University of BRISTOL

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.

Thursday 24th September, 2020

Declaration

[illegible]

Matthew Meades, Thursday 24th September, 2020

Executive Summary

Technological advances in the financial world have evolved how traders interact with financial exchanges, with a considerable amount of trading now carried out using trading algorithms. Despite this, human traders are still employed by companies, valued for their intuition, experience, and insight. Recent research has focused on using deep learning to capture these capabilities, with one paper training a Deep Learning Neural Network (DLNN) to trade simply by observing an existing profitable algorithmic trader. This offered an impressive proof of concept, suggesting that human traders could also be observed and learned from in the same way, however this DLNN was only provided with limited market data.

An extension of this research was conducted which trained a DLNN trader using a considerable amount of more in-depth market data, collected from millions of trades executed by multiple algorithmic trading strategies across thousands of varied market sessions. When re-implemented back in to a simulated exchange the DLNN could use the current market data to predict profitable prices at which to trade, resulting in an overall impressive performance. When evaluated against seven algorithmic traders by comparing their average profits, the DLNN trader was able to outperform the other traders on 11 occasions, match them on 2, and was only outperformed in one experiment - despite being trained on data collected these same seven traders.

This DLNN trader was named 'DeepTrader', and although it performed unexpectedly well, no in-depth analysis was carried out investigating how it was able to trade so profitably. Understanding deep learning networks is important because it is often valuable to be able to explain the decisions that they make, especially if they are used by an investment bank as a sales trader, executing trades using client's money. It is also beneficial to understand them so that they can be optimised and any unnecessary data, which may actually be damaging to their predictions, can be removed. Low level market data is also cheap, whereas in-depth market data is very expensive, so companies could save considerable money if it was revealed that a DLNN trader only needed basic, inexpensive market data to trade.

Therefore, this research conducted a preliminary investigation in to the success of DLNN traders, specifically DeepTrader, and can be divided in to two sections. The first section focused on replicating DeepTrader, but also experimenting and trying to optimise various elements throughout this process, and the second section focused on analysing which of the 13 features used to train DeepTrader were the most important, and which were redundant.

To summarise, the main contributions of this research were as follows:

- A successful replication of DeepTrader called DeepTrader2, and a detailed methodology of how DeepTrader2 was created.
- An in-depth analysis of DeepTrader2 across multiple market conditions, verifying that it could not only equal the original DeepTrader's performance but actually performed better.
- Results ranking the importance of each of DeepTrader's 13 features when training the model.
- An analysis revealing the surprising result that isolating the least or most important features in a trader didn't correspond to a statistically significant difference in performance.
- A conference paper, co-authored by Aaron Wray, myself, and Dave Cliff, accepted to be presented at the 2020 IEEE Symposium Series on Computational Intelligence (SSCI) (see Appendix B).

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
  - 1.1.1 Feature Analysis . . . 2
  - 1.1.2 DeepTrader Optimisation . . . 2
- 1.2 Project Approach . . . 2
  - 1.2.1 Aims and Objectives . . . 3
  - 1.2.2 Project Deliverables . . . 3

2 Contextual Background **5**
- 2.1 A Brief History of Algorithmic Trading . . . 5
- 2.2 Deep Learning in Finance . . . 6
  - 2.2.1 Deep Learning Applied to Automated Trading . . . 6
- 2.3 Analysis of Neural Networks . . . 7

3 Technical Background **9**
- 3.1 Financial Markets . . . 9
  - 3.1.1 The Limit Order Book . . . 9
  - 3.1.2 Market Equilibrium . . . 10
  - 3.1.3 Bristol Stock Exchange . . . 11
- 3.2 Existing Automated Trading Agents . . . 11
  - 3.2.1 Basic Strategies . . . 11
  - 3.2.2 ZIP . . . 12
  - 3.2.3 GDX . . . 12
  - 3.2.4 Adaptive Aggressive (AA) . . . 12
- 3.3 Overview of Deep Learning . . . 14
  - 3.3.1 Technical Overview . . . 14
  - 3.3.2 Types of Neural Networks . . . 15
  - 3.3.3 Training a DLNN . . . 15
- 3.4 DeepTrader . . . 16
  - 3.4.1 DeepTrader Features . . . 17
  - 3.4.2 Training DeepTrader . . . 18
  - 3.4.3 DeepTrader Performance . . . 18

