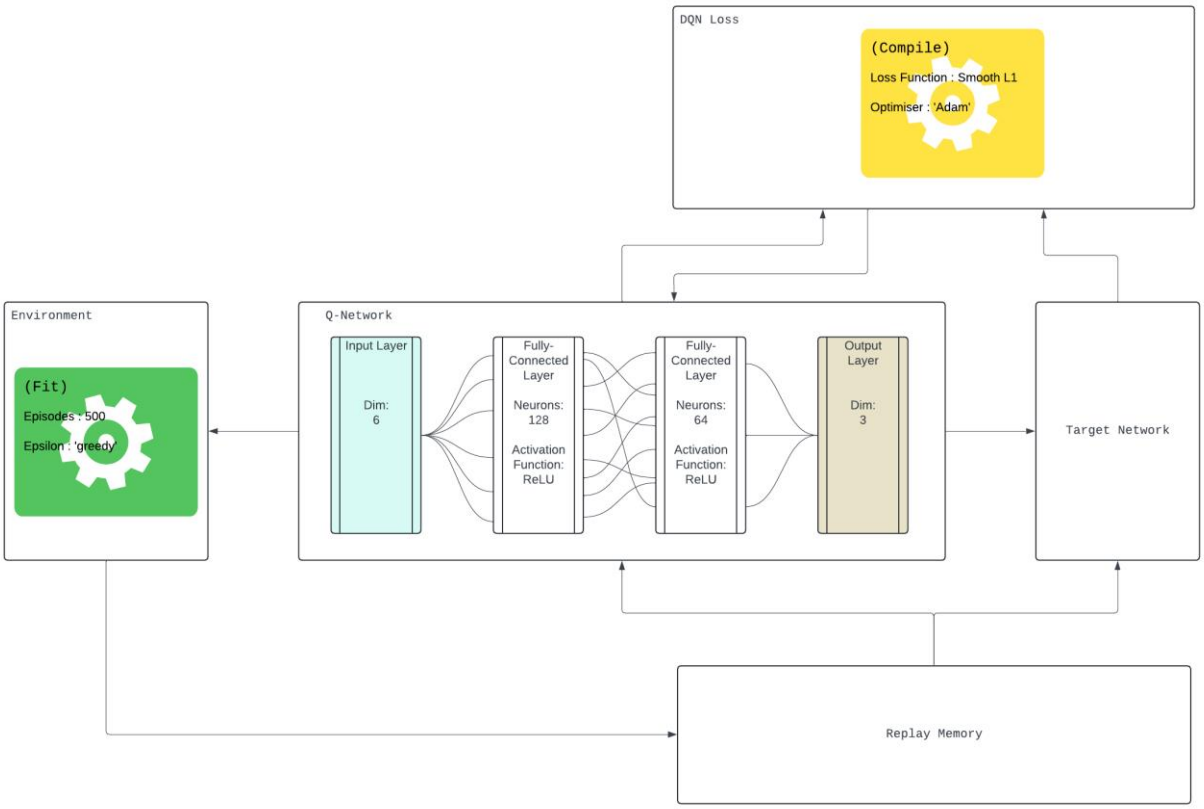


Figure 4: DQN Model Architecture

Model Architecture – Long Short Term Memory

The LSTM model's architecture is constructed using the neural network module functionalities provided by PyTorch. This model includes structured layers specifically designed for sequential data analysis, catering to the requirements of working with stock market data.

Starting with the LSTM layer, it is configured with variable input dimensions to adapt to the number of features derived from the stock data, commonly focusing on one crucial feature, the 'Close' price. Several LSTM layers are stacked, utilising a specified number of hidden neurons to encapsulate temporal dependencies effectively. This multi-layer setup ensures the model captures deeper temporal attributes without succumbing to common dilemmas like vanishing or exploding gradients, often encountered in traditional recurrent neural networks.

Following the LSTM layers, the processed outputs are directed to a Fully Connected (Linear) layer, which condenses the LSTM outputs to a singular continuous value depicting the predicted stock price. Within the LSTM architecture, a hyperbolic tangent (tanh) activation function is inherently implemented, optimising the transformation of processed data into actionable outputs directly.

During the model compilation stage, the Mean Squared Error (MSE) loss function is employed, targeting the minimization of the average squared differences between the predicted and actual stock prices. This loss function is especially apt for regression problems like price prediction, aiding the model in re-

fining its accuracy. For optimization, the Adam optimizer is selected for its robust computational efficiency and adaptive learning rate adjustments, enhancing the model's training effectiveness compared to standard stochastic gradient descent approaches.

Prior to actual training, the 'Close' prices are normalised using a “MinMaxScaler”, adjusting the data to range from -1 to 1. This normalisation aids in expediting convergence by ensuring uniform scales among input features. Data is prepared by correlating sequences of stock prices as inputs to their subsequent price outputs, formatted explicitly to suit the LSTM and the model uses iterative learning through epochs, continuously refining through batches of data to minimise the loss function.

Model Evaluation

Testing Strategy #1 – Evaluation Metrics

The first approach for evaluating the performance of the models on their task is by using their usual evaluation metrics, however, as the two models stem from differing algorithms, they also use different metrics. This means that little can be determined from the comparison of the two. Instead, these metrics serve as a benchmark of the models independent from one another.

DQN Metrics

As a reinforcement learning model, the DQN uses cumulative and average rewards as metrics because these metrics directly tie to its objective of maximising the total expected return throughout an environment. Unlike LSTM models, which prioritise accuracy and error reduction in sequence prediction tasks, DQN focuses on evaluating the efficacy and robustness of its decision-making policy. Average reward gives insight into the consistency and immediate effectiveness of the policy per episode, while cumulative reward measures the long-term success and overall impact of the policy across multiple episodes.

Average Reward: The average reward received during testing episodes, which indicates the effectiveness of the policy.

$$R_{AVG} = \frac{1}{N}\sum_{i=1}^{N}\sum_{i=1}^{T_i} r_{i,t}$$

Where R_{AVG} is the average reward per episode, N is the total number of episodes, T_i is the number of time steps or actions in the i-th episode and $r_{i,t}$ is the reward received at time step of episode .

Cumulative Reward: Total reward accumulated over a testing period or a specific number of episodes, which provides insight into the overall effectiveness of the learned policy.

$$R_{CUM} = \sum_{i=1}^{N}\sum_{i=1}^{T_i} r_{i,t}$$

Where R_{CUM} is the total cumulative reward and the rest retain their definitions as above.

LSTM Metrics

As a sequence prediction model, the LSTM uses accuracy, precision, recall, and F1-score as its primary evaluation metrics because these metrics directly assess the model's effectiveness in classification and prediction tasks. Unlike DQN models, which focus on the maximisation of rewards, LSTM models prioritise the accuracy of sequence prediction and the minimisation of classification errors. The correctness of a classification falls under one of four categories:

- **TP** (True Positives): Correct predictions that an instance is positive.
- **TN** (True Negatives): Correct predictions that an instance is negative.
- **FP** (False Positives): Incorrect predictions that an instance is positive.
- **FN** (False Negatives): Incorrect predictions that an instance is negative.

