DEPARTMENT OF COMPUTER SCIENCE

An Investigation in to the Success of Deep Learning Trading Agents

Matthew Meades

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Sunday 13th September, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Matthew Meades, Sunday 13th September, 2020

Executive Summary

A compulsory section, of at most 1 page

Contents

Supporting Technologies

A compulsory section, of at most 1 page

This section should present a detailed summary, in bullet point form, of any third-party resources (e.g., hardware and software components) used during the project. Use of such resources is always perfectly acceptable: the goal of this section is simply to be clear about how and where they are used, so that a clear assessment of your work can result. The content can focus on the project topic itself (rather, for example, than including "I used LaTeX to prepare my dissertation"); an example is as follows:

- Python2/Python3, Boto3 library, Fabric library
- Amazon Web Services EC2 Instances
- Bristol Stock Exchange (BSE) self contained minimal market simulation
-

Notation and Acronyms

An optional section, of roughly 1 or 2 pages

Any well written document will introduce notation and acronyms before their use, *even if* they are standard in some way: this ensures any reader can understand the resulting self-contained content.
Said introduction can exist within the dissertation itself, wherever that is appropriate. For an acronym, this is typically achieved at the first point of use via "Advanced Encryption Standard (AES)" or similar, noting the capitalisation of relevant letters. However, it can be useful to include an additional, dedicated list at the start of the dissertation; the advantage of doing so is that you cannot mistakenly use an acronym before defining it. A limited example is as follows:

AES	:	Advanced Encryption Standard
DES	:	Data Encryption Standard
	⋮	
$\mathcal{H}(x)$	:	the Hamming weight of x
$\mathbb{F}_q$	:	a finite field with q elements
x_i	:	the i-th bit of some binary sequence x, st. $x_i \in \{0, 1\}$

Acknowledgements

An optional section, of at most 1 **page**

It is common practice (although totally optional) to acknowledge any third-party advice, contribution or influence you have found useful during your work. Examples include support from friends or family, the input of your Supervisor and/or Advisor, external organisations or persons who have supplied resources of some kind (e.g., funding, advice or time), and so on.

Chapter 1

Introduction

Over the last few decades, trading in financial markets has been transformed by an increase in both computational power and availability. A significant result of this is the global rise of algorithmic trading; computers following defined sets of instructions which determine whether or not to execute a trade. Many papers have been published investigating different trading algorithms (e.g. [10][15][14]), with significant results in 2001/2011 showing that some of these trading agents could even outperform amateur human traders [23][12] [21]. As a result, trading firms began adopting algorithmic traders, with 2011 figures estimating that they were responsible for around 50% of global equity trading volume [11][22], and over 80% of Forex transactions in 2016 [3].

Algorithmic traders are evidently useful but human traders are still employed by companies, with much of their trading success attributed to their intuition, experience, and instinct. Recently, efforts have been made to try and emulate this intuition using deep learning. Le Calvez (2018) successfully trained a Deep Learning Neural Network (DLNN) purely through observation of an algorithmic trader, which went on to outperform the original trader [5]. This provided an impressive proof of concept that if an algorithm can be observed and its behaviour learned, then the same could be done with a human trader. An extension of this work by Wray (2020) used data across the entire Limit Order Book (LOB), collected from multiple traders in tens of thousands of market sessions, to train his own deep learning trader called DeepTrader [32]. DeepTrader was able to outperform, or match the performance of, every other trader in one in many tests, and all but one trader in balanced group tests.

Despite the success of DeepTrader its predecessors, there has been no in depth analysis aimed at identifying the reasons why these traders are successful. Neural networks are very abstract tools for modelling complex relationships, and so it is not directly obvious what information they are using to make their trades. However, it is important to analyse these decision making processes - there may be features being input to DeepTrader which are of no use to the model, or which actually hinder its performance and are detrimental to the quality of its predictions. Discovering which features are useful for price prediction could also allow DeepTrader to be optimised, leading the way in the development of future deep learning traders.

The aim of this project is to begin an investigation in to the underlying mechanisms which contribute to the success of these traders, first by replicating the work completed by Wray earlier this year, then by varying the input features both at the training and trading stages to determine their impact on trader performance. This will identify the most important features needed for trader success alongside the redundant ones, allowing DeepTrader to be optimised and laying the foundations for future trader development.

1.1 Motivation

In most industry applications, in order for people to trust machine learning they need a reasonable explanation behind the predictions it makes. For example, when a business is making high risk trades with a customer's investment using algorithmic traders, it is reassuring when the business is able to justify which current market conditions mean that this is a good trade to make. However, if the trader

being used is a DLNN such as DeepTrader, this isn't as easy a task. Because deep learning models are an abstract, complicated structure of many interconnected nodes, it is not possible to simply look inside at the algorithms used to see which market variable is influencing the current prediction. This is the first motivation behind this work because as machine learning becomes more commonplace, it is important to try and understand why they work the way they do.

Discovering which market features are influencing the trading price also has multiple other benefits. Previous work by Le Calvez used only data available at the top of the LOB [5], while DeepTrader uses features across the entire depth of the LOB, so it will be valuable to understand whether looking at that depth is actually necessary. Identifying the important features can also help to streamline the DeepTrader model, removing any unnecessary features to improve training time and reduce the amount of data which needs to be calculated and stored. If data deeper down in the LOB is actually redundant, this can be beneficial for trading companies because often, this information isn't readily available and costs a lot to acquire; using only data from the top of the LOB would save a considerable amount of money.

It is also a possibility that some of the data which is being input to the neural network is damaging the quality of its predictions. A model is only as good as the data it is trained on, so if some features are irrelevant and complicate the method that the model is using to derive the correlations between the inputs and the output, then they could damage its predictive capability. Research in to the information content of a limit-order book has shown that only 22% of the information used for price prediction is contained past the best ask and best bid [6], with most orders submitted to the LOB actually ending in cancellations [17]. Inputting this information in to a neural network could result in poor inferences being made by trying to accommodate all the data, and so in the process of trying to use all the information available it creates larger errors in the model's predictions. Therefore, identifying and removing these features would be beneficial for trader performance.

Finally, when training neural networks an important pre-processing stage is to remove redundant features, which means when two or more independent features are highly correlated. Including redundant values is unnecessary and provides no benefit when training the model, only an increase in training times and potentially even making the model worse [4]. Therefore, including redundant data can even be as bad as irrelevant data, and is another reason why it is important to understand feature importance.

Mention other motivations to optimise deeptrader using different network architectures and data pre-processing techniques?

1.2 Project Approach

This project is split in to two main aims, each with its own execution and analysis:

- Replicating the work reported by Wray in 2020 [32] and producing my own version of DeepTrader with comparable performance, called DeepTrader2.
- Performing feature importance analysis on DeepTrader2, then running experiments in Bristol Stock Exchange (BSE) with variations of DeepTrader2 which only use a subset of the original features. This will serve to isolate the essential features and allow the design of an optimised version of DeepTrader2, with more of an understanding behind its decision making process.

The replication of DeepTrader follows the same process outlined by Wray in his original implementation, however there are several avenues which can be explored which could serve to, in theory, improve the performance of DeepTrader. While it is not an official aim to improve the performance of the original DeepTrader, these experiments will be carried out and recorded to determine if DeepTrader can be improved upon. For example, the original DeepTrader uses all the trades from every trader as training data, but isolating and training on only the trades from the best traders could lead to better performance. The same evaluation techniques to Wray will also be used to quantify and compare DeepTrader2 to the original, enabling the first aim to be achieved.

The feature importance analysis consists of using feature perturbation to determine which of the 13 input metrics have the most effect on the model's predictions. The features with the highest importance are

isolated and used to create a 'best' trader, whilst the features with the lowest importance will be used to create a 'worst' trader. These can then be evaluated in experiments in BSE to confirm the initial importance findings. Then, the 'best' trader can have features removed one at a time to create a minimal implementation of DeepTrader2, but with comparable performance.

1.2.1 Aims and Objectives

The first aim, outlined above, can be divided in to the following objectives:

1. Reconfigure BSE and automate data generation to produce the data required to train DeepTrader.
2. Create and train a neural network with an architecture similar to Wray 2020, then re-implement this back in to BSE to trade against the other automated traders.
    - Research and experiment with training parameters until an optimal configuration is achieved.
3. Run experiments to establish the performance of DeepTrader2 relative to the original.
4. Repeat steps 1 - 3, with modifications at any step, until DeepTrader2 has a comparable performance to DeepTrader.

The second aim can be divided in to the following objectives:

1. Perform feature analysis methods on the Neural Network to determine which market metrics contribute the most to training/test loss, and which are redundant.
2. Create variations of DeepTrader2 using the most important and the least important features and reimplement back in to BSE.
3. Run experiments across multiple market conditions and configurations to determine whether removing the redundant features has a statistically significant effect, verifying that the removed features are irrelevant.

1.2.2 Project Deliverables

Completion of the above objectives, and successful experimentation, will lead to the following deliverables:

- Write this list once I'm done at the end

Chapter 2

Contextual Background

This project presumes a basic understanding of the world of trading and real-world financial markets, with a market defined as a medium through which buyers and sellers can interact and exchange assets. Over half of the world's financial markets, including Hong Kong, NYSE, Tokyo, and LSE [13], are based on a Limit Order Book system, which in essence is a ledger containing all of the current buy and sell (bid and ask) limit orders submitted by all traders within the market.

On these LOB-based markets, a lot of information is available to traders which can allow them to determine an optimal price at which to submit an order. Details of the information that can be extracted is discussed in more detail in section 3.1, but first it is important to understand the various automated trading strategies which have been developed in previous publications to trade within these markets.

2.1 A Brief History of Algorithmic Trading

For this project, an understanding of the recent developments regarding trading algorithms is required as this will provide context for the experiments carried out. The trading strategies discussed here are the ones which are used to generate the market data used to replicate DeepTrader, and are also the ones it is compared against to evaluate DeepTrader's success. It is important to recognise the significance of outperforming each trading strategy so that the impact of this work can be fully appreciated.

One of the most commonly cited publications within automated trading is the 1993 paper by Gode and Sunder [15] in which they developed 'zero-intelligence' (ZI) trading agents; one of which was constrained by a maximum/minimum price for bids/offers (ZI-C), and the other which was unconstrained (ZI-U). The results of this study showed that while the unconstrained agents traded randomly with no underlying trends, the constrained ZI-C agents converged to the equilibrium price for that market in the same way that human traders did. This proved their hypothesis that a market's allocative efficiency (a measure of the proportion of total available profit that is successfully extracted from the market by traders [15]) was a consequence of the market structure, not trader intelligence, or so they thought.