4 Project Execution **21**
- 4.1 Replicating DeepTrader . . . 21
  - 4.1.1 Data Collection . . . 21
  - 4.1.2 Neural Network Architecture . . . 22
  - 4.1.3 Neural Network Tuning . . . 23
  - 4.1.4 BSE Experiments . . . 26
- 4.2 DeepTrader2 Evaluation . . . 29
  - 4.2.1 One-in-Many Tests . . . 29
  - 4.2.2 Balanced Group Tests . . . 30
  - 4.2.3 Overall Comparison . . . 30
- 4.3 Feature Importance . . . 30
  - 4.3.1 Permutation Importance . . . 30
  - 4.3.2 Initial Feature Importance Analysis . . . 31
- 4.4 DeepTrader2 Variations . . . 32

4.4.1 Training DeepTrader2$_{\text{Best6}}$ & DeepTrader2$_{\text{Worst7}}$ 32
4.4.2 BSE Experiments 33

5 Analysis of Results **35**
5.1 DeepTrader2 Analysis 35
5.1.1 DeepTrader2 vs DeepTrader 35
5.1.2 Varying Market Conditions 38
5.2 Feature Variation Analysis 40
5.2.1 One in Many Tests 40
5.2.2 Balanced Group Tests 41
5.2.3 Kruskal-Wallis Tests 43

6 Conclusion **45**
6.1 Contributions 45
6.2 Future Work 45
6.3 Final Evaluation 46

A DeepTrader Code **51**

B Conference Paper **53**

Acknowledgements

Firstly, I would like to thank my supervisor Dave Cliff for his advice and guidance throughout this project. I would also like to thank my parents for supporting me both emotionally and financially throughout this Master's degree, and my girlfriend Georgia for her encouragement, her valuable insight, and her constant support.

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 [24][12][22]. 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][23], and over 80% of Forex transactions in 2016 [3].

Over half of the world's financial markets, including Hong Kong, NYSE, Tokyo, and LSE [13], are based on a Limit Order Book (LOB) system. This is essentially an anonymised ledger containing the price and quantity of all the current buy and sell orders submitted by the traders in a market, alongside a record of all previous transactions. On these LOB-based markets a significant amount of information is available, which algorithmic traders can utilise to try to quantify and analyse the current market state, allowing them to calculate a profitable price at which to trade.

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 could be observed and its behaviour learned, then the same could be achieved with a human trader. An extension of this work by Wray (2020) used a wider range of market data, collected from multiple traders in tens of thousands of market sessions, to train his own deep learning trader called DeepTrader [33]. DeepTrader used this data to predict profitable prices at which to submit orders to the exchange, and was able to outperform, or match the performance of, all but one other trading algorithm in the experiments carried out.

Despite the success of DeepTrader and its predecessors, there has been no in-depth analysis aimed at identifying the reasons why these traders were successful. Neural networks (NN) are very abstract tools for modelling complex relationships, and so it is not directly obvious which 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 main aim of this project was to begin an investigation in to the underlying mechanisms which contributed 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 would identify the most important features needed for trader success, as well as the redundant ones, allowing future iterations of DLNN traders to be improved further.

1.1 Motivation

1.1.1 Feature Analysis

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 an easy 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 main motivation behind this work because as machine learning becomes more commonplace, it is important to try and understand why these networks work the way they do.

Discovering which market features are influencing the trading price also has multiple other benefits. Previous work by Le Calvez only used basic data available at the top of the LOB [5], also called 'level 1' data, while DeepTrader used a wider range of market data to try to capture its entire state, so it will be valuable to understand whether more detailed data 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 information deeper in the LOB is actually redundant, this can be beneficial for trading companies because often it isn't readily available and costs a lot to acquire; therefore using only level 1 data would save companies 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 they can complicate the relationships that the model is trying to find between the inputs and the output, damaging its predictive capability. A retrospective analysis of prediction accuracy when using various information from a limit-order book has shown that only 22% of the information used for price prediction is contained deeper than level 1 data [6], with most orders submitted to the LOB beyond level 1 actually ending in cancellations (and therefore not very useful) [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. Consequently, identifying and removing these misleading 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 necessary to understand feature importance.