Accuracy: Accuracy provides a straightforward percentage of correct predictions across the dataset, reflecting overall model performance. The formula is as follows:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Precision and **recall** offer insights into the model's ability to correctly predict positive labels without significant error, important in cases where the cost of false positives or false negatives is high. The formula for precision is as follows:

$$\frac{TP}{TP + FP}$$

The formula for recall is as follows:

$$\frac{TP}{TP + FN}$$

F1-score combines precision and recall into a single metric that balances both, providing a holistic view of the model's accuracy. The formula is as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Testing Strategy #2 – Profit-Back Testing

The most insight that this project will give is from the second evaluation strategy; profit back testing. As the DQN uses an agent to perform actions in the stock market, the best way to evaluate performance is through the efficacy of those actions, therefore, a benchmark of profit is needed to compare both models against. For this benchmark the buy and hold trading strategy was chosen which simply buys as much stock as possible at the start and sells it all at the end. See Figure 5 for an example illustration.

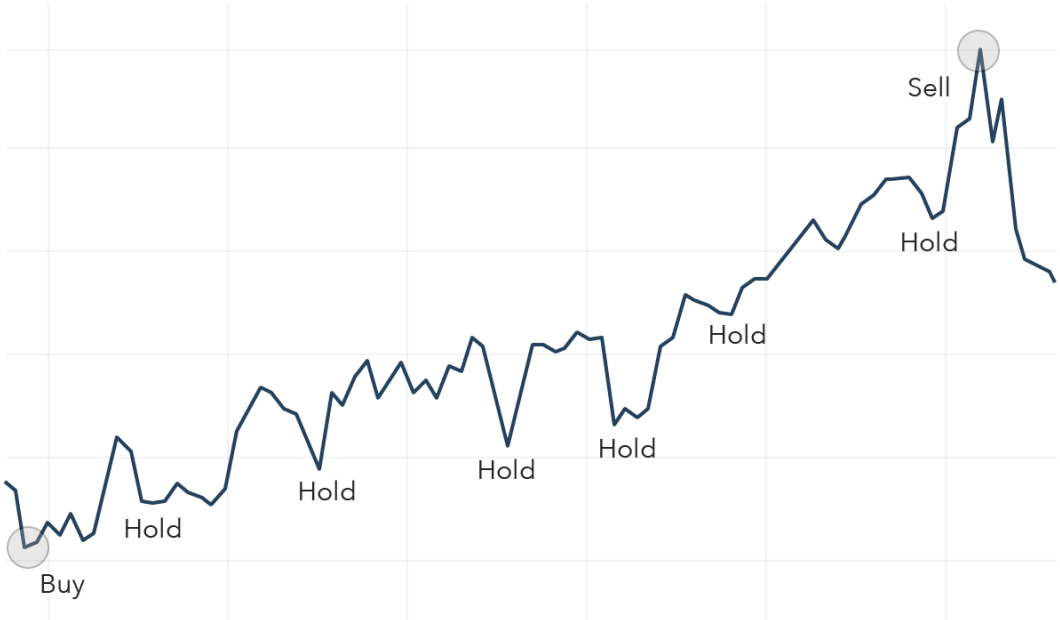


Figure 5: Illustration of Buy-And-Hold

Results and Discussion

Overview

The results section of this dissertation provides a comprehensive analysis of the performance of LSTM and DQN, applied to stock trading across five different stocks. The key aim was to assess whether the DRL represented by DQN could outperform the recurrent neural network approach of LSTM. Extensive testing and evaluation have been performed using multiple experiments to explore various aspects of each model's ability to predict stock prices and execute profitable trades.

For the LSTM model, evaluation metrics centre around accuracy and profit-back testing, featuring detailed insights into the model's robustness across various stocks. The DQN model results use a different set of metrics, focusing on cumulative and average rewards, to reflect the learning and decision-making optimization process over time. The discussion in the results section critically analyses these observed results, allowing for future research efforts to expand upon this work.

Results of LSTM Model

As mentioned earlier, five experiments have been conducted over the five chosen stocks (see Table 2). Details about the model's architectural design and the data preprocessing can be found in the implementation chapter. The LSTM model is evaluated primarily on accuracy and profit back testing based on model predictions. The experiment that performs the best will be analysed further, including the evaluations outlined in “Testing Strategy #1”. The upcoming subsections present the results from the LSTM experiments, focusing on profit back testing and model accuracy. For a quick overview of the results profit, benchmarked against the buy-and-hold strategy see Table 7.

LSTM Evaluation Metrics Results – Accuracy

In this subsection, the accuracy of Long Short-Term Memory (LSTM) models is evaluated. Training, validation, and simulation accuracy results from five experiments designed to test the LSTMs efficacy are presented. Please refer to Figure 6 for a detailed visualisation of the accuracy metrics across these experiments.

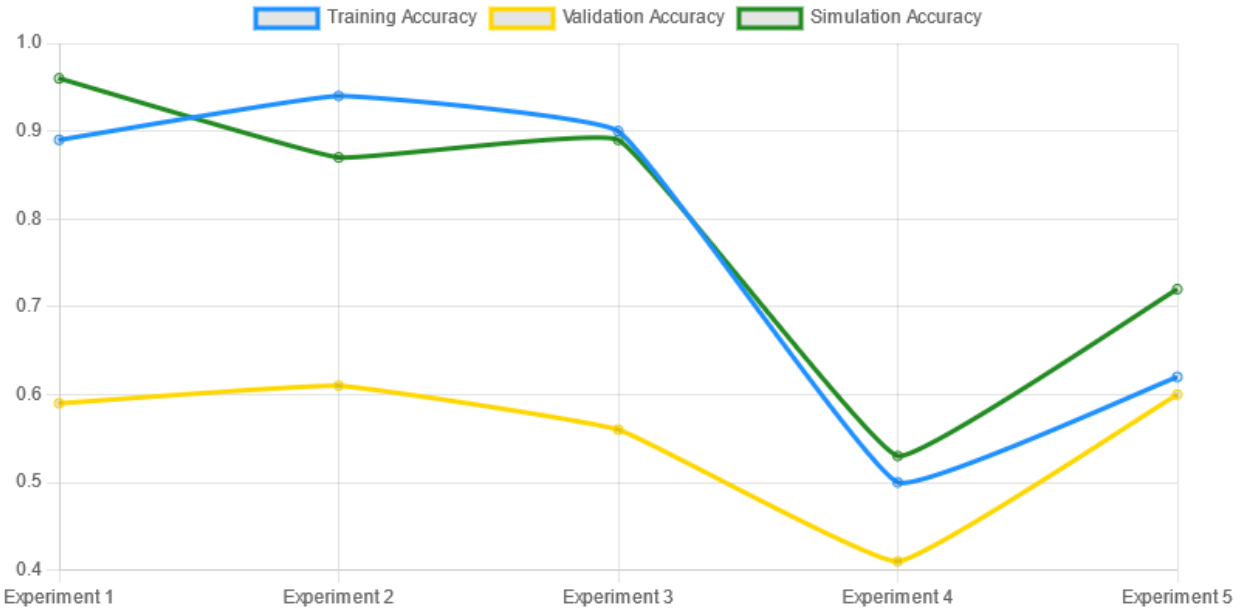


Figure 6: LSTM Accuracy Results

From the accuracy results of the LSTM model, it is clear that different experiments yielded varied performances on the various stocks. Experiment 1 was the highest accuracy in the simulations, while Experiment 4 achieved the lowest overall accuracy results of the five. Experiment 5 had a similarly poor accuracy score compared to Experiments 1, 2 and 3 but was still .2 higher than Experiment 4. Validation accuracy presented especially low results.