In contradiction of this result, Cliff showed in 1997 that Gode and Sunder's conclusion was an artefact of their experiments design, and that if the supply/demand profiles were altered, the ZI-C traders no longer converged to the theoretical equilibrium [10]. To demonstrate this, Cliff produced his own 'zero-intelligence-plus' (ZIP) traders which quoted prices based on a profit margin that could change depending on market variations. In the same year, Gjerstad and Dickhaut developed an algorithm based on an adaptive belief function used to maximise the agent's gain [14], with a small modification of this work by Tesauro and Das resulting in an even more successful Modified GD (MGD) trading algorithm [28].

Subsequent market simulations showed that these MGD trading agents could outperform all other current algorithmic traders (ZIP, ZI-C, Kaplan, GD), with ZIP coming in second. More impressively in 2001, Das et al. were able to show that both ZIP and MGD could outperform human traders, in the first study in which humans and trading agents were able to interact in the same market [23].

Following these results, another extension of GD was published in 2002 by Tesauro and Bredin called

GDX [27], which again consistently traded more profitably than both GD and ZIP traders. Finally in 2006 Vytelingum suggested a trading strategy called the "Adaptive Aggressive" (AA) strategy, which showed strategic dominance in both agent-to-agent and agent-to-human experiments, proving it to be the best current trading agent [31][12]. AA held its title for over 10 years, however recent studies have begun to doubt its versatility, suggesting that it can be beaten by simpler strategies in altered market conditions, including more realistic conditions. For example, a 2015 MSc Thesis by Vach [30] showed that overall, GDX could outperform AA when the market composition (i.e. how many of each trading algorithm is present) is modified, and another study using models constrained with realistic reaction times showed that when speed is considered, AA can be outperformed by the basic Shaver strategy [18]. Further studies completed by Snashall and Cliff in 2019 showed, by exhaustive testing of millions of market simulations, that AA in fact does not dominate when more realistic market conditions are used, being outperformed by more basic strategies. [26][9]

Given this review of the literature revealing that GDX can in fact outperform AA, and the study published in 2011 showing that GDX could outperform human traders [21], it could be assumed that GDX is currently the best performing automated trading strategy. However, since GDX has not undergone an exhaustive testing on the scale that was performed on AA this cannot be verified, and so for the purposes of this research the dominant strategies to aim to outperform are both AA and GDX.

2.2 Deep Learning in Finance

Deep learning concerns itself with training models to learn information using mechanisms inspired by findings in neuroscience. The task of time series forecasting for stock prices has become a popular application for this technology. As an example, stock market analysts will look for patterns in stock price data to try and predict how the price is likely to change in the near future, relying on their years of expertise. The machine learning approach is to use years of stock price data to 'train' a network of interconnected neurons, which are able to modify their connections so that a certain input, or sequence of inputs (say the last 60 seconds of price data), will result in a certain output (whether or not the stock price is about to rise or fall). Training the network to discover these underlying relationships between current and future behaviour is essentially an attempt to replicate the stock market analyst's years of experience.

This collection of interconnected artificial neurons is called a Neural Network (NN), and if the network has enough layers of neurons to be a considerably complex system, it becomes a Deep Learning Neural Network (DLNN) [19]. Similarly to algorithmic trading, deep learning gained popularity as computational power became more affordable, with recent noteworthy achievements by companies like DeepMind [25]. More technical background on deep learning can be found in section 3.3, but this section first aims to convey recent advancements in applying these techniques to the world of automated trading.

While deep learning is notably popular in the field of financial market prediction, with a review of deep learning applied to stock price prediction finding 115 relevant papers published in the last 3 years (2017-2019) [20], it is important to not confuse this application of deep learning with deep learning applied to automated trading. As noted by Le Calvez in 2018 [5], financial time series forecasting isn't applicable to a 'live' trading environment because instead of predicting future transaction price trends based on previous trades, a DLNN trader is trying to reproduce the behaviour of an existing trader in response to the current state of the LOB. Therefore, when considering previous literature in this field only publications focused on developing a trading agent using deep learning approaches are relevant, of which there are considerably fewer examples.

2.2.1 Deep Learning Applied to Automated Trading

Fit in work by Niño [2] in which the price variation was predicted for the next minute of activity using a Deep Neural Network (DNN) trained on stock market data. This price prediction then set the limits for an algorithmic trading agent, allowing it to trade within those limits to ensure an overall profit. This is essentially a combination of price forecasting and algorithmic trading, with the price prediction setting a limit for the algorithmic trader to trade within, instead of directly predicting the price of the next trade. The first relevant publication for this area of research is the work conducted by Le Calvez in 2018 [5],

in which he built upon a preliminary exploration of applying DLNNs to replicate ZIP traders in a 2017 Master's thesis by Tibrewal [29]. Le Calvez built upon this work and progressed it by reimplementing his DLNN trader back into BSE, allowing it to trade 'live' against other algorithmic trading strategies, providing a proof of concept that a DLNN could be trained to trade by simply observing and learning from a single profitable ZIP trader. This was impressive because not only could the DLNN replicate the trading strategy's performance, when competed against each other the DLNN trader actually outperformed the original ZIP trader, implying that it had become more perceptive than the ZIP algorithm and could interpret market data in a way that ZIP could not. Le Calvez subsequently argued that because the DLNN could learn purely by observation, then it is entirely plausible that it could also learn to replicate human traders, and possibly outperform them.

Although an impressive proof of concept, Calvez's approach was limited by the fact that the DLNN trader only considered the current best bid/ask prices when deciding whether to trade, disregarding other market metrics contained in the LOB. These factors can have a significant effect on the market price, and therefore would be valuable information to a trader making the decision of whether to buy/sell/hold its assets, and what price it should buy/sell at. This work was also limited by the fact that only the data from a single trading strategy was used to train the neural network. Different strategies can have certain advantages in different situations, so capitalising on this could have the potential to create an even better performing trader.

These limitations were addressed in the recent work carried out by Wray, 2020 [32], in which a DLNN was trained using millions of trades collected from 7 different trading strategies across over 39,000 market sessions, using information spanning the full depth of the LOB. The trader created using this approach was named DeepTrader, and when reimplemented in to BSE it outperformed all other trading strategies in one-in-many tests, and all but AA in balanced group tests. DeepTrader's performance demonstrated an impressive next step from Le Calvez's work, reinforcing the idea that deep learning could provide a new approach to develop algorithmic trading strategies, but to progress it must first be understood how these traders have performed so successfully.

While the results produced by Wray's work are impressive, they also introduced several questions and avenues for further exploration. The first of these is the fact that DeepTrader trades very profitably despite using trades from all traders regardless of profitability, meaning the average quality of the data it's trained on being relatively low. A neural network can only perform as well as the data it's trained on, and therefore it can be assumed that if the quality of the data improves, the network could perform even better. Secondly, DeepTrader was only evaluated under certain market conditions, and was by no means exhaustively tested, and so its performance under different conditions could be entirely different. This is likely, because of its high variation even in favourable market conditions, and so this is another interesting avenue to pursue.

Arguably, the most interesting route is to investigate exactly why DeepTrader performs so well, and which features within the market it is using to predict its trade prices. Neural networks are famed for their ability to model complex relationships, but to achieve this they also possess a very high level of abstraction. This means that the decision making process they follow is difficult to interpret, and therefore the underlying mechanisms contributing to DeepTrader's success, for the most part, aren't defined. However, there are methods of analysing neural networks to try and understand how they work, and this is the main focus of this research.

2.3 Analysis of Neural Networks

The abstraction of neural networks appears to make it impossible to understand exactly what's going on, often being referred to as "black boxes", but there are ways of modifying the input and observing the effect on the output to try and infer what has happened in between. However, determining feature importance can be more complicated than it first seems, with a paper showing that small, imperceptible feature perturbations to an image input can dramatically change which network features contribute to the image's classification [1]. This application is focused around image classification problems, although it indicates that feature importance may change drastically even with small changes to the input, and so is something to keep in mind during this analysis.

None of the papers mentioned above ([2] [5] [32]) go on to explore the reasons why their work was so successful and what features contributed to their trader's price predictions. However, a publication focused on deep learning price prediction using LOB data by Zhang et al. investigated their model sensitivity by individually perturbing each input, then observing the effect on the model's predictions [33]. This allowed a basic image to be constructed of which inputs were the most important, and therefore gave an insight in to how the model was using the LOB data to make predictions. Because the paper focuses on uses LOB data to generate price predictions, and the inputs to the network are different, the results are not transferable to this work, although the method used to generate those results is.

Feature perturbation involves modifying the input for each feature one at a time, then observing the effect this has on the model's predictions; if there is a considerable decrease in the quality of the model's predictions, that feature has a high importance. More complex feature importance analysis methods exist, but while feature perturbation is an important first step, the analysis of the network doesn't always correlate to analysis of the subsequent trader. Therefore, in this work feature perturbation will be used for an initial analysis, but once there is an indication of which features are important multiple traders can be created for experimentation to determine whether these features really are the ones which contribute to trading ability.

Chapter 3

Technical Background

3.1 Financial Markets

This section aims to provide an idea of the information contained within the financial markets we are using, and therefore the information that is available to the traders within these markets. As mentioned before, this research presumes an initial understanding of financial markets and basic economics, but will cover the more advanced metrics used to quantify the market by the various automated traders.

3.1.1 The Limit Order Book

Most of the world's financial exchanges are based on a Limit Order Book system, which as described earlier is a record of all the current limit orders placed by traders to buy and sell units of an item. Limit orders are orders containing the quantity, type (bid/ask), and the limit price, which for buyers is the maximum price they can buy, and for sellers the minimum price they can sell. Traders receive these limit orders from their customers, and proceed to try and trade as optimally as possible to make a profit while fulfilling their customer orders. A visual representation of a LOB can be seen in Figure 3.1, which shows a clear division between the bid (buying) side, and the ask (selling) side of the book.