1.1.2 DeepTrader Optimisation

The secondary motivation for this thesis was similar to the motivation for the original work by Wray, because part of this research aimed to explore different ways in which to optimise the original DeepTrader. From reviewing its design and implementation I believed there were some modifications which could improve DeepTrader's performance, and so this research also investigated these because delivering the best trader possible would provide an even better proof of concept. Refining the process used to develop DeepTrader and documenting these experiments produced an even more in depth schematic for creating a deep learning trader, providing a platform for future research to build on both this and Wray's work further.

1.2 Project Approach

This project was split into two main aims, each with its own execution and analysis:

- Replicating the work reported by Wray in 2020 [33] and producing my own version of DeepTrader with at least comparable performance, called DeepTrader2.

- Performing feature importance analysis on DeepTrader2, then running experiments in Bristol Stock Exchange (BSE - a minimal simulation of a real-world exchange, see Section 3.1.3) with variations of DeepTrader2 which only used a subset of the original features.

The replication of DeepTrader followed the same process outlined by Wray in his original implementation although with several design modifications, some of which improved performance. For example, the original DeepTrader used all trades from every trader as training data, but isolating and training on only the trades from the best traders actually lead to better performance. While it was not a primary aim to refine the original DeepTrader, there was value in recording the different design decisions made in case of any future work looking to further improve on DeepTrader2. The same evaluation techniques as Wray were also used to quantify and compare DeepTrader2 to the original, enabling the first aim to be completed.

The feature importance analysis consisted of using feature perturbation to determine which of the 13 input metrics had the most effect on the model's predictions. The features with the highest importance were isolated and used to create a 'best' trader, whilst the features with the lowest importance were used to create a 'worst' trader. These were then evaluated in experiments in BSE to attempt to confirm the initial importance findings. The expected results from these experiments were that if the correct features had been isolated, then there shouldn't be any statistical difference in performance between the best and the original trader, possibly even an improved performance from the best trader. At the same time, it was expected that the worst trader would trade significantly worse because it didn't have access to the same market data as the other two traders.

1.2.1 Aims and Objectives

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

1. Reconfigure BSE and automate data generation to produce the data required to train DeepTrader2.
2. Create and train a neural network with an architecture similar to Wray [33], 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. Perform a confidence interval analysis to quantify DeepTrader2's performance.
5. Repeat steps 1 - 4, with modifications at any step, until DeepTrader2 can produce a comparable performance to DeepTrader.

The second aim was divided in to the following objectives:

1. Perform feature analysis methods on the DeepTrader2 neural network to determine which market metrics contributed the most to training/test loss, and which were redundant.
2. Create variations of DeepTrader2 using the most and least important features, then reimplement these back in to BSE.
3. Run experiments across multiple market conditions and configurations to collect a variety of data.
4. Conduct a statistical analysis to quantify the effect of isolating different subsets of features.

1.2.2 Project Deliverables

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

- Validation and verification of the success of Wray's work in creating DeepTrader.
- DeepTrader2 - An overall improved version of DeepTrader which could outperform every other trader in the same experiments carried out by Wray.
- A comprehensive evaluation of DeepTrader2 across multiple different market conditions.
- Results and conclusions from a feature analysis, which provided an initial insight in to the importance of the features used for training DeepTrader2.

Chapter 2

Contextual Background

2.1 A Brief History of Algorithmic Trading

The trading strategies discussed here are the ones which were used to generate the market data used to replicate DeepTrader, and are also the ones it was 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 two '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 theoretical equilibrium price for that market in the same way that human traders did. The market equilibrium is the point at which the quantity demanded is equal to the quantity supplied, and so there is no need for the price of the traded asset to change. This work 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 experimental design, and that if the supply/demand profiles were altered, the ZI-C traders no longer converged to the theoretical equilibrium [10]. 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 [29].

Subsequent market simulations showed that these MGD trading agents could outperform all other current algorithmic traders (ZIP, ZI-C, GD), with ZIP placing 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 [24].