LSTM Experiments Profit-Back Results

Table 4 presents the profit resulted from profit trading back-testing based on the LSTM model prediction of each of the five stock experiments. This table illustrates the profit performance for each experiment based on the LSTM model's trading simulation compared to the buy and hold strategy and the percentage change in price from the model's profit to the buy and hold profits.

Table 4: LSTM Profit-Back Results

Experiment ID	Stock ID	Profit-Back (LSTM)	Buy And & Hold	% Change (B&H to Model)
1	AAPL	$5933.01 (+49.4%)	$2535.50 (+21.1%)	57.2645% Increase
2	JPM	$5917.25 (+49.3%)	-$464.65 (-3.9%)	107.852% Increase
3	XOM	$5823.71 (+48.5%)	-$1066.74 (-8.9%)	118.317% Increase
4	KO	-$1682.033 (-14%)	-$68 (-0.6%)	95.9573% Decrease
5	PFE	$3690.05 (+30.8%)	-$784.10 (-6.5%)	121.249% Increase

From the trading simulation profit results for the LSTM model, it is clear that Experiments 1, 2, 3 and 5 vastly outperformed the buy-and-hold strategy. Experiment 4 had the highest percentage change from the buy-and-hold strategy to the LSTM model profits, however the percentage increase from the initial capital to the profit level was only 4th best. Experiment 1 outperformed the others on profit-back closely followed by Experiment 2. Conversely, Experiment 4 experienced the most significant loss and was outperformed by the buy-and-hold strategy.

Discussion of LSTM Experiment Results

From the results of the LSTM model experiments, evaluated on accuracy and profit back-testing (as presented in Figure 6 and Table 4), it is clear that all experiments except Experiment 4 showed good outcomes. Considering the accuracy and profit back-testing outcomes each of the stock experiments the following observations can be made for each experiment:

Experiment 1: This experiment yielded an impressive accuracy in simulations outperforming even the training accuracy which is highly unexpected and which none of the other experiments achieved in kind. The model also recorded a substantial 57.2645% increase in change and a +49.4% profitability increase when trading AAPL stock. This high performance could be attributed to the model's ability to harness a wide variety of market signals, enhancing its forecasting precision. However, a notable gap between high training accuracy and simulation accuracy suggests potential over-fitting, questioning the generalizability of the model when applied to unseen data. Despite this, the ability to generate significant profit shows the LSTMs prowess in stock price prediction solutions.

Experiment 2: This experiment also showcased strong profitability with a +49.3% return on investment. The prolonged validation and testing phases likely contributed to a refined model tuning, offering a more generalised performance across different stocks. This is evident from sizeable profit figures such as a 107.852% increase over the buy-and-hold strategy for JPM stock. This demonstrates that careful

feature selection and data handling can significantly enhance model performance in stock market predictions.

Experiment 3: Similar to Experiments 1 and 2, Experiment 3 continued to demonstrate superior performance over the conventional buy-and-hold strategy, indicating effective learning and prediction capabilities under specific configuration constraints. Most impressive is the fact that even though the buy and hold performance was the lowest with a -8.9% return on investment, the experiment was still able to closely follow the performance of Experiments 1 and 2.

Experiment 4: Experiment 4, contrary to the others, experienced a notable underperformance in profitability, particularly visible in the trading simulation for KO stock, marking a 95.9573% change decrease and -14% return on investment. The result from this experiment could signify a deficiency in the model setup, perhaps due to the exclusion of key features that are critical for predicting this specific stock's price movements effectively. The significant profit drop in this setting suggests that the absence of certain market data can critically impair the model's effectiveness, leading to poor trading outcomes.

Experiment 5: Experiment 5 showed a remarkable recovery from the dip seen in Experiment 4, with profitability significantly exceeding the buy-and-hold strategy, evidenced by a +30.8% increase in investment. Although the accuracy was lower compared to Experiments 1, 2, and 3, the model still successfully capitalised on the features used, indicating a good model specificity to the features that were indeed included. This experiment hints at the potential applicability of LSTM models in scenarios where certain data might be unavailable, yet sufficient data exists to make productive predictions.