Figure 3.1: Visualisation of a Limit Order Book, with best bid/ask as the top entries in the table (Image acquired from [8])

The top of each side of the LOB records the current highest (best) bid, and the current lowest (best) ask, alongside the quantity, with the rest of the orders shown below getting worse further down the book. The yellow box above the orders shows the most recent limit order submitted by a trader, which in this case hasn't appeared on the bid side of the book because the price was actually higher than one of the previous asks. Instead there was a transaction, which can be seen as an entry on the green 'Tape' at the bottom of the figure, detailing the quantity, price, and time at which the trade took place. Traders make money through this process by ensuring that, if they are buying, the transaction price of a trade is lower than the limit price, meaning if they buy a unit for less than the customer order specifies they get to keep the difference as profit. For a seller, they have to make sure they sell for higher than their limit price specifies.

In 'level 1' market data, the only information available is the price and quantity of the best bid and best ask, plus the price and quantity of the most recent transaction (shown in red on the tape). In the work mentioned earlier by Le Calvez [5], only level 1 market data was used, whereas the work which followed by Wray [32] used the entire LOB, also referred to as 'level 2' data. Level 1 data is still useful to calculate basic metrics to describe the market such as the spread, which is the difference between the best ask and best bid (3.1), and the midprice, which is the midpoint of the spread and acts as a basic estimation of the market equilibrium (3.2). Market equilibrium is defined as the point at which supply and demand are balanced, and is represented by the point at which the supply and demand curves meet on the graph shown in Figure 3.1. Knowing the market equilibrium is desirable because this is the point at which all traders (buyers and sellers) can obtain the maximum amount of value from the market.

$$Spread = p_a - p_b \tag{3.1}$$

$$Midprice = \frac{p_a + p_b}{2} \tag{3.2}$$

A more useful variable to predict market equilibrium is the microprice, which provides a closer approximation by considering the differences in supply and demand. In situations where a large quantity is available on one side of the LOB, for example if the supply is much greater than the demand, then this means and the equilibrium price should decrease, and will actually be lower than the midprice indicates. If there is a surplus of bids then it can be assumed that the equilibrium price is actually higher than the midprice because more people want to buy, i.e. there is increased demand. Equation 3.3 below shows that the microprice takes the best price on the ask side of the book (p_a) and multiplies it by the quantity of orders at the best bid price (q_b), then sums it with the best bid (p_b) multiplied by the quantity of best ask orders (q_a), providing a measure of how skewed either way the equilibrium is.

$$Microprice = \frac{(p_a \times q_b) + (p_b \times q_a)}{q_a + q_b} \tag{3.3}$$

If level 2 data is used and the full LOB is taken in to account, more comprehensive metrics can be calculated which describe a multitude of market features, which is why this approach was taken by Wray. Details of these variables are explained in the later section which focuses on the metrics used to train DeepTrader (3.4.1).

3.1.2 Bristol Stock Exchange

The Bristol Stock Exchange (BSE) is an open source tool used to model real-world financial markets [7], creating a minimal simulation of an exchange containing a LOB which can record the trading of a single type of unnamed asset at a time [8]. Several assumptions and simplifications have been made which differentiate BSE from real-world exchanges, such as each trader only being able to submit one order at a time, with the original order being deleted if a second order is placed by the same trader. There is also zero latency between all traders, meaning that any changes to the market can be acted upon by any trader instantaneously. The only order type which traders are allowed to submit in BSE is a limit order, whereas real financial exchanges can allow many different order types which can enable various different trading strategies.

The advantages of BSE are that it allows the user to populate the market with any combination of trading agents, manipulate the market dynamics (supply/demand) to produce a wide range of experiments, collect all the data required for analysis, and run these experiments essentially cost-free. BSE has shown to be a valuable resource, used in years of teaching and several previous publications (e.g. [29] [5] [9] [26] [32]), and therefore it is unlikely that the code used contains any errors. A significant amount of teaching material and documentation is also available detailing how BSE functions, and so modifying and running experiments for this project was both straightforward and repeatable.

3.2 Existing Automated Trading Agents

As mentioned previously in section 2.1, multiple different automated trading strategies have been developed over the last 30 years. Because trader performance must be measured relative to the other traders

in the market, understanding the strategies behind these traders provides important context to be able to appreciate the discussion of results later. Also, because Wray's DeepTrader was compared to these traders [32] my replication of Wray's work will also need to be compared to these traders. In further investigations I will also be evaluating my traders in comparison to these established trading strategies, and so at the very least a brief overview of these is required.

3.2.1 Basic Strategies

Giveaway

Firstly, the most basic strategy included in these experiments is called 'Giveaway', so named because it simply attempts to trade at the limit price specified on the randomly allocated customer order. Because of how BSE was created it is impossible for any traders to trade at a loss, but Giveaway isn't likely to make much money either because profit comes from the difference between the customer order price and the price at which the trade executes. If the strategy is aiming to simply trade at this price they aren't likely to encounter many profitable situations.

ZIC

The second strategy, developed by Gode and Sunder in 1993, is the Zero-Intelligence Constrained (ZIC) trader [15]. As the name suggests this trader also isn't intelligent and has a very simple trading strategy, but it does have the potential to make money. The method employed by ZIC is to simply generate random prices at which to submit quotes, but these prices are constrained by the customer's limit order so that it can never trade at a loss. While BSE doesn't allow this anyway, this was the main difference between the ZI-U and ZI-C traders developed in 1993 which caused ZIC to actually trade profitably.

Shaver

The next trading agent, called Shaver, is the first to actually use market information to calculate a quote price. The aim of Shaver is to take the existing best ask/bid and *shave* a penny off, beating the existing best ask/bid and ensuring that any if a trade does occur, it can make the highest profit possible at that time. Again, this trading agent is also constrained by its own customer order price, so will never drop below (for sellers) or above (for buyers).

Sniper

The final basic strategy, called Sniper, extends the Shaver strategy and is a variation inspired by Kaplan's Sniper which won an automated trading competition in 1990 [24].In BSE, Sniper continuously shaves an amount off its quote price, however the amount it shaves starts off very small and it is very unlikely that it will make a trade. Then, as the market nears close, it increases the amount shaved off the quote to ensure that Sniper can execute a trade before the market session ends. The reality of this is that a market populated solely by Sniper traders will sit stagnant until the end of the market session, at which point they will all drastically reduce their ask prices/raise their bid prices to trade. However, if a market contains other strategies, this tactic can often be advantageous.

3.2.2 ZIP

The Zero-Intelligence Plus (ZIP) trader, developed by Cliff in 1996 [10], is the first trader mentioned so far to update its strategy whenever there is a change in the LOB, not just when it is selected by BSE to submit an order. The information gathered from the LOB is used to update a buying/selling margin which is subtracted/added to the trader's customer order, resulting in a price at which to submit a quote. How large this margin is depends on the market, specifically the current best ask and best bid. These calculations maximise the profit ZIP can gain while also maximising the chance that the quote will actually be accepted by another trader.

3.2.3 GDX

At around the same time as ZIP, the GD trading algorithm, named for its inventors Gjerstad and Dickhaut [14], was formulated based around the idea that the market could provide an indication of how likely it was that an offer would be accepted. The work they carried out concluded that this was possible, and it resulted in a successful trader with subsequent work showing that it could outperform human traders

[28]. The GD trader essentially collects data from every trade and order submitted to the LOB so far, using this to construct its belief functions $p(a)$ and $q(b)$ for sellers and buyers respectively (shown below in equations 3.4 and 3.5).

$$p(a) = \frac{TAG(a) + BG(a)}{TAG(a) + BG(a) + RAL(a)} \tag{3.4}$$

$$q(b) = \frac{TBL(b) + AL(b)}{TBL(b) + AL(b) + RBG(b)} \tag{3.5}$$

In these equations, $p(a)$ and $q(b)$ both represent the estimated probability that a quote at price a/b will be accepted. Using equation 3.4 as an example, $TAG(a)$ corresponds to the amount of asks which have been accepted so far with a price $\geq a$, $BG(a)$ is the number of bids submitted to the LOB with price $\geq a$, and $RAL(a)$ is the number of asks which were not accepted which have a price $\leq a$. Equation 3.5 details the same method for calculating this probability, except with the ask and bid metrics reversed. These belief functions can then be used to select a price a/b which maximises the agent's 'surplus', defined as $p(a) \times (a - limitprice)$ for sellers and $q(b) \times (limitprice - b)$ for buyers.

This algorithm was then extended by Tesauro and Bredin in 2002 to create the Gjerstad-Dickhaut eXtended (GDX) trader [27]. The modifications made to the strategy involve using dynamic programming to maximise profit across the entire market session, instead of just for individual trades. It does this by weighing up the profit of trading the asset immediately and the potential profit gained by trading at a future time, leading to an overall more successful strategy.

3.2.4 Adaptive Aggressive (AA)

When the AA trading agent was developed by Vytelingum in 2006, it was regarded as the dominant algorithmic trading strategy [31], although recently several studies have begun to doubt this claim [30] [18] [9] [26]. Nonetheless, AA has still proved to be a successful strategy in many situations, including against human traders [12], and this is likely due to its ability to adapt aggressiveness when trading. Aggressiveness here refers to a trader's inclination to sacrifice profit in an attempt to increase the likelihood that an offer will be accepted. This is tuned within each agent using a short and a long term learning mechanism, both of which are updated after any change to the LOB to improve the agent's trading strategy.

Similarly to how ZIP calculates its profit margin, for short-term learning AA uses the Widrow-Hoff learning algorithm to tune its aggressiveness factor $r(t)$. The equation isn't included here, for more details see [31], but in essence the short term learning algorithm tunes the trader's aggressiveness relative to the market to either improve the likelihood of a trade (by increasing aggressiveness) or increase potential profit (by decreasing aggressiveness). The long-term learning algorithm, shown below in equation 3.6, updates the parameter θ based on the market volatility, an estimation of which is calculated using Vernon Smith's α in equation 3.7. If the market is considered volatile, it is best for the trading agent to react more aggressively, whereas a less aggressive approach is better suited to a market with low volatility. β_2 represents the learning rate of the function, and $\theta^*(\alpha)$ determines the optimal θ based on the current volatility. Smith's α is calculated using the last N transactions, summing the difference between the price of each transaction, p_i, and the market equilibrium, p^*, which is described below in equation 3.8.

$$\theta(t+1) = \theta(t) + \beta_2(\theta^*(\alpha) - \theta_t) \tag{3.6}$$

$$\alpha = \frac{\sqrt{\frac{1}{N}\Sigma_{i=T-N+1}^{T}(p_i - p^*)^2}}{p^*} \tag{3.7}$$

The bidding strategy utilised by the AA algorithm uses a value called the market equilibrium, p^*, which is an estimation of the point at which demand meets supply, and the net gain of all traders within the market is optimised. It is calculated using the moving average equation, shown in 3.8, which is essentially summing the product of the last N transaction prices p_i with a weight ω_i, which applies the highest weight to the most recent transaction, decreasing by a factor of $\omega_{i-1} = \lambda\omega_i$. This is then divided by the

sum of the weights to get a value for the estimated market equilibrium price, p^*. In BSE, $\lambda = 0.9$ and $N = 5$, meaning the last five transactions were used for the calculation.

$$p^* = \frac{\Sigma_{i=(T-N)+i}^{T} \omega_i p_i}{\Sigma_{i=(T-N)+i}^{T} \omega_i} \tag{3.8}$$

Using the competitive equilibrium, a trading agent can be classified as either an intra-marginal or an extra-marginal trader, with different strategies for each. An intra-marginal buyer is one which has a limit price higher than the current estimated equilibrium (p^*), and a seller with a limit price lower than p^*, meaning it is likely that it will be able to trade at a profit. On the other hand, an extra-marginal buyer will have a limit price lower than p^*, and a seller will have a limit price higher than p^*, so it is unlikely that it will be able to trade profitably.