Following these results, another extension of GD was published in 2002 by Tesauro and Bredin called GDX [28], 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 [32][12]. AA held its title for over 10 years, however recent studies have begun to cast doubt on its versatility, suggesting that it can be beaten by simpler strategies in certain market conditions, including more realistic conditions. For example, a 2015 MSc Thesis by Vach [31] 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 (see Section 3.2.1 for details) [18]. Further studies completed by Snashall and Cliff in 2019 showed, by exhaustive testing through millions of market simulations, that AA in fact does not dominate when more realistic market conditions are used, being outperformed on several occasions by more basic strategies [27][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 [22], 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 strategies to aim to outperform were 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 expertise from years of experience. 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, and if the network contains three or more layers of neurons it becomes a deep learning neural network (DLNN) [20]. Similarly to algorithmic trading, deep learning gained popularity as computational power became more affordable, with recent noteworthy achievements by companies like DeepMind [26] - more technical background on deep learning can be found in Section 3.3.

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) [21], 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, financial time series forecasting isn't applicable to a 'live' trading environment because instead of predicting future transaction price trends based on a series of previous trades, a DLNN trader is trying to reproduce the behaviour of an existing trader in response to the current state of the market [5]. 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

A combination of price forecasting and algorithmic trading was explored by Niño *et al.* in which a deep neural network (DNN) trained on stock market data predicted the price variation for the next minute of activity [2]. This prediction then set the price limits for an algorithmic trading agent, allowing it to trade within those limits to ensure an overall profit. While this work employed both price forecasting using deep neural networks and algorithmic trading, the trader being used was still deriving its trade price using pre-defined algorithms instead of directly predicting the trade price, and so was not a true DLNN trader.

The first relevant publication for this area of research was the work conducted by Le Calvez in 2018 [5], in which he expanded upon a preliminary exploration of applying DLNNs to replicate ZIP traders in a 2017 Master's thesis by Tibrewal [30]. 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, but also when tested 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 was entirely plausible that it could also learn to replicate human traders, and possibly outperform them.

Although an impressive proof of concept, Le Calvez's approach was limited by the fact that the DLNN trader only considered the current best bid and ask prices when deciding whether to trade, disregarding other market metrics contained in the LOB. These other 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 [33], 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 a great 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 further it must first be understood how these traders have performed so successfully.

While the results produced by Wray's work were impressive, they also introduced several questions and avenues for further exploration. The first of these was the fact that DeepTrader trades very profitably despite using training data gathered from every trader regardless of profitability, meaning the average quality of the data it was trained on was relatively low. A neural network can only perform as well as the data it's trained on, and therefore it could be assumed that if the quality of the data improves, the network would 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 was likely considering its high variation even in favourable market conditions, and so this was another interesting avenue to pursue.

Arguably, the most interesting route was to investigate exactly why DeepTrader performs so well, and which features within the market it was 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 evident. However, there are methods of analysing neural networks to try and understand how they work, and this was 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 is going on, often being referred to as "black boxes", but there are ways of modifying the input then 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 focused on image classification problems, although it indicated that feature importance may change drastically even with small changes to the input, and so was something to keep in mind during this analysis.

None of the papers mentioned above ([2] [5] [33]) go on to explore the reasons why their work was so successful, and which 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 [34]. 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 focused on using LOB data to generate price predictions, and the inputs to the network were different, the results were not transferable to this research, although the method used to generate these results was.

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 was an important first step, an analysis of the network doesn't always correlate to the analysis of the subsequent trader. Therefore, in this work feature perturbation was used for an initial analysis, but once there was an indication of which features were important, multiple traders were created for experimentation on an exchange to determine whether these features really were the ones which contributed to trading ability.

Chapter 3

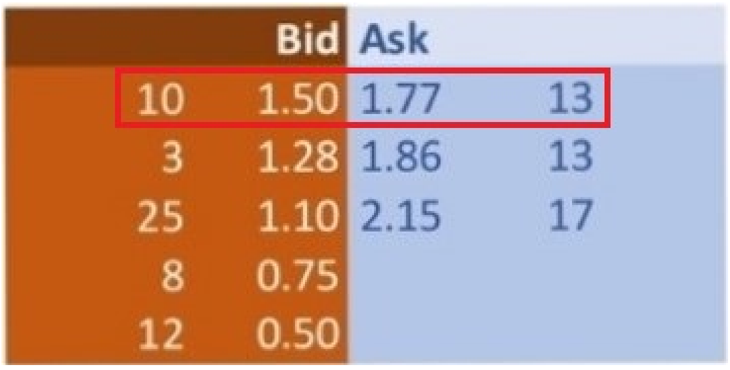
Technical Background

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 covers 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 asset. Customer limit orders are orders supplied to a trader containing the quantity, type (buy/sell), and the limit price, which is the price (specified by the customer) at which they would like to trade. The top of each side of the LOB records the current best (highest) order to buy, and the current best (lowest) order to sell, alongside the quantity, with the rest of the orders getting worse further down the book. 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, and also highlights the best bid and best ask at the top of their respective sides.