Results of DQN Model

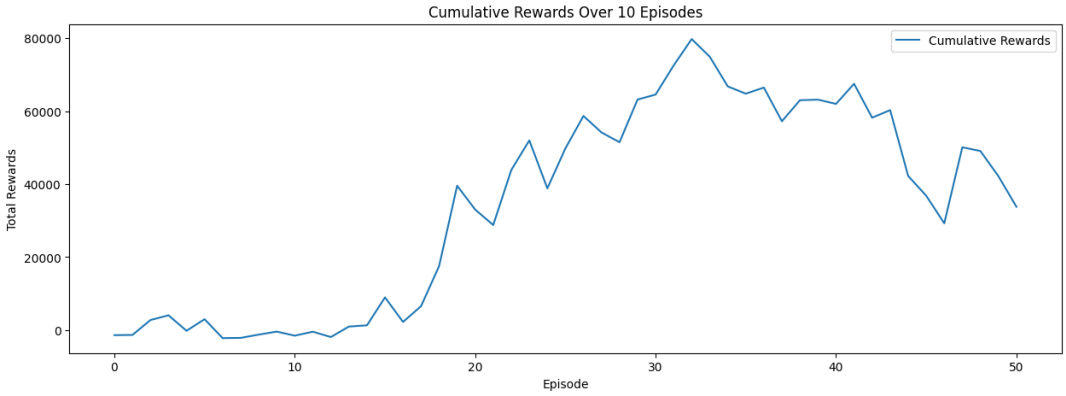
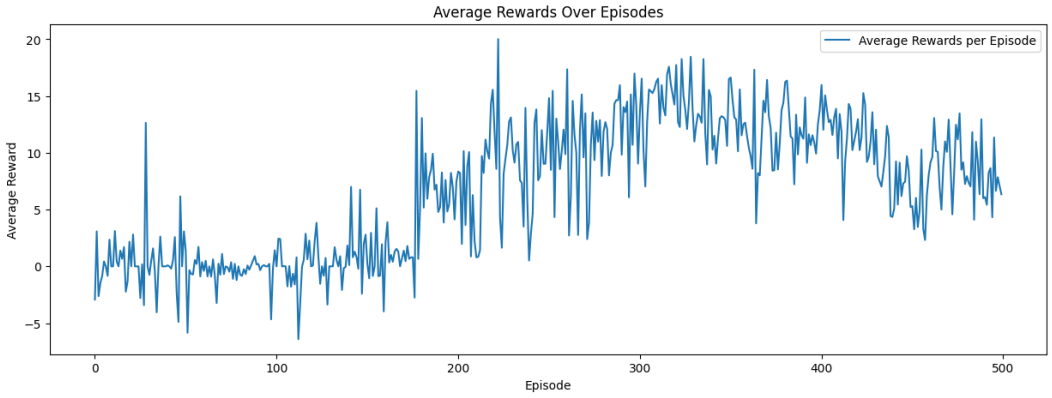
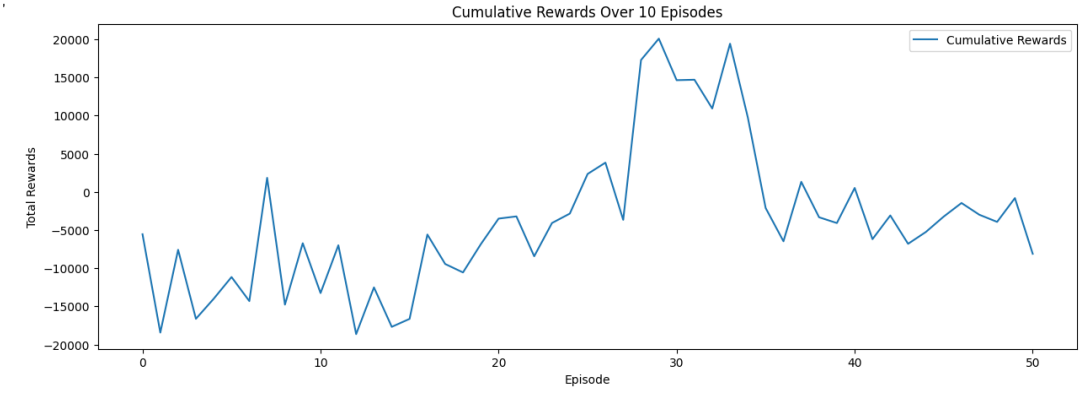
This section transitions from the evaluation of LSTM models to exploring the results of DQN models across five distinct experiments performed on selected stocks (refer to Table 2). Detailed discussions on the DQN model's architecture and data preparation processes are explored in the implementation chapter for further information. In this section, the focal assessment metrics for the DQN experiments will be profit-back testing simulated on the model's trading predictions, with a supplementary evaluation on cumulative and average rewards across the learning episodes. Each experiment benchmarked the DQN's performance against conventional metrics against the buy-and-hold strategy, offering an encompassing view of its decision-making optimization and profitability over time. The forthcoming subsections delineate the DQN's reward metrics outcomes to demonstrate how effectively the model adapts and learns through its reinforced learning protocol over 500 episodes (as detailed in Table 5). Subsequently, the profit-back results (shown in Table 6) will be explored to quantify the practical applicability of the DQN in real trading scenarios, followed by a comprehensive discussion to synthesise these findings.

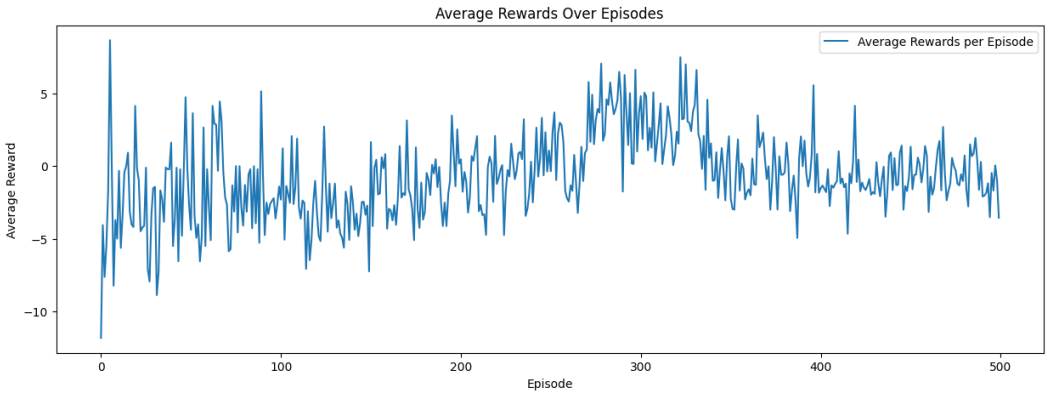
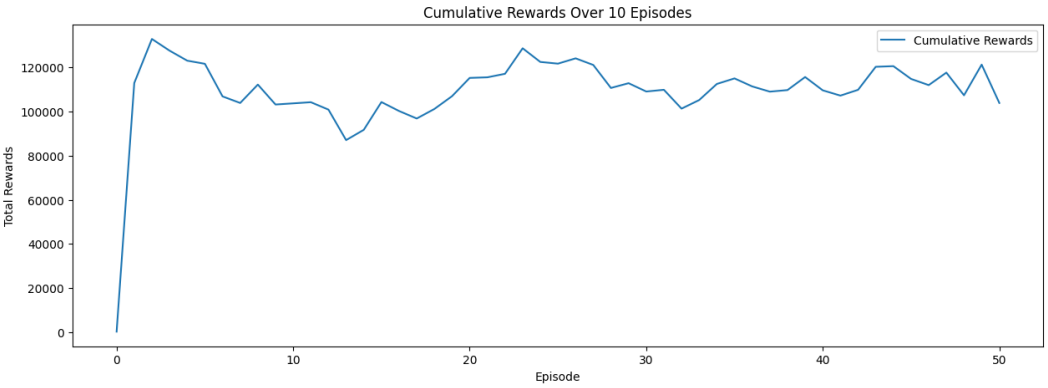
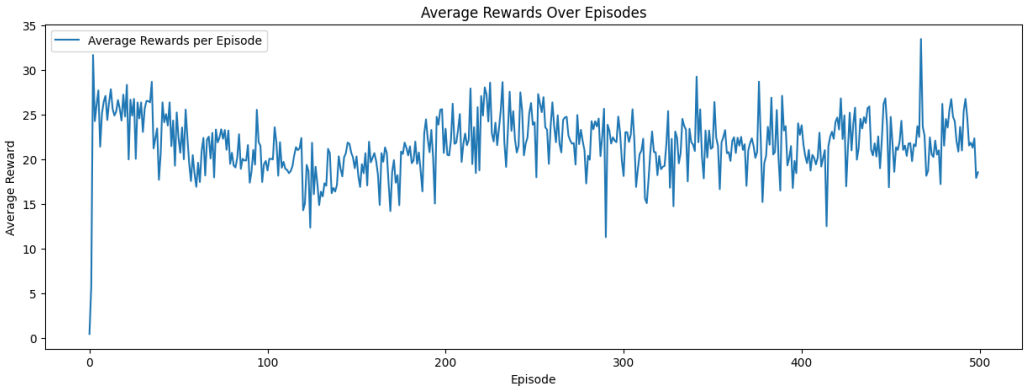
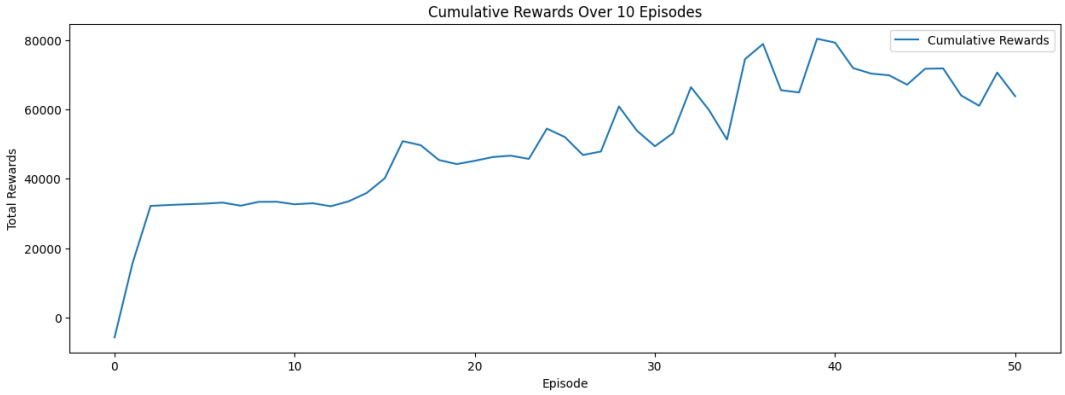
DQN Evaluation Metrics Results – Rewards