The aim of the AA aggressiveness algorithm is to establish a target price at which it is most desirable to trade, given the current market state and its current aggressiveness. For an intra-marginal trader, if it has a low aggressiveness (passive) then it will consider prices that are above the current equilibrium price for a buyer, and below the calculated equilibrium price for a seller. This means that it will risk waiting longer to trade in the hope of making more profit from the transaction. Alternatively, an aggressive intra-marginal trader will do the opposite, trading at a price above p^* for a buyer and below p^* for a seller. Because extra-marginal buyers cannot trade above p^*, and sellers cannot trade below p^*, the aggressive trader simply aims to trade at the limit price and ensure that it doesn't make a loss. The more passive extra-marginal traders will also tend towards the limit price but over a longer time period, and so are more likely to end up making a profit.

All three of AA, GDX, and ZIP, have shown to outperform humans on occasion and are all complex, adaptive traders. While ZIP has been outperformed by the other two, all three of these are important to compare DeepTrader against to determine success.

3.3 Overview of Deep Learning

In order to understand the process taken to create, and subsequently replicate, DeepTrader, a basic knowledge of machine learning is required. Machine Learning is ....

Deep Learning is an extension of this and involves ..., it has higher levels of abstraction due to the extra layers of complexity.
Can follow the outline of the google ML course to explain.
Talk about sample weighting,
go on to discuss different types of NN - RNN & LSTM, CNN.

3.4 DeepTrader

DeepTrader, as created by Wray in 2020 [32] and mentioned several times so far, is the focus of this research and is the trader that will be replicated, modified, and analysed, so an understanding of its design is essential.

3.4.1 DeepTrader Features

For the creation of DeepTrader, 13 market metrics were decided upon to try and capture the state of the market at the moment that a transaction occured. These were either taken directly from the LOB, or calculated using the information available. Since the purpose of this research is to investigate how the information being fed in to DeepTrader results in such successful trade prices, it is important to understand each of these market features and what information they actually represent about the market.

The features used are listed below:

1. Time - the time that has elapsed since the start of the market session.

2. Bid/Ask Flag - a binary value indicating whether the trader that initiated the trade is buying or selling.
3. Customer Order Price - the price of the customer limit order for the trader that initiated the trade.
4. Spread - the difference between the current best ask and best bid on the LOB, shown in 3.1.
5. Midprice - the midpoint between the current best ask and best bid. This is the most simplistic method used to estimate the equilibrium price of the market, shown in 3.2.
6. Microprice - an estimation of the market equilibrium which takes in to account the quantity at the top of the LOB. This accounts for differences in the supply and demand, and can be seen fully in equation 3.3.
7. Best Bid - the current best (highest) order on the LOB to buy a unit.
8. Best Ask - the current best (lowest) offer on the LOB to sell a unit.
9. Time since last trade, Δt - the amount of time which has elapsed since the last transaction occured (for the first trade this is simply the time).
10. LOB Imbalance, I - uses the current volume of buy and sell orders to determine whether the LOB is bid or ask heavy, as shown in equation 3.9. This can be a strong indicator for price direction, as demonstrated by a 2015 paper [16], and so is a useful feature to include.

$$I = \frac{q_b - q_a}{q_b + q_a} \tag{3.9}$$

11. Number of Quotes, N - The current total number of orders submitted to the LOB.
12. Market Equilibrium, p^* - An estimation of the current market equilibrium, explained fully in the dissection of the AA trader (see 3.2.4), and shown in equation 3.8.
13. Smith's α - An estimation of the current market volatility, also explained in section 3.2.4 and shown in equation 3.7.

In addition to these 13 inputs, the price of each transaction was also recorded so that it could be used as the output when training the model.

From this list of features, the ones classified as level 1 market data are: time, spread, midprice, microprice, best bid, best ask, Δt. This is because they only require the top of the LOB and the most recent transaction, whereas the bid/ask flag, LOB imbalance, number of quotes, p^*, and α all require further information. The customer order limit price for other traders is actually not available publicly on the LOB, but when DeepTrader is reimplemented back in to BSE it can use its own customer order prices as input. For training purposes, because this is what BSE is designed for, the customer order prices of other traders was stored along with the other features.

In terms of importance, it is highly likely that the customer order is crucial to determine a good trade price because it needs to be known for a trader to trade profitably. With regards to the other metrics, a paper published in 2009 concluded that the top of the LOB, in particular the best ask and best bid, contribute 80% of the information required for price discovery, with the other levels of the LOB left to contribute only 20% [6]. This suggests that level 1 market data will be much more valuable and contribute more to DeepTrader's success.

3.4.2 Training DeepTrader

The method used to train DeepTrader was to use BSE to generate millions of trades across various market populations (and permutations of traders) to vary the data as much as possible. In addition to this, a method within BSE was used to vary the supply and demand over the course of the session, following an increasing sinusoidal function as shown in Figure 3.2, therefore exposing the traders to a range of different supply and demand curves. However, using this only shifts the supply and demand curves up or down changing the market equilibrium point, but it doesn't change the shape of the curves, so this

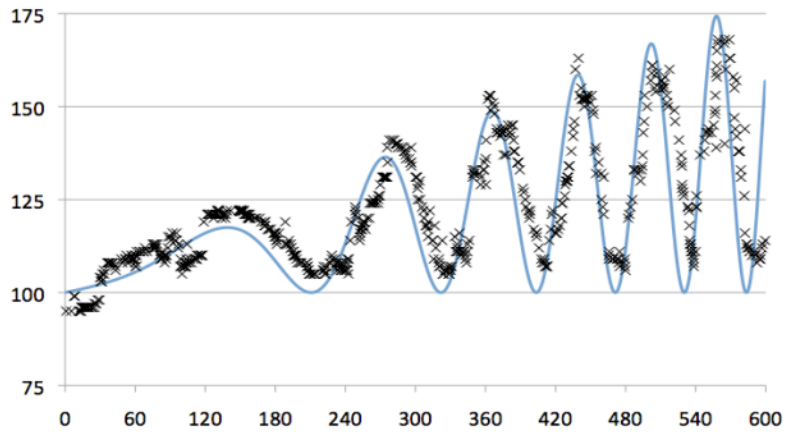

Figure 3.2: Function used to dynamically vary the supply and demand curves during the market session in BSE (Image taken from [8])

may be a limitation in terms of data quality. Other methods may have been used to make the data more realistic however the full BSE configuration is not specified in Wray's work [32].

This data was then normalised and used to train an RNN, which uses an LSTM module as the input layer, followed by three fully connected layers with 5, 3, and 1 nodes respectively as shown in Figure 3.3. Whilst DeepTrader employs an LSTM module, the way in which the data is processed doesn't actually utilise the main advantages of recurrent neural networks. As mentioned before, RNN's are designed to capture a series of sequential data, for example the last 60 seconds of a stock price, and output the direction that it predicts the stock price will move next. However, in this application the input is not sequential data, it is simply the current state of the market represented by the 13 inputs chosen above.

Therefore, whilst the LSTM module may be advantageous, it is not performing its main purpose of remembering past events to influence future ones because it is only considering 13 metrics at a time, and so could be replaced by a different module. For example, a CNN module was used to capture the spatial structure of the LOB, alongside other modules, which resulted in accurate price prediction in work carried out by Zhang et al. [33]. Regardless, DeepTrader produced impressive results and so will be replicated using the LSTM module, but another potential avenue to explore would be to actually use sequential data to train the network.

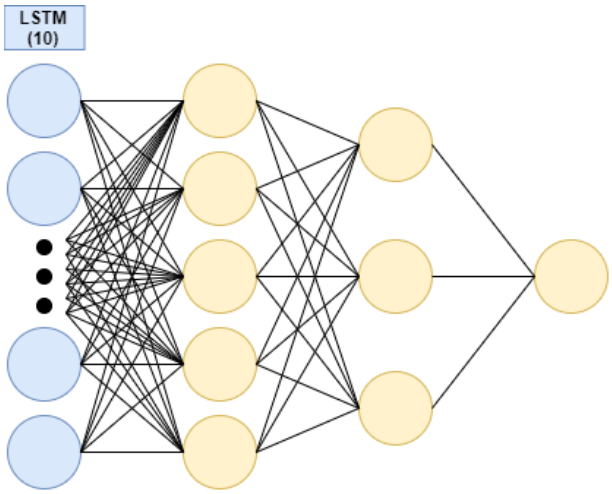

Figure 3.3: Diagram of the network architecture used by Wray to train DeepTrader.

The training configuration used by Wray was reported in full and so was straightforward to follow for

the execution of this project (see [32] for full details), although some parameters were modified because they seemed to achieve better results (see 4.1.3).

3.4.3 DeepTrader Performance

Experimental results were gathered by first reimplementing DeepTrader back in to BSE, which was relatively simple because the process was well documented and the code was included. Wray's code can be seen in appendix A, but to summarise: when DeepTrader's *getOrder*() function is called, it first checks if it currently has a customer order available. If it does, the current market state is captured (in the exact same way as when the training data was collected), the values are normalised and are then input to the model. The model prediction is then de-normalised to get a trade price, which is then checked against the limit price of the current customer order - if the trade price is greater than the limit price for a buyer, the price is set to the limit price, and if the price is less than the limit price for a seller it is also set to the limit price (this prevents the trader from trading at a loss). Finally, the order is submitted to the LOB.