	Bid	Ask	
10	1.50	1.77	13
3	1.28	1.86	13
25	1.10	2.15	17
8	0.75		
12	0.50		

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

Sales traders receive limit orders from their customers and proceed to try and trade as optimally as possible to make a profit, while at the same time fulfilling their customer orders. 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 customer limit price specifies. Automated traders use the information shown in Figure 3.1, and their customer limit order, to try and determine a price to trade at which generates the most profit.

In 'level 1' market data, the only information available is the price and quantity of the best bid and best ask (highlighted in Figure 3.1), plus the price and quantity of the most recent transaction. In the work mentioned earlier by Le Calvez [5], only level 1 market data was used, whereas the work which followed by Wray [33] 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).

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

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

3.1.2 Market Equilibrium

The market equilibrium price is a useful variable for describing a market, defined as the point at which supply and demand are balanced, and represented by the green cross shown in Figure 3.2 where the supply and demand curves meet. Market equilibrium is significant because if traders are able to execute transactions at the equilibrium price, then this allocation of market assets is said to be Pareto efficient; "An allocation is Pareto efficient if no-one can be made better-off without someone else being made worse-off" [23]. Calculating this equilibrium price accurately is therefore beneficial because this is the point at which all traders are inclined to converge upon, and the price at which it is more likely that a transaction will occur without a considerable loss of profit.

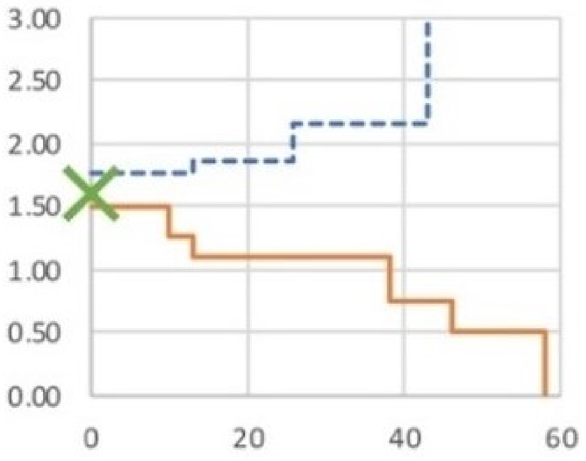


Figure 3.2: Supply/Demand curves for the LOB shown in Figure 3.1 (Image copied from [8])

A more useful variable than midprice 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 the equilibrium price should decrease, and will actually be lower than the midprice indicates. Alternatively, 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 more accurate indication of where the equilibrium price may be, using only the values at the top of the LOB.

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

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).

3.1.3 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]. All traders within BSE are sales traders, which means their limit prices are provided by external, randomly generated customer orders, as opposed to proprietary traders which can decide their prices autonomously.

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, although if a limit order crosses the spread and executes immediately it is defined as a market order. Real financial exchanges can allow many more 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. [30] [5] [9] [27] [33]), 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 [33] 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 [25].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 was formulated based around the idea that the market could provide an indication of the likelihood that an order 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 [29]. 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 [28]. 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 [32], although recently several studies have begun to question this claim [31] [18] [9] [27]. 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 [32], 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 an estimation of the market equilibrium, p^*. It is calculated using the moving average equation, shown in Equation 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 followed to create and replicate DeepTrader, a basic knowledge of deep learning is required. Therefore, an in-depth explanation of each aspect of this topic is beyond the scope of this work, and this section aims to provide an overview of the relevant details.

The field of machine learning (ML) relates to building models which have the ability to learn and improve through experience, but are not explicitly programmed to do so. This process involves observing data, making predictions based on that data, comparing these predictions to the actual values to calculate the accuracy of the model, and then automatically iterating upon and improving the model to maximise its accuracy. This is achieved by passing data through a series of interconnected nodes called a neural network, each with weighted connections between them, and once the model accuracy has been calculated these weights can be iteratively modified to try and produce more accurate predictions.