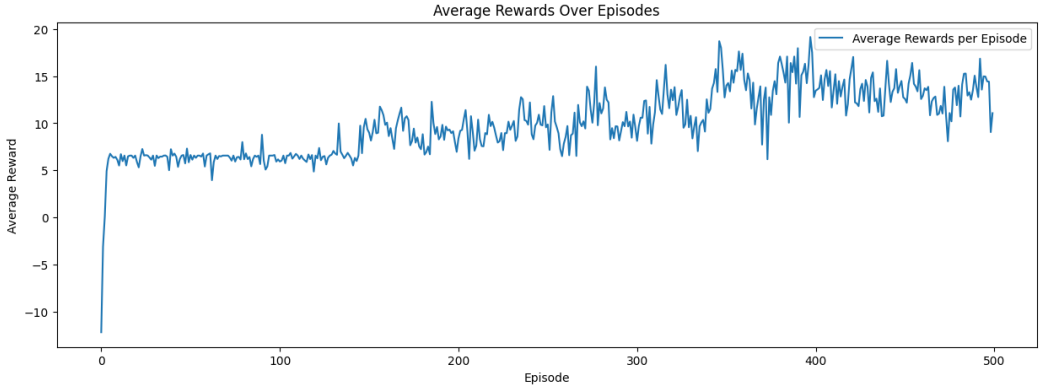
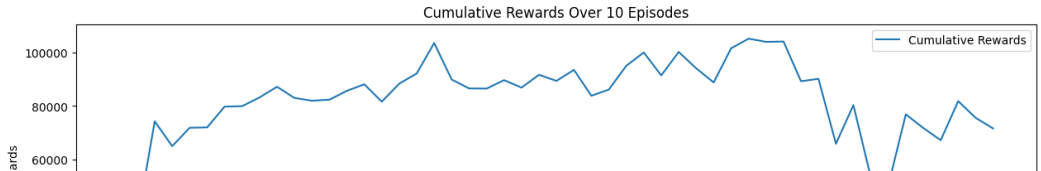
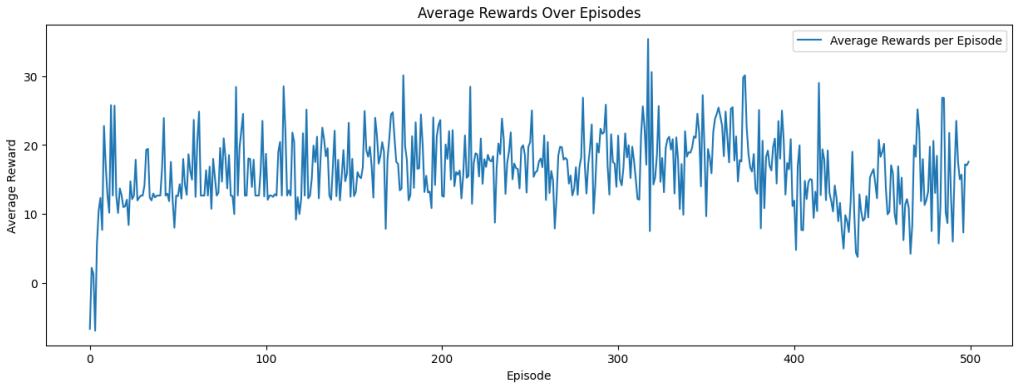
This section highlights the evaluation metrics for the DQN model focusing on two key performance indicators: cumulative rewards and average rewards per episode. The analysis covers the DQN's performance over a span of 500 episodes, with cumulative rewards recorded every 10 episodes. This structured observation interval allows us to discern trends and stability in the learning process of the DQN across training episodes. The average rewards for each episode are also recorded to provide a granular view of the model's learning efficacy on an episodic basis. The results in the table below are intended to shed light on the DQN's capability to optimise decision-making over time.

Table 5: DQN Reward Metrics Results

Experiment ID	Cumulative
	Average

1	 Cumulative Rewards Over 10 Episodes Cumulative Rewards Total Rewards Episode  Average Rewards Over Episodes Average Rewards per Episode Average Reward Episode
	 Cumulative Rewards Over 10 Episodes Cumulative Rewards Total Rewards Episode

2	 Average Rewards Over Episodes Average Rewards per Episode Average Reward Episode
3	 Cumulative Rewards Over 10 Episodes Cumulative Rewards Total Rewards Episode  Average Rewards Over Episodes Average Rewards per Episode Average Reward Episode
	 Cumulative Rewards Over 10 Episodes Cumulative Rewards Total Rewards Episode

4	
5	 

From the reward results of the DQN over its training periods for each experiment and stock, it is evident that the reward levels for each experiment yielded distinctly different results. Specifically, Experiment 3 vastly outperformed the other experiments in both cumulative and average rewards reaching a cumulative peak of 120,000 and averaging around 25. Furthermore, Experiment 3 never had a negative average reward unlike the other experiments and its cumulative reward levels were stable throughout the 500 episodes. Similarly, Experiment 5 rarely had any negative average rewards and had the second most stable cumulative rewards. Conversely, Experiments 1 and 2 had lower cumulative and average rewards overall with Experiment 4's results belonging in the middle of the two resulting groups. Building

on this overview of the DQN model's performance, an analysis of the outcomes from each experiment will appear in a later subsection.