On real world exchanges the other traders and their proportions within the market are unknown, and therefore it is hard to draw an absolute comparison between two traders. Two types of experiment were carried out to directly measure DeepTrader's performance against each other trader: one-in-many tests, and balanced group tests. Both of these approaches involve populating the market with only two different traders, and so a direct comparison can be drawn. The one in many tests involved adding a DeepTrader buyer and seller to the market amongst 39 buyers and sellers of another trading strategy. The balanced group tests involved adding an equal amount of each trading strategy (20 buyers and sellers of each) to the market, which is considered the fairest method of comparing traders.

Once the experiments were completed, a confidence interval analysis was used to confirm whether DeepTrader had outperformed the other traders within a 90% confidence interval. This provides a range of values for which it is 90% certain that the mean lies between, and so if the lower bound of the first trader's range is above the upper bound of a second trader's range, it is 90% certain that the first trader has outperformed the second. Using this method takes in to account the variation of DeepTrader's performance across all market sessions, which is important given the high variability produced by DeepTrader. The equation to calculate the confidence interval can be seen below, with c_{lower} and c_{upper} representing the upper and lower confidence intervals for each trader.

$$c_{lower} = \bar{x} - \frac{Z^*_{0.9}\sigma}{\sqrt{n}} \qquad c_{upper} = \bar{x} + \frac{Z^*_{0.9}\sigma}{\sqrt{n}} \tag{3.10}$$

where $\bar{x}$ represents the mean of the values, σ is the standard deviation, n is the total number of values, and $Z^*_{0.9}$ is the quantile defined for values drawn from a normal distribution, equal to $Z^*_{0.9} = 1.645$.

A summary of these confidence interval results achieved by DeepTrader can be seen below in Figure 3.4, showing that in all but one situation (DeepTrader vs AA in balanced group tests) DeepTrader matched or outperformed all other traders with 90% confidence. This is included to illustrate the number of traders which DeepTrader could outperform, although doesn't indicate how much by because this is covered in the later comparison of DeepTrader and DeepTrader2.

	Balanced Group	
	result	comment
AA	↓	only test where DeepTrader was outperformed
GDX	†	DeepTrader significantly outperformed
Giveaway	-	no statistical difference in performance
Shaver	†	DeepTrader outperformed
Sniper	†	DeepTrader significantly outperformed
ZIC	†	DeepTrader significantly outperformed
ZIP	-	no statistical difference in performance

	One In Many	
	result	comment
AA	†	DeepTrader outperformed but with a high variability
GDX	†	DeepTrader significantly outperformed
Giveaway	†	DeepTrader outperformed
Shaver	†	DeepTrader outperformed but with a high variability
Sniper	†	DeepTrader outperformed but with a high variability
ZIC	†	DeepTrader outperformed but with a high variability
ZIP	†	DeepTrader outperformed but with a high variability

Figure 3.4: Summary of the Wray's experimental results, [32]), comparing DeepTrader to all other traders in one-in-many and balanced group tests.

Chapter 4

Project Execution

The execution of this research project is split in to two distinct sections:

- The first component aims to replicate Wray's work and re-create a version of DeepTrader which can attain similar performance to the original, which we will call DeepTrader2 to save confusion. This section is very much an implementation of existing work, although also contains some experimentation around market/training parameters which were not previously specified, or for parts which I believed could be improved.
- The second section is a novel, purely experimental section focused on systematically modifying DeepTrader2, running neural network analysis and then market simulations to determine the importance of each feature. This allowed conclusions to be drawn regarding the underlying mechanisms driving the original DeepTrader's success, discussed in section 5.2, laying the foundations for future work to build in this direction.

4.1 Replicating DeepTrader

4.1.1 Data Collection

The first step to replicate DeepTrader was to collect the data necessary for training the neural network. The market conditions for this were well documented and so were straightforward to setup:

- 40 buying agents and 40 selling agents in each trading session, for a total of 80 traders
- Each session contained 4 of the possible 7 trading agents and each possible combination of these was used, leading to 35 distinct combinations of 4 agents.
- The trader proportions in each session were varied in combinations of (20, 10, 5, 5), (10, 10, 10, 10), (15, 10, 10, 5), (15, 15, 5, 5), (25, 5, 5, 5), and all 35 possible unique permutations of these.
- This led to 35 proportions multiplied by 35 trader combinations to generate 1225 unique market configurations, with 32 simulated market sessions run for each configuration (totalling 39,200 total market sessions).
- Each of these sessions was run for 600 simulated seconds.

While BSE is a minimal simulation of a real-world exchange, the market parameters used when generating the data were set to be as realistic as possible; customer order prices were generated randomly from a specified range, the supply and demand were dynamically varied over the session, and the times at which customer orders were distributed followed an exponential distribution (and therefore was not constant).

When collecting data, it was assumed that to create a neural network that can trade profitably, the data used to train it must come from profitable traders, otherwise it would learn the behaviour of less profitable traders and make unfavourable trades. For this reason, the initial dataset that was collected only included the best buyer and the best seller from each market session, producing just over 380,000 total trades with which to train and validate the neural network. However, when re-implemented back in to BSE the trader was not successful, failing to outperform even the basic ZIC and Shaver strategies.

This failure was attributed to the small size of the dataset, and therefore training the DLNN on more trading data would likely produce better results. The next idea was to record every trade from every trader, but to also record the profit of each trader and rank each of them by profitability - 1 being the most profitable trader, 80 being the least profitable. These rankings would allow the DLNN to prioritise samples from traders which overall traded more profitably (i.e. made good trades) over samples from traders which perhaps didn't perform so well, through a process known as sample weighting (discussed in more detail here [REF]).

Once BSE was modified, the training data was generated by utilising Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances, which are virtual Linux machines on which scripts can be run remotely. The Python Botov3 library was used to set up 32 instances, which could then be accessed and automated using the Python Fabric library and TMUX. This allowed the data generation to be carried out in parallel and reduced the run time from multiple days to only a few hours. To pre-process the data for training the model a Python script was used to combine the multiple csv files in to a single file.

When training a neural network, the range of each feature can determine its impact on the model predictions. For example, a feature ranging between 0 and 1000 will have more influence than a value ranging between 0 and 10. To ensure that each feature contributes equally and to prevent any predictions being skewed by a certain feature, min-max normalisation was carried out (using the equation below) on all the data to ensure all values were between the range 0 - 1.

$$x_{norm} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

4.1.2 Neural Network Architecture

The network architecture was based on that which was specified in the previous literature, and is shown in the figure below. It contains an LSTM input layer with 10 nodes and an input shape of (13, 1) to accommodate all 13 market metrics, followed by 3 fully connected layers with 5, 3, and 1 node respectively. In the variations following which change the number of features passed to the neural network, the input shape also changes accordingly.

**** INSERT IMAGE HERE OF NETWORK ARCHITECTURE ****

Dropout

Whilst the original architecture was eventually kept the same, including a dropout layer was investigated because in RNN's this has been shown to improve performance by reducing overfitting to the data, however this comes at the cost of having less data to train on [REF]. In this case it resulted in worse performance from the trading agents when reimplemented back in to BSE. The dropout layer was added just after the LSTM layer, and for the experiments shown below dropout was set to 0.2 - i.e. 20% of the training data was randomly removed in each epoch.

**** INSERT SUMMARY GRAPH OF DROPOUT VS NO DROPOUT ****

4.1.3 Neural Network Tuning

Ranking Traders

As mentioned above, during data collection each trader was ranked according to how profitable it was within that individual session. This could then translate to a weight which could be associated with every trade which that trader made, allowing the data with a greater weight to more heavily influence the predictions made by the neural network. To achieve this, an array of values with weights for each sample is passed to the model when it is being trained and the loss function is calculated by taking the weights of each sample in to account. Because the goal when training the model is to minimise loss, the greatest improvements come from fitting the model more closely to those samples which are weighted more heavily, meaning that the resultant network is more closely fitted to the data input by traders with a weight of 1 than with a weight of 0.05.

However, when trained using identical parameters, and with the final loss as similar as possible, ranking the traders didn't appear to improve the performance of DeepTrader2 when re-implemented back in to BSE. This can be seen in the figure below.

**** GRAPH OF 5, 10, 20 EPOCHS TRADERS FOR RANKED VS UNRANKED ***

Overfitting Data

Even though the layers of the network had been established, several model parameters needed to be tuned to minimise the loss of the model and improve its price predictions. To avoid overfitting to the training data a validation set was used to test the model once it had been trained, ensuring that it could make accurate predictions on new data. However, although the DLNN achieved a very low loss both on the testing and training data, once reimplemented in to BSE it became apparent that loss was not a good predictor of trader performance. Even though the model wasn't overfitting the data in the conventional sense, as indicated by the low loss value achieved on the validation set, models which were trained for too many epochs performed worse in BSE than models trained for fewer epochs.

The parameters which affect the final loss value the most are the learning rate and the number of epochs they are trained for. Because the learning rate had already been specified in previous work and produced successful results, and also because from experience learning rate can be much harder to tune when training a model, the number of epochs was varied to determine an optimal configuration.

**** INSERT SUMMARY GRAPH OF 3 DIFFERENT TRADERS' PERFORMANCE: 5, 10, 20 EPOCHS ****

Learning Rate Decay

An optional parameter which can be included alongside learning rate when training the model is learning rate decay. This decreases the learning rate by a fixed amount each epoch, allowing for better optimisation and fine tuning once the loss has decreased by a certain amount. This parameter was never mentioned in the original DeepTrader paper, although has shown in various literature that it can help with model optimisation and so was included in this model [REF]. The results below show that this optimisation does translate to a better trader performance when 'live' trading in BSE.

**** INSERT SUMMARY GRAPH OF DECAY VS NO DECAY ****

Final Configuration

After exploring these different combinations of learning rate, epochs, decay and dropout, the final version of DeepTrader2 used in this work was trained using the following:

- 1.5×10^{-5} learning rate
- 1×10^{-6} decay rate
- 16,384 batch size
- Trained for a total of 10 epochs

4.1.4 BSE Experiments

Once the final network configuration had been established, the model could be implemented as a trading strategy back in to BSE. BSE was then reconfigured to run the two different experiments used to originally evaluate DeepTrader's performance: the one-in-many test and the balanced group test. One-in-many testing populates the market with a single instance of the trader to be compared (DeepTrader) and the rest of the market with another trader, while balanced group testing populates the market with an equal number of both traders.