Deep learning neural networks are simply neural networks that contain more than two layers of interconnected nodes, the aim being to try and extract higher level features from the data. The structure of a neural network consists of an input layer followed by any number of hidden layers, for which the input and output for every node not observable, with the final layer referred to as the output layer. Machine learning is split in to supervised and unsupervised learning; supervised learning observes input data, then verifies its predictions using the expected values provided, whereas in unsupervised learning the model only has access to input data and is expected to extract features without comparing to a specified output. For this research, only supervised learning is considered.

3.3.1 Technical Overview

Activation Functions

Each node within a neural network uses an activation function to calculate its output based on the input (or set of inputs) that it receives. Every input will have a weight, w, associated with it which scales the input, and is the variable which is modified when the network tunes itself to improve its predictions. This can be described using Equation 3.9, in which the output of the neuron in layer b (y_b) is the result of the activation function (f) acting on the sum of the weights multiplied by the outputs of the previous layer (as shown in Figure 3.3).

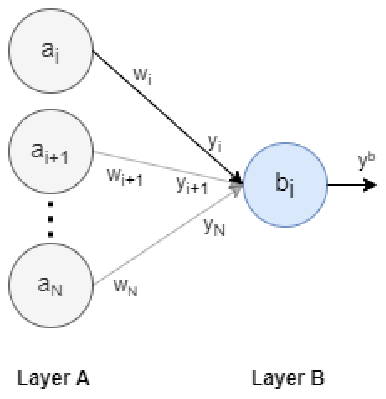


Figure 3.3: Diagram of the weighted inputs from a layer of a neural network feeding in to an output layer

$$y^b = f(\Sigma(w_i^a \cdot y_i^a)) \tag{3.9}$$

Backpropagation Learning

If input data was passed through a neural network and the resulting output matched exactly with the expected output, then the network doesn't need to be modified and works perfectly. However, this is never the case and the network needs to be able to adjust the weights on the connections between nodes so that the network prediction can match, or move closer to, the expected output. An approach for basic neural networks involves distributing the 'blame' for the bad prediction between the weights and modifying them accordingly, but in a DLNN there are multiple weights contributing to the input for each node, so a more complex method is required.

Backpropagation is a technique that divides the error proportionally between each weight for every input. Different deep learning approaches use different functions to calculate this error value, so for this example the mean squared error (MSE) is used to compare the model output (y_i) to the expected output ($\hat{y_i}$) (shown in Equation 3.10).

$$E = \frac{(\hat{y_i} - y_i)^2}{2} \tag{3.10}$$

This loss function is then differentiated with respect to every weight inside the network, allowing the weights to be updated depending on the local gradient of the loss function. The amount by which the weights are updated on every iteration can be tuned using the learning rate, η, of the network, which is known as a hyperparameter and can heavily influence model performance. Equation 3.11 shows the equation used to update the weights for each node k using the derivative of the error function with respect to each node, and using the learning rate to scale the result (for derivations and more detailed calculations see [20]).

$$\Delta w_k = \eta \frac{\partial E}{\partial w_k} \tag{3.11}$$

3.3.2 Types of Neural Networks

Various types of neural network have been developed for many different applications; convolutional neural networks (CNNs) for image recognition and classification, radial basis function neural networks used for power restoration systems, and recurrent neural networks (RNNs) used for speech conversion and price forecasting models. This research is focused on RNNs because this is the network utilised by Wray in his development of DeepTrader.

Recurrent Neural Networks

Recurrent neural networks operate on the principle of storing the outputs from previous inputs and processing them alongside the next input, simulating a memory of past events within the network which can influence future predictions. Using information from previous timesteps alongside current inputs makes RNNs a considerably complex system, and this complexity can cause problems when calculating the error gradients during backpropagation. This issue became known as the vanishing (or exploding) gradient problem, and was essentially an accumulation of all the errors from previous timesteps resulting in very small (or large) errors.

A solution to this problem was proposed in 1997 by Hochreiter and Schmidhuber when they introduced Long Short-Term Memory (LSTM) neural networks, which utilised a series of gates which ensured a constant error across each LSTM cell [19]. While conventional RNNs were unable to consider input from greater than 10 time steps ago, LSTM networks were able to bridge across time lags in excess of 1000 timesteps; this allowed long term time dependencies to be captured alongside the short term.