DQN Experiments Profit-Back Results

Table 6 presents the profits resulting from profit trading back-testing based on the DQN model prediction of the five stock experiments. This table, similarly to table 4 in the LSTM results section, illustrates the profit performance for each experiment based on the DQN model's trading simulation compared to the buy and hold strategy. As well as the percentage change in price from the model's profit to the buy and hold profits.

Table 6: DQN Profit-Back Results

Experiment ID	Stock ID	Profit-Back (DQN)	Buy And & Hold	% Change (B&H to Model)
1	AAPL	$1382.55 (+11.5%)	$2535.50 (+21.1%)	83.393% Decrease
2	JPM	-$1190.98 (-9.9%)	-$464.65 (-3.9%)	60.9859% Decrease
3	XOM	$257.33 (+2.1%)	-$1066.74 (-8.9%)	514.542% Increase
4	KO	$463.58 (+3.9%)	-$68 (-0.6%)	114.668% Increase
5	PFE	-$1363.68 (-11.4%)	-$784.10 (-6.5%)	42.5012% Decrease

Discussion of DQN Experiment Results

This analysis elaborates on the performance of the DQN model across five different experiments, evaluated based on cumulative and average rewards, as well as profit-back testing with reference to the traditional buy-and-hold strategy. The DQN experiments were designed to examine its adaptability and effectiveness in optimising stock trading decisions.

Experiment 1: Experiment 1 demonstrated modest performance with an +11.5% increase in profit-back when using the DQN model to trade AAPL stock. However, when compared to the buy-and-hold strategy, there's an 83.393% decrease, indicating that the DQN model was significantly underperforming the passive strategy. This suggests that the model might not have effectively captured or capitalised on the price movement patterns of AAPL within the parameters set for this experiment, perhaps due to limitations in state or reward design which hampered learning efficacy.

Experiment 2: The DQN model in Experiment 2 recorded a -9.9% change in profit when trading JPM, and even more starkly, led to a 60.9859% decrease against the buy-and-hold approach. This underperformance might reflect insufficient or inadequate learning from the trading environment, potentially due to an unoptimized action space or reward function that did not align well with the financial dynamics of JPM stock.

Experiment 3: Contrastingly, Experiment 3 is the standout performer with the model achieving a notable +514.542% increase over the buy-and-hold strategy while handling XOM stocks. This impressive feat, coupled with the most stable and positive cumulative rewards trend (peak of 120,000) and a high average reward of 25, indicates a highly effective model configuration tailored to the specific dynamics of XOM stock. This could imply a well-optimised balance of risk and reward in the model's policy network.

Experiment 4: Experiment 4 provided a moderate success with a +3.9% profit-back and a 114.668% increase over the negative result from the buy-and-hold strategy in trading KO stock. The positive outcomes here suggest that while the model's configuration might not be as fine-tuned or optimal as in Experiment 3, it still managed to leverage opportunities that the passive strategy missed, possibly by capturing short-term advantageous trades that the simple buy-and-hold could not.

Experiment 5: Finally, Experiment 5 reported a -11.4% in profit-back with DQN trading PFE stock, and a 42.5012% decrease compared to the buy-and-hold strategy, marking it as another underperforming setup similar to Experiments 1 and 2. This highlights potential flaws in model suitability or robustness for trading PFE stocks, potentially indicating an inappropriate formulation of the state space or an inadequate response to market volatility.

The variegated results across the five experiments illustrate how critical the correct configuration of the DQN model is, including its reward structure, state representation, and action selection process, to effectively learn and make profitable trading decisions. Experiment 3's superior results underscore the potential of DQNs when optimally configured, while the underperformance in Experiments 1, 2, and 5 signals the need for further refinement in these models or reconsideration of an approach specific to each stock's trading characteristics. These insights point towards the necessity for adaptive, stock-specific strategies when applying DQN models in trading scenarios to fully harness their potential in real-world financial markets.

LSTM and DQN Model Comparison

This section aims to compare the performance of LSTM and DQN, in predicting future stock prices and trading. Given the core objective of this dissertation is to determine whether a DRL algorithm can outperform a recurrent neural network (RNN) like LSTM, a detailed comparative analysis is conducted. Herein, we explore both models' performances across the five selected stock trading experiments through profit-back testing results and leveraging evaluation metrics such as accuracy (for LSTM) and rewards (for DQN).

An integrated view from the profit-back results (Tables 4 and 6) reveals noticeable differences in profitability and performance consistency between the LSTM and DQN models across varying market conditions and stock selections.

Table 7: Combined Profit-Back Results

Experiment ID	Stock ID	Profit-Back (DQN)	Profit-Back (LSTM)	Buy And & Hold
1	AAPL	$1382.55 (+11.5%)	$5933.01 (+49.4%)	$2535.50 (+21.1%)
2	JPM	-$1190.98 (-9.9%)	$5917.25 (+49.3%)	-$464.65 (-3.9%)
3	XOM	$257.33 (+2.1%)	$5823.71 (+48.5%)	-$1066.74 (-8.9%)
4	KO	$463.58 (+3.9%)	-$1682.033 (-14%)	-$68 (-0.6%)
5	PFE	-$1363.68 (-11.4%)	$3690.05 (+30.8%)	-$784.10 (-6.5%)

LSTM exhibits superior profitability in most experiments, notably with Experiment 1 (AAPL: +49.4%), Experiment 2 (JPM: +49.3%), Experiment 3 (XOM: +48.5%), and Experiment 5 (PFE: +30.8%). These

results contrast starkly with those of the DQN, wherein the profits often failed to surpass or even approach those of the LSTM model, aside from its standout performance in Experiment 3 with XOM stock (+2.1% with a massive 514.542% relative increase due to negative buy-and-hold performance).

When examining the models' stability and performance consistency through their training, LSTM tends to achieve a more reliable output across different stocks. Conversely, DQN shows highly variable results, ranging from substantial negative to notable peaks (as seen with XOM in Experiment 3). This inconsistency could suggest an underlying challenge within the reward structure or the adaptation to different market environments.

Although LSTM struggled with some possible over-fitting in some instances (as suggested by poorer simulation versus training accuracy), its overall profitability suggests good generalisation when properly regularised and tuned. On the other hand, DQN's significant fluctuations indicate potential difficulties in model tuning or policy network configuration to consistently interpret and react to market dynamics optimally.

For LSTM, accuracy across experiments provided insights into the model's capacity to generalise and perform under different conditions. Experiment 1, showing high accuracy and profitability, demonstrates the model’s capability in capturing complex patterns and applying them profitably in real trading.