One-in-Many Tests

For the experiments shown below, one DeepTrader2 buyer and one DeepTrader2 seller were put in to the market alongside 39 buyers and 39 sellers of each trading strategy, for a total of 80 trading agents total in each session. These sessions were then run 100 times each and the average profit per trader (APPT) was calculated for each session, providing 100 data points to plot and compare against the original DeepTrader.

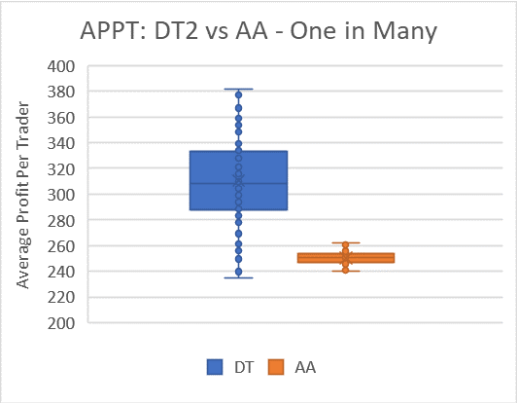

Figure 4.1: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × AA

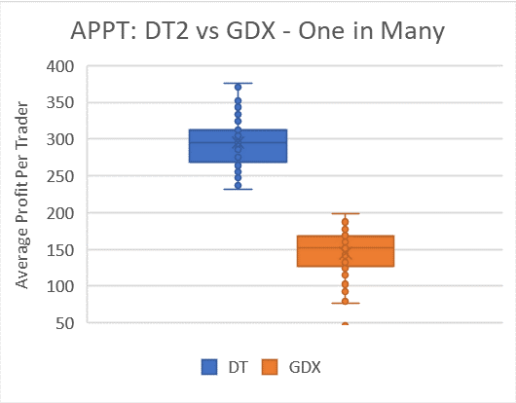

Figure 4.2: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × GDX

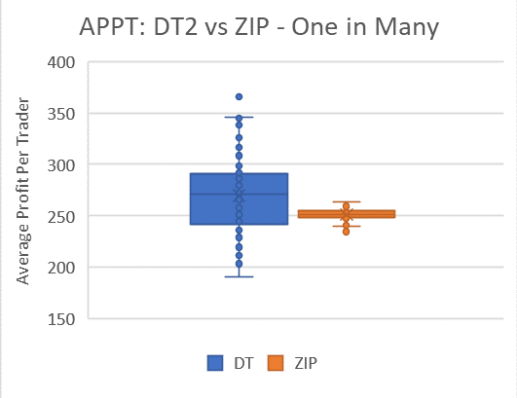

Figure 4.3: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × ZIP

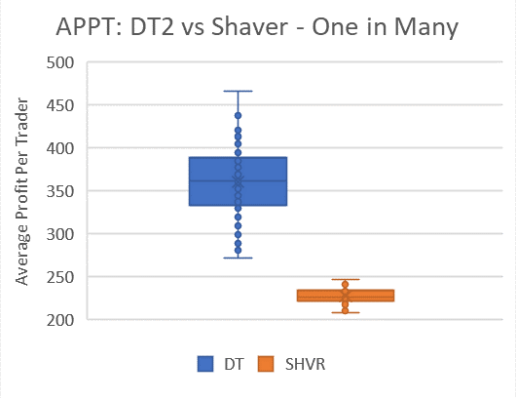

Figure 4.4: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Shaver

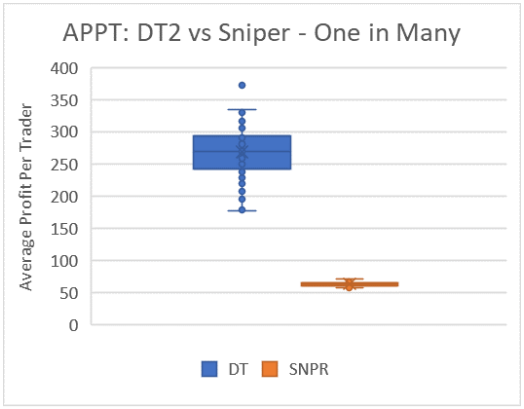

Figure 4.5: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Sniper

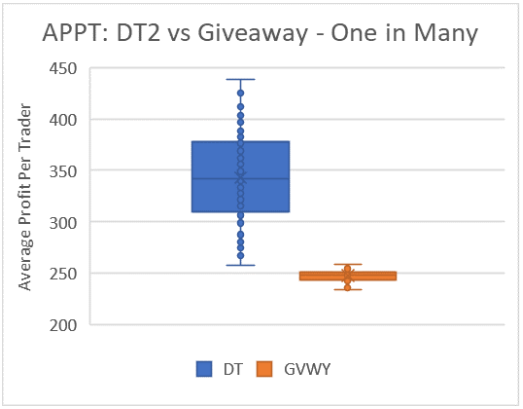

Figure 4.6: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Giveaway

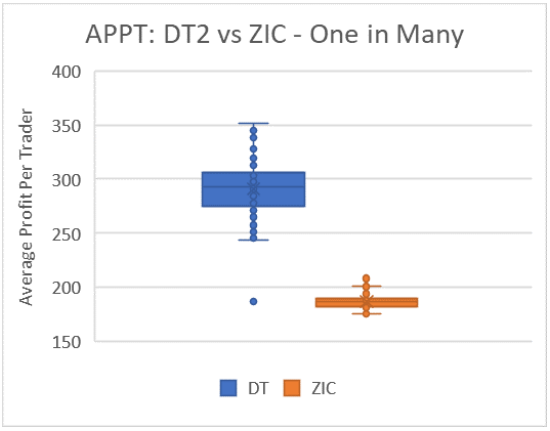

Figure 4.7: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × ZIC

Balanced Group Tests

In each experiment shown below, 20 DeepTrader2 buyers and 20 DeepTrader2 sellers populated the market alongside 20 buyers and 20 sellers of another trading strategy. Again, the APPT was calculated in each session for both traders and these distributions were plotted against the original DeepTrader to compare.

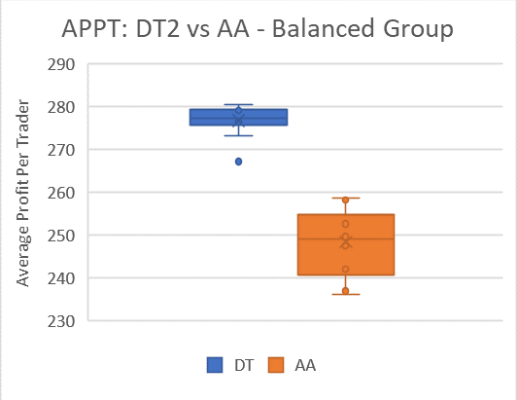

Figure 4.8: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × AA

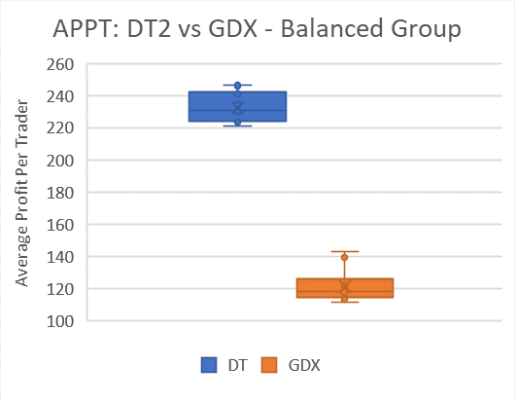

Figure 4.9: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × GDX

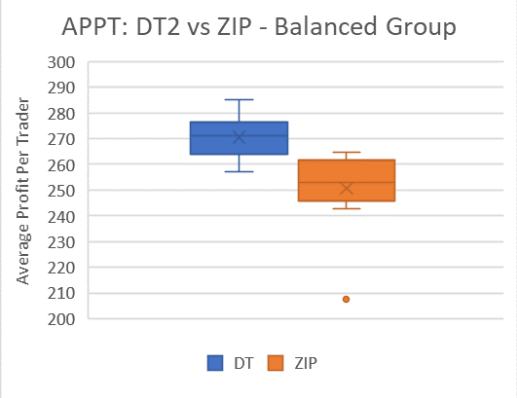

Figure 4.10: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × ZIP

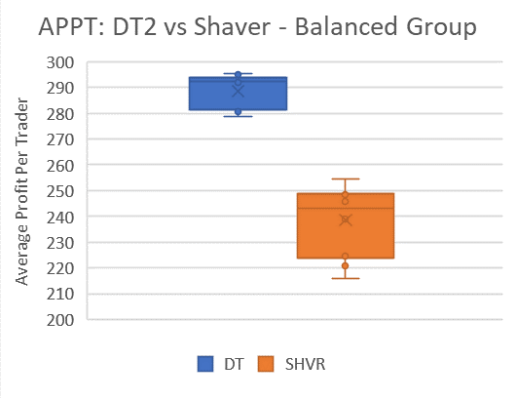

Figure 4.11: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Shaver

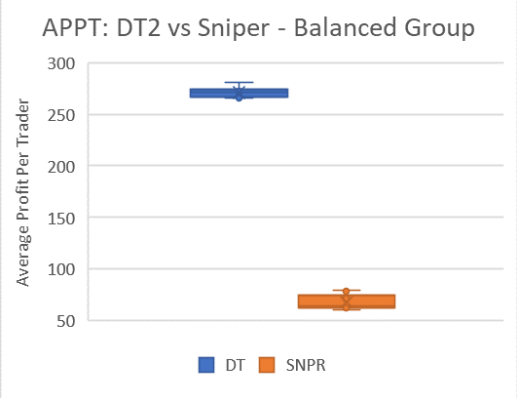

Figure 4.12: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Sniper

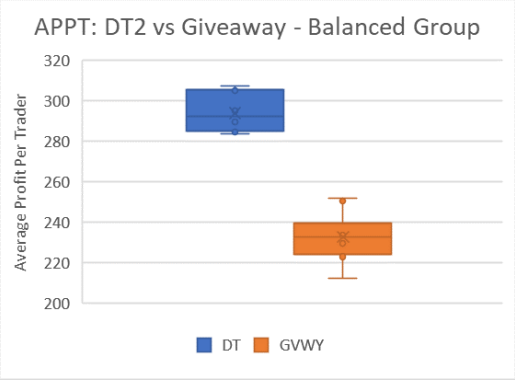

Figure 4.13: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Giveaway

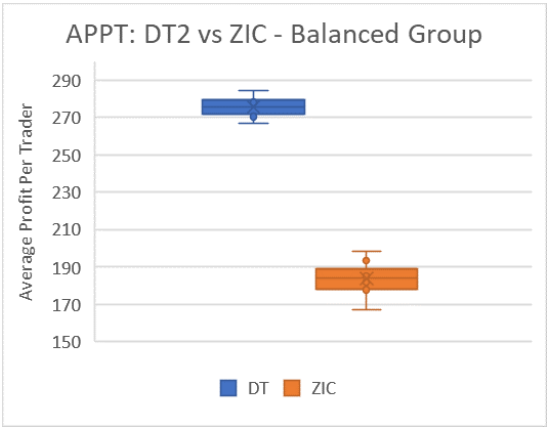

Figure 4.14: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × ZIC

4.2 DeepTrader2 Evaluation

To determine how well the original DeepTrader trading agent had been replicated, the same statistical analysis used previously was carried out to allow trader performance to be compared. In Wray's work, a confidence interval analysis was used to ascertain whether DeepTrader had actually outperformed the other traders, within a 90% confidence interval (see 3.4.3). For each comparison, if the lower bound of the first trader's APPT is higher than the upper bound of the second trader's APPT, then it can be concluded that the first trader can outperform the second with 90% confidence.

4.2.1 One-in-Many Tests

Table 4.1 below shows the results of the confidence interval analysis for the one in many tests carried out. The situations in which DeepTrader2 outperformed the other trader, with a 90% confidence interval, are highlighted green. Since the lower bound of DeepTrader2's profit distribution was greater than the upper bound of the other trader's profit in every situation, it can be concluded that DeepTrader 2 successfully outperformed every other trader in all one in many tests, much like the original DeepTrader.

Traders	Trader X		DeepTrader2	
	c_{lower}	c_{upper}	c_{lower}	c_{upper}
AA	249.767	251.328	304.742	316.161
GDX	139.923	150.236	290.005	300.320
ZIP	250.585	252.304	263.584	275.696
Shaver	226.064	228.937	353.765	366.505
Sniper	63.428	64.464	262.665	274.595
ZIC	185.709	187.769	286.740	295.440
Giveaway	246.748	248.467	335.830	349.790

Table 4.1: Confidence interval analysis of DeepTrader2's average profit for 100 one in many tests.

4.2.2 Balanced Group Tests

Table 4.2 below shows the results of the confidence interval analysis for the balanced group tests carried out. The situations in which DeepTrader2 outperformed the other trader, with a 90% confidence interval, are highlighted green. Again, since the lower bound of DeepTrader2's profit distribution was greater than the upper bound of the other trader's profit in every situation, it can be concluded that DeepTrader 2 successfully outperformed every other trader in all balanced tests, actually outperforming the original DeepTrader.

Traders	Trader X		DeepTrader2	
	c_{lower}	c_{upper}	c_{lower}	c_{upper}
AA	247.132	249.614	276.054	277.289
GDX	119.907	123.263	230.880	233.869
ZIP	248.495	253.080	269.364	271.943
Shaver	236.454	240.641	287.477	289.553
Sniper	66.804	69.0513	270.233	271.717
ZIC	182.410	185.115	274.827	276.468
Giveaway	230.844	234.581	292.240	295.210

Table 4.2: Confidence interval analysis of DeepTrader2's average profit for balanced tests.

4.2.3 Overall Comparison

Overall, DeepTrader2 performed well in most experiments although didn't quite match the performance demonstrated by the original DeepTrader. While DeepTrader2 outperformed all other trading strategies in the one-in-many tests, DeepTrader2 couldn't match that performance when trading against a market populated by ZIP traders.

**** INSERT TABLE SUMMARY OF NEW VS OLD DEEPTRADER (ONE IN MANY) ****
When considering the balanced group tests, again DeepTrader2 performed very well against most trading strategies...

**** INSERT TABLE SUMMARY OF NEW VS OLD DEEPTRADER (BALANCED) ****

4.3 Individual Feature Importance

Once DeepTrader had been replicated and demonstrated a reasonably similar performance, the investigation in to why it performs so well could begin. To try and understand the underlying mechanisms behind the trade prices it generates, the first step was to isolate the features which produced these profitable trade prices. Because each of the 13 features are based on different market metrics, understanding which ones contribute most to the trade price provides an idea of which market parameters are the most important when trading.

4.3.1 Permutation Importance

Permutation importance is calculated once the network has been trained and involves systematically disrupting the input to each feature, essentially causing that feature to then compromise any predictions that the model tries to make. All 13 tests with 'corrupted' data should perform worse than with the original, uncorrupted data, and by plotting how much worse each variation performs the importance of each feature can be inferred - larger disparities in loss meaning that feature has a greater impact on model performance, and smaller disparities meaning that feature doesn't really affect model performance.

In these experiments, the final version of DeepTrader was tested against the validation set to set a benchmark loss value with which to compare against. Then, each feature in the validation data set was randomly shuffled one at a time - i.e. the first column was shuffled, then the network was tested and its loss value recorded, then the first column was reset and the second column shuffled, and repeat. The results from this experiment can be shown below in 4.15, the features showing the highest loss having the greatest impact on model performance.

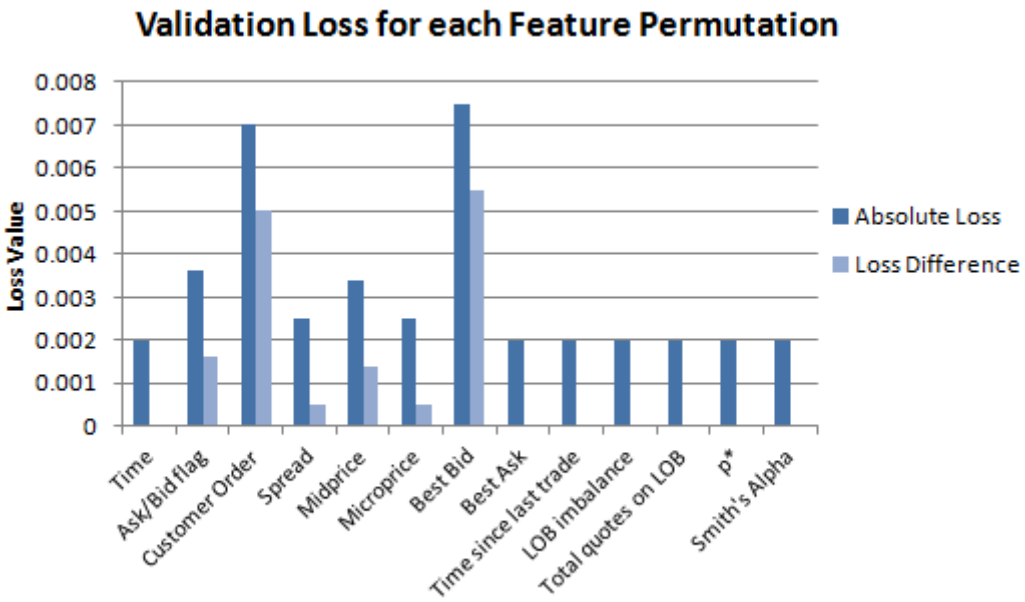

Figure 4.15: Loss value and the difference in loss from ideal plotted for each feature permutation

From Figure 4.15 it can be seen that the best bid feature has the most significant impact on prediction accuracy, with customer order being the second most significant. Then, the flag indicating whether the customer order was a bid or an ask, followed by the midprice, microprice, and spread. When every other

feature was randomised there was no perceived change in loss, and therefore individually they each have no effect on the predictions that the model makes.

4.3.2 Initial Feature Importance Analysis

From the results shown in Figure 4.15, the following table of ranked features can be produced, showing the most important down to the joint least important features:

Rank	Training Loss	Features
1	0.0075	Best Bid
2	0.0070	Customer Order
3	0.0036	Bid/Ask Flag
4	0.0034	Midprice
5	0.0025	Microprice
	0.0025	Spread
6	0.0020	Time
	0.0020	Best Ask
	0.0020	Time since last trade
	0.0020	LOB imbalance
	0.0020	Number of quotes on LOB
	0.0020	P*
	0.0020	Smith's Alpha

Table 4.3: DeepTrader training features ranked in order of importance

The significance of several of these features is fairly intuitive, although for other features it is interesting how little impact they had on model predictions. For example, the best bid combined with the spread gives the complete range of prices at which the next trade will occur, and therefore the best ask essentially becomes redundant, explaining its minimal impact on the model performance. The high importance of the best bid, midprice, and spread indicate that the trader is only concerned with the market metrics associated with the top of the LOB, essentially disregarding all current quotes less than the best bid and higher than the best ask. The importance of the customer order price and the bid/ask flag are also intuitively significant, since the trade price will have to be lower than the customer order if the order is a bid (flag = 0) or higher than the customer order if the order is an ask (flag = 1).

Because of how the LSTM module was used in this network, it is not too surprising that the time of each trade had no effect. Instead of training the network on a series of previous trades, and then using that to predict the next trade like in price forecasting, each individual market snapshot is considered independently of previous trades. Therefore, it became irrelevant when a trade occured within a market session. However, it is interesting that the time since last trade had such little impact because this indicates that the trading strategy employed doesn't try to capitalise on quick shifts in the market, and instead just tries to trade at a good price irrespective of the trades that have just happened.

4.4 DeepTrader Variations

Further to this individual feature analysis, the next step was to consider if the six features (ranks 1 - 5 in 4.3) which seem to actually influence the model predictions are all that is necessary for the model to function. This would also confirm that the seven other features (rank 6 in 4.3), which appear to have no bearing on the trading performance of the model, were indeed irrelevant. This involved creating, training, and testing two more versions of DeepTrader; $\text{DeepTrader}_{\text{Best6}}$ containing only the top 6 features detailed above, and $\text{DeepTrader}_{\text{Worst7}}$ containing only the bottom 7 features.

4.4.1 Training $\text{DeepTrader}_{\text{Best6}}$ & $\text{DeepTrader}_{\text{Worst7}}$

The two variations of DeepTrader were created by removing all but the desired features from the training data and modifying the input shape of the model, then training the model using the same parameters defined above in section 4.1.3. However, for the results in table 4.4, the models were trained for 20 epochs

instead of 10 so that any difference in training speed was mitigated (due to the different sizes of training data) and each model was allowed to converge to show the disparities in loss.