In contrast, the reward system used in evaluating DQN provided a different perspective, focusing on the model's ability to optimise decisions dynamically across episodes. While Experiment 3 showed DQN could achieve remarkable success under specific configurations, the general applicability and consistency across different stocks were less reliable than LSTM.

The overarching synthesis of the comparative study between LSTM and DQN models in stock trading simulations suggests that LSTM, because of its robust generalisation and consistent profitability across diverse stocks and market conditions, holds superior utility over DQN in most trading scenarios. However, the striking success of DQN in certain configured settings (as evidenced in Experiment 3) underscores the potential of DRL algorithms when optimally tailored to specific market dynamics or trading strategies.

This comparative analysis sets the stage for future work on the topic of DRL solutions for stock trading which will be explored in the upcoming section.

Conclusions & Future Work

This dissertation explored the relative competencies of DQN versus LSTM networks in the context of stock price predictions, focusing on their ability to outperform each other and traditional buy-and-hold strategies. The study aimed to find areas of which DRL may surpass RNNs in this field.

Despite its innovative approach, the dissertation found that DQN models, while showing promise in isolated cases such as Experiment 3, generally did not sustain consistent results across all scenarios, unlike their LSTM counterparts which displayed remarkable robustness and profitability. This finding aligns with the core analysis where LSTM models generally provided greater stability and predictability in profit generation compared to the more volatile outcomes observed with DQN models. Mirroring the sentiment found in the literature review, where DRL solutions such as this one often fails to predict future market movements accurately.

Reflecting on the outcomes of this research, there are several key areas this project helped highlight for future exploration. First is the idea for further refinement of the DQN model's action space. This could be highly beneficial, particularly in volatile markets. For example, integrating a minimum change threshold for executing trading actions would help in dampening the impact of market noise and focusing on substantial trends. Enhanced analysis concerning the feature selection that influences model behaviour is also an expansion to highlight. Implementing a more granular approach in feature engineering could potentially mitigate the issues around over-fitting in LSTM and under-fitting in DQN to optimise model performance. As this project used a pure DRL solution, exploring hybrid models that integrate both RNN and DRL elements could provide a more balanced approach, leveraging the predictability of LSTM and the adaptive decision-making capabilities of DQN. The synergy of the two models could potentially elevate the performance metrics beyond the current capabilities of individual models. Lastly, experimenting with incorporating sentiment analysis, particularly from social media platforms and financial news, could provide another layer of context to the models, potentially improving their responsiveness to market sentiment changes.

Reflecting on existing literature as discussed in the related work section, several studies such as those by Kabbani and Duman (2022) and Shi et al. (2021) have shown DRLs capacity to adapt to complex market environments and regulatory differences, showcasing higher efficiency over traditional methods and the ability to outperform regular trading strategies in diverse scenarios. This aligns with the findings of this dissertation, where LSTM displayed robust performance while DRL showed potential in specific experiments. However, echoing concerns from regression-based analyses by Nti, Adekoya, and Weyori (2021), there remains over-dependence on historical data with both models utilised in this study. This is, therefore, still a key area for improvement that future studies could explore further.

In closing, while this project did not conclusively establish DRL as superior to LSTM in stock prediction tasks, it lays a substantial groundwork and opens several avenues for further research. The journey through the intricate dynamics of financial markets continues, promising exciting opportunities for innovation and discovery in the combination of finance and machine intelligence. This exploration not only pushes the frontier of academic research but also offers practical insights and tools for financial analytics and trading strategy development.

Reflection on learning

This dissertation has been a challenging undertaking, and I have learned a great deal about both machine learning and the stock market. When I chose this dissertation topic, I felt I had a good, basic conceptual understanding of machine learning models and the methodology behind them. However, before beginning the dissertation I felt I needed to expand my knowledge in this area and began some background reading about market statistical techniques and machine learning models in greater depth, so that when I began the major work on the dissertation, I would have a greater understanding of the underlying theories, principles, and software and programs I would need to be comfortable working with. However, the background reading I was able to complete felt like it made little contextual sense until I began implementing the DQN machine learning model of my own accord.

Many times, in this process I found myself waiting hours training my models only to realise that I had a simple syntax error or another minor bug in my code. After this happened a few too many times I learned a valuable lesson in the importance of reviewing and testing code before launching into resource-intensive tasks. This was a discipline that I gradually developed over the course of my project. The habit of double, and sometimes triple, checking my work before proceeding became second nature - a reflection of the lessons learned from the many hours lost to simple mistakes. Additionally, this dissertation increased my appreciation for version control due to the worry of making mistakes. The realisation that a single misstep could lead to irreversibly overwriting crucial files cemented my reliance on version control systems. Git became my safety net, ensuring that even in the face of errors, my work was preserved, allowing me to roll back to previous versions when necessary.

Another major takeaway from this project has been the importance of good time management. Working on this project has been a challenge working in a straight-forward timeline. In the beginning, I allotted time for each stage of the process in a Gantt chart for my initial plan. However, as I was working on the project, it developed in such a way that I ended up moving forward several steps in my timeline, seeing if the approach I made was working and then revising my initial approach and stepping back to a previous point in the process. These revisions were very taxing on my mental state during the project, unable to shake feeling like I was making negative progress. However, after the first few revisions I learned to be more flexible to changes in my planned timeline for the project and realised that it was all part of the project's development process, which I would need to accept in order to deliver a respectable final outcome for the dissertation.

Furthering this point, I became unwell for two weeks straight and completed little to no work on the project. I had already completed the programming of my project at this time which made it less impactful of a blow, however it still set me back further than where I planned to be at that point in the process. Pushing back my schedule that much meant that I had to accept that I wouldn't meet the deadlines in my Gantt chart. This worried me a lot at the time as it caused a lot of uncertainty as I was not confident that I would be able to complete the project before the school's internal deadline. However, with encouragement from my family and friends and the support of my supervisor, I kept making steady progress with the project and I became decreasingly worried about making the deadline in time.

Overall, throughout this project I feel I have learned a great deal about both modern computing and the direction that the industry is heading in with machine learning, and furthermore, how the stock market operates and its intricacies of predicting future prices, making me realise why this problem is yet to be solved to a great degree of accuracy. This dissertation and the skills I have developed over the course of these few months have led me to feel confident about tackling new projects and challenges in my future career.