Trader	Features	Training Loss	Validation Loss
DeepTrader	All Features	0.0010	0.0014
$\text{DeepTrader}_{\text{Best6}}$	Top 6 Features	0.0009	0.0011
$\text{DeepTrader}_{\text{Worst7}}$	Bottom 7 Features	0.0083	0.0098

Table 4.4: Training and validation loss results from training each DeepTrader variation for 20 epochs

As can be seen in table 4.4, the original DeepTrader model and $\text{DeepTrader}_{\text{Best6}}$ achieved similar training loss, with $\text{DeepTrader}_{\text{Best6}}$ even marginally outperforming the original on both training and validation datasets. $\text{DeepTrader}_{\text{Worst7}}$ performed considerably worse than both other DeepTrader variations, indicating that the price predictions were much less accurate. Whilst absolute loss isn't an entirely accurate indicator of how well the trader will perform in BSE (as stated earlier), the relative difference in loss here strongly suggests that both DeepTrader and $\text{DeepTrader}_{\text{Best6}}$ will achieve a similar performance, whereas $\text{DeepTrader}_{\text{Worst7}}$ can be expected to perform much worse.

4.4.2 BSE Experiments

In the same way as the original DeepTrader, the two variations were reimplemented back in to BSE to trade 'live'. To determine their performance, the same one-in-many and balanced group tests as before were carried out to assess their performance, with the results shown below.

One-in-Many Tests

**** INSERT BOX PLOTS OF ORIGINAL VS BEST VS WORST VS EACH TRADER (ONE IN MANY) ****

Balanced Group Tests

**** INSERT BOX PLOTS OF ORIGINAL VS BEST VS WORST VS EACH TRADER (BALANCED) ****

Could also include a confidence interval analysis? Should probably leave that until next chapter?

Chapter 5

Analysis of Results

5.1 DeepTrader Analysis

- Maybe do a more in depth analysis of section 4.2.3 - talk about the reasons behind why mine potentially didn't perform as well as Aaron's?
- Discuss more intricate differences - e.g. mine did better with x, his did better with y. These details are very likely due to differing market conditions - this could mean that the favourable performance of DeepTrader on this occasion, and on Aaron's occasion, could be an artefact of the experimental setup - regardless of the amount of market sessions run, if DeepTrader is intrinsically designed to perform better in those conditions, it will consistently perform well.
  - Include analysis of different market conditions affecting trader performance in one-in-many tests
  - Further work definitely necessary to determine the actual performance of DeepTrader
  - Would be interesting to investigate this for the different feature variations as well to see if different features offer advantages in certain situations?
- Future work should probably include an exhaustive analysis exploring multiple different market conditions.
- Future work should look in to an appropriate cut off for trader rank - pulled 40 out of thin air, likely that only the top 20 would perform even better???

5.2 Feature Variation Analysis

- Perform an analysis on the results reported in 4.4, possibly a confidence interval analysis between best 6 and original to determine if the performance is actually the same/better.
- Maybe have a section on the worst7 trader, discussing each of the metrics which were deemed irrelevant - how big of a disadvantage this trader is at from not having the customer order.
- Could potentially create a best trader which has best ask instead of best bid? See if they're interchangeable as long as spread is included?
- Talk about how I explored removing a single input but the variation in trader performance was too high to distinguish any differences in performance

Chapter 6

Conclusion

A compulsory chapter, of roughly 5 pages

The concluding chapter of a dissertation is often underutilised because it is too often left too close to the deadline: it is important to allocation enough attention. Ideally, the chapter will consist of three parts:

1. (Re)summarise the main contributions and achievements, in essence summing up the content.
2. Clearly state the current project status (e.g., "X is working, Y is not") and evaluate what has been achieved with respect to the initial aims and objectives (e.g., "I completed aim X outlined previously, the evidence for this is within Chapter Y"). There is no problem including aims which were not completed, but it is important to evaluate and/or justify why this is the case.
3. Outline any open problems or future plans. Rather than treat this only as an exercise in what you *could* have done given more time, try to focus on any unexplored options or interesting outcomes (e.g., "my experiment for X gave counter-intuitive results, this could be because Y and would form an interesting area for further study" or "users found feature Z of my software difficult to use, which is obvious in hindsight but not during at design stage; to resolve this, I could clearly apply the technique of Smith [7]").

Bibliography

[1] James Zou Amirata Ghorbani, Abubakar Abid. Interpretation of neural networks is fragile. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, volume 33, July 2019.

[2] Andrés Arévalo, Jaime Nino, German Hernandez, Javier Sandoval, Diego León, and Arbey Aragón. Algorithmic trading using deep neural networks on high frequency data. In *Applied Computer Sciences in Engineering: 4th Workshop on Engineering Applications*, pages 144–155, 2017.

[3] Alessandro Bigiotti and Alfredo Navarra. Optimizing automated trading systems. In *Digital Science*, pages 254–261. Springer, 2019.

[4] Vighnesh Birodkar, Hossein Mobahi, and Samy Bengio. Semantic redundancies in image-classification datasets: The 10% you don't need. *CoRR*, abs/1901.11409, 2019.

[5] Arthur Le Calvez. Deep learning can replicate adaptive traders in a limit-order-book financial market. Master's thesis, University of Bristol, Bristol, UK, 2018.

[6] Charles Cao, Oliver Hansch, and Xiaoxin Beardsley. The information content of an open limit-order book. *Journal of Futures Markets*, 29:16 – 41, 01 2009.

[7] D. Cliff. The bristol stock exchange, 2012. GitHub open-source repository at https://github.com/davecliff/BristolStockExchange, last accessed on 23/05/2020.

[8] Dave Cliff. An open-source limit-order-book exchange for teaching and research. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI 2018)*, pages 1853–1860. Institute of Electrical and Electronics Engineers (IEEE), 1 2018.

[9] Dave Cliff. Exhaustive testing of trader-agents in realistically dynamic continuous double auction markets: Aa does not dominate. In *Proceedings of the 11th International Conference on Autonomous Agents and Artificial Intelligence (ICAART 2019)*, volume 2, pages 224–236. SciTePress, 3 2019.

[10] Dave Cliff and Janet Bruten. Minimal-intelligence agents for bargaining behaviors in market-based environments. Technical Report HPL-97-91, Hewlett Packard, August 1997.

[11] Philip Treleaven Dave Cliff, Dan Brown. Technology trends in the financial markets: A 2020 vision. Technical Report DR-3, UK Government Office for Science, Foresight, 2011.

[12] Marco De Luca and Dave Cliff. Human-agent auction interactions: Adaptive-aggressive agents dominate. pages 178–185, 2011.

[13] Martin D. Gould et al. Limit order books. *Quantitative Finance*, 13:1709–1742, 4 2013.

[14] Steven Gjerstad and John Dickhaut. Price formation in double auctions. *Games and Economic Behavior*, 22(1):1–29, 1998.

[15] Dhananjay K. Gode and Shyam Sunder. Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *The Journal of Political Economy*, 101(1):119–137, 1993.

[16] Martin D. Gould and Julius Bonart. Queue imbalance as a one-tick-ahead price predictor in a limit order book. *Market Microstructure and Liquidity*, 02(02), 2016.

[17] Martin D. Gould, Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, and Sam D. Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

[18] Henry Hanifan and John P Cartlidge. Fools rush in: Competitive effects of reaction time in automated trading. In *12th International Conference on Agents and Artificial Intelligence (ICAART-2020)*, volume 1, pages 82–93. SciTePress, March 2020.

[19] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Weiwei Jiang. Applications of deep learning in stock market prediction: recent progress, 02 2020.

[21] Marco De Luca and Dave Cliff. Agent-human interactions in the continuous double auction, redux. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*, pages 351–358. SCITEPRESS, 2011.

[22] John Cartlidge Marco De Luca, Charlotte Szostek and Dave Cliff. Studies of interactions between human traders and algorithmic trading systems. Technical Report DR-13, UK Government Office for Science, Foresight, 2011.

[23] Jeffrey O. Kephart Rajarshi Das, James E. Hanson and Gerald Tesauro. Agent-human interactions in the continuous double auction. In *International Joint Conferences on Artificial Intelligence*, page 1169–1176, 2001.

[24] J. Rust, J. Miller, and R. G. Palmer. Behavior of trading automata in a computerized double auction market. In *The Double Auction Market: Institutions Theories & Evidence*, pages 155–198, 1992.

[25] Maddison C. et al. Silver D., Huang A. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[26] Daniel Snashall and Dave Cliff. Adaptive-aggressive traders don't dominate. In *Agents and Artificial Intelligence, 11th International Conference, ICAART 2019*, pages 246–269, 2019.

[27] Gerald Tesauro and Jonathan Bredin. Strategic sequential bidding in auctions using dynamic programming. pages 591–598, 01 2002.

[28] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. 09 2001.

[29] K. Tibrewal. Can neural networks be traders? explorations of machine learning using the bristol stock exchange, 2017. Unpublished Masters Thesis, University of Bristol, Bristol, UK.

[30] Daniel Vach. Comparison of double auction bidding strategies for automated trading agents. Master's thesis, Charles University, Prague, Czech Republic, 2015.

[31] Perukrishnen Vytelingum. The structure and behaviour of the continuous double auction, 2006. Ph.D. Thesis, University of Southampton, Southampton, UK.

[32] Aaron Wray. Deeptrader: A deep learning approach to training an automated adaptive trader in a limit-order-book financial market. Master's thesis, University of Bristol, Bristol, UK, 2020.

[33] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11), 08 2018.

Appendix A

Appendix

```
class DeepTrader (Trader):
    def getorder (self, time, countdown, lob):
        if len (self.orders) < 1:
            # no orders : return NULL
            order = None
        else:
            qid = lob['QID']
            tape = lob['tape']
            otype = self.orders[0].otype
            limit = self.orders[0].price

            # creating the input for the network
            x = self.create_input(lob)

            normalized_input = (x- self.min_vals[:self.n_features]) / (
                self.max_vals[:self.n_features]- self.min_vals[:self.n_features])
            normalized_input = np.reshape(normalized_input, (1, 1, -1))

            # retrieving the networks output
            normalized_output = self.model.predict(normalized_input)[0][0]

            # denormalizing the output
            denormalized_output = ((normalized_output) * (
            self.max_vals[self.n_features]
            - self.min_vals[self.n_features]))
            + self.min_vals[self.n_features]
            model_price = int(round(denormalized_output, 0))

            if otype == "Ask":
                if model_price < limit:
                    self.count[1] += 1
                    model_price = limit
            else :
                if model_price > limit:
                    self.count[0] += 1
                    model_price = limit

            order = Order(self.tid, ...)
            self.lastquote = order
        return order
```