Appendices

Appendix A

CPU Specifications

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 2199.998
cache size      : 56320 KB
```

```
processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 2199.998
cache size      : 56320 KB
```

RAM Specifications

```
svmem(total=13609451520, available=12630577152, percent=7.2, used=584118272, free=8823353344, active=375508992,
inactive=4004192256, buffers=136085504, cached=4065894400, shared=1404928, slab=156725248)
```

GPU Specifications

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   51C    P8              10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

References

Akkaynak, B. (2023) 'A study on the comparison of technical indicators used in stock price prediction with the BAHP method', *JOURNAL OF LIFE ECONOMICS*, 10(1), pp. 1–15. Available at: https://doi.org/10.15637/jlecon.1954.

Aroussi, R. (2024) 'yfinance: Download market data from Yahoo! Finance API'. Available at: https://github.com/ranaroussi/yfinance (Accessed: 5 April 2024).

Baker, M. and Wurgler, J. (2007) 'Investor Sentiment in the Stock Market', *Journal of Economic Perspectives*, 21(2), pp. 129–151. Available at: https://doi.org/10.1257/jep.21.2.129.

CFA Institute (2023) 'Technical Analysis of Stocks & Securities'. Available at: https://www.cfainstitute.org/en/membership/professional-development/refresher-readings/technical-analysis (Accessed: 11 April 2024).

Corporate Finance Institute, Technical Analysis (2020) *Technical Analysis - A Beginner's Guide*, *Corporate Finance Institute*. Available at: https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/technical-analysis/ (Accessed: 11 April 2024).

Daniswara, D.A., Widjanarko, H. and Hikmah, K. (2022) 'THE ACCURACY TEST OF TECHNICAL ANALYSIS OF MOVING AVERAGE, BOLLINGER BANDS, AND RELATIVE STRENGTH INDEX ON STOCK PRICES OF COMPANIES LISTED IN INDEX LQ45', *Indikator: Jurnal Ilmiah Manajemen dan Bisnis*, 6(2), p. 16. Available at: https://doi.org/10.22441/indikator.v6i2.14806.

Droettboom, J.D.H., Michael (2024) 'matplotlib: Python plotting package'. Available at: https://matplotlib.org (Accessed: 5 April 2024).

Fama, E.F. and French, K.R. (1993) 'Common risk factors in the returns on stocks and bonds', *Journal of Financial Economics*, 33(1), pp. 3–56. Available at: https://doi.org/10.1016/0304-405X(93)90023-5.

Fischer, T. and Krauss, C. (2018) 'Deep learning with long short-term memory networks for financial market predictions', *European Journal of Operational Research*, 270(2), pp. 654–669. Available at: https://doi.org/10.1016/j.ejor.2017.11.054.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*, 9(8), pp. 1735–1780. Available at: https://doi.org/10.1162/neco.1997.9.8.1735.

Investopedia (2021a) *Simple Moving Average (SMA): What It Is and the Formula*, *Investopedia*. Available at: https://www.investopedia.com/terms/s/sma.asp (Accessed: 11 April 2024).

Investopedia (2021b) *Stochastic Oscillator: What It Is, How It Works, How To Calculate*, *Investopedia*. Available at: https://www.investopedia.com/terms/s/stochasticoscillator.asp (Accessed: 11 April 2024).

Investopedia (2021c) *What Is a Momentum Indicator? Definition and Common Indicators*, *Investopedia*. Available at: https://www.investopedia.com/investing/momentum-and-relative-strength-index/ (Accessed: 11 April 2024).

Investopedia (2021d) *What is EMA? How to Use Exponential Moving Average With Formula*, *Investopedia*. Available at: https://www.investopedia.com/terms/e/ema.asp (Accessed: 11 April 2024).

Investopedia (2021e) *What Is MACD?*, *Investopedia*. Available at: https://www.investopedia.com/terms/m/macd.asp (Accessed: 11 April 2024).

Investopedia (2024) *Technical Analysis of Stocks and Trends Definition*, *Investopedia*. Available at: https://www.investopedia.com/terms/t/technical-analysis-of-stocks-and-trends.asp (Accessed: 11 April 2024).

Kabbani, T. and Duman, E. (2022) 'Deep Reinforcement Learning Approach for Trading Automation in the Stock Market', *IEEE Access*, 10, pp. 93564–93574. Available at: https://doi.org/10.1109/ACCESS.2022.3203697.

Khalil, K. *et al.* (2019) 'Economic LSTM Approach for Recurrent Neural Networks', *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(11), pp. 1885–1889. Available at: https://doi.org/10.1109/TCSII.2019.2924663.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', *Nature*, 521(7553), pp. 436–444. Available at: https://doi.org/10.1038/nature14539.

Lin, L.-J. (1992) 'Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching', *Machine Learning*, 8(3/4), pp. 293–321. Available at: https://doi.org/10.1023/A:1022628806385.

Mnih, V. *et al.* (2015) 'Human-level control through deep reinforcement learning', *Nature*, 518(7540), pp. 529–533. Available at: https://doi.org/10.1038/nature14236.

Nti, I.K., Adekoya, A.F. and Weyori, B.A. (2021) 'A novel multi-source information-fusion predictive framework based on deep neural networks for accuracy enhancement in stock market prediction', *Journal of Big Data*, 8(1), p. 17. Available at: https://doi.org/10.1186/s40537-020-00400-y.

NumPy (2024) *NumPy documentation — NumPy v1.26 Manual*. Available at: https://numpy.org/doc/stable/ (Accessed: 31 March 2024).

Padial (Bukosabino), D.L. (2023) 'ta: Technical Analysis Library in Python'. Available at: https://github.com/bukosabino/ta (Accessed: 5 April 2024).

Pandas (2024) *pandas - Python Data Analysis Library*. Available at: https://pandas.pydata.org/ (Accessed: 31 March 2024).

Pino, I. (2024) *How much should you be investing? Here's what experts have to say*, *Fortune Recommends*. Available at: https://fortune.com/recommends/investing/how-much-of-your-income-should-go-toward-investing/ (Accessed: 20 April 2024).

Pruitt, G. (2016) *The ultimate algorithmic trading system toolbox + website: using today's technology to help you become a better trader*. Hoboken, New Jersey: Wiley (Wiley trading).

PyTorch (2024) *PyTorch documentation — PyTorch 2.2 documentation*. Available at: https://pytorch.org/docs/stable/index.html (Accessed: 5 April 2024).

Rawat, T. and Khemchandani, V. (2016) 'Feature Engineering (FE) Tools and Techniques for Better Classification Performance', *International Journal of Innovations in Engineering and Technology*, 8(2). Available at: https://doi.org/10.21172/ijiet.82.024.

Sakhare, N. and Sagari, S.S. (2019) 'Performance analysis of regression based machine learning techniques for prediction of stock market movement.', *International Journal of Recent Technology and Engineering*, (7), pp. 655–662.

Shi, Y. *et al.* (2021) 'Stock trading rule discovery with double deep Q-network', *Applied Soft Computing*, 107, p. 107320. Available at: https://doi.org/10.1016/j.asoc.2021.107320.

Shleifer, A. and Vishny, R.W. (1986) 'Large Shareholders and Corporate Control', *Journal of Political Economy*, 94(3), pp. 461–488.

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

TensorFlow (2023) *Train a Deep Q Network with TF-Agents | TensorFlow Agents*, *TensorFlow*. Available at: https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial (Accessed: 18 March 2024).

tqdm (2024) ‘tqdm’.

Watkins, C.J.C.H. and Dayan, P. (1992) ‘Q-learning’, *Machine Learning*, 8(3–4), pp. 279–292. Available at: https://doi.org/10.1007/BF00992698.