3.3.3 Training a DLNN

In practice, training a DLNN involves both pre-processing the data beforehand, and tuning the model parameters in just the right way to achieve the desired performance. Training a model refers to the process of passing it the input data, evaluating its outputs using a loss function, allowing it to automatically update its weights, and then repeating until it achieves an acceptable performance. The model parameters discussed here are: epochs and batches, learning and decay rate, loss functions, sample weighting, and dropout.

Epochs and Batches

Showing the model every sample in the training dataset is referred to as one epoch, and models are often trained for multiple epochs to expose them to as much data as possible. However, if trained for too many epochs there is a risk of overfitting the model to the training data, which means that it won't be able to perform with the same accuracy when exposed to new data. For this reason a separate set of validation data is used to verify that the model can still perform well on new data.

Batches are used to split the training data into more manageable pieces, which the network processes one at a time and after each batch the model updates its parameters. This is less computationally expensive because the loss function is computed on a subset of the data instead of the whole dataset, however using very small batch sizes risks converging on a non-optimal solution because the model has only considered a small amount of data.

Learning Rate

As mentioned above, the learning rate (η) of a model is the amount by which the network weights are updated after it has completed one epoch, shown in Equation 3.11. If the learning rate is set too high, the weights will change by too much each time the model is trained, causing the error to never converge to zero, but if it is set too low the training time can increase dramatically. Optimising the learning rate is a process of trial and error, and can be balanced with how many epochs the data is trained for.

Decay can be used in conjunction with learning rate, and is the amount by which the learning rate decreases at the end of each epoch. This allows for quick convergence at the start of training, and then for later epochs the model is able to narrow down its loss optimisation in smaller 'jumps' because the weights are being updated by a smaller amount, making it more likely that the loss will converge more closely to the true minimum. Again, tuning decay can be a trial and error process.

Loss Functions

As discussed earlier in Section 3.3.1 the error for each input is calculated by an error function, but the overall error for the model can be aggregated using a loss function. This quantifies how close the overall model is to its desired output, and is the metric by which it is evaluated at the end of each epoch. Similarly to how the error was calculated in Equation 3.10, the mean squared error can also be used as an overall measure of loss, shown in Equation 3.12.

$$Loss = \frac{1}{N}\Sigma_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.12}$$

Sample Weighting

The process of sample weighting involves passing a one-dimensional vector (the same length as the number of samples) to the network, which contains a value for every sample corresponding to how 'important' it is. When the model calculates its loss function, it uses the importance vector to weight each sample's loss, and so the more important samples will contribute more to the overall loss. Therefore, when trying to minimise loss the model will prioritise and try to match the expected outputs for the most important samples, fitting its predictions more closely to the important data.

Dropout

Dropout refers to the process of randomly removing a fraction of the training data at each epoch, and is often utilised when working with RNNs. This is a form of model regularisation, which attempts to generalise the predictions made by the model to achieve greater performance on new data and prevent overfitting. While the model has less data to train on, likely increasing the training loss, the trade off is that it will have better performance on the validation data.

3.4 DeepTrader

This section details the process followed in the formulation, training, and evaluation of DeepTrader, and explains the concepts behind several of its features.

3.4.1 DeepTrader Features

For the creation of DeepTrader, 13 market metrics were used to 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 was buying or selling an asset.
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 Equation 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 Equation 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 an asset.
8. Best Ask - the current best (lowest) offer on the LOB to sell an asset.
9. Time since last trade, Δt - the amount of time which had 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 3.13. This can be a strong indicator for price direction, as demonstrated by a 2015 paper [16], and so is a useful feature to include.

$$I = \frac{q_b - q_a}{q_b + q_a} \tag{3.13}$$

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 Section 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, the features 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 one of the advantages of BSE, the customer order prices for other traders were stored along with the other features.

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 each session, following an increasing sinusoidal function as shown in Figure 3.4, thereby exposing the traders to a range of different supply and demand curves. However, this only shifts the supply and demand curves up or down, changing the market equilibrium point without changing the shape of the curves, so this could limit the range of the data collected. Other methods may have been used to make the data more realistic however the full BSE configuration is not specified in Wray's research [33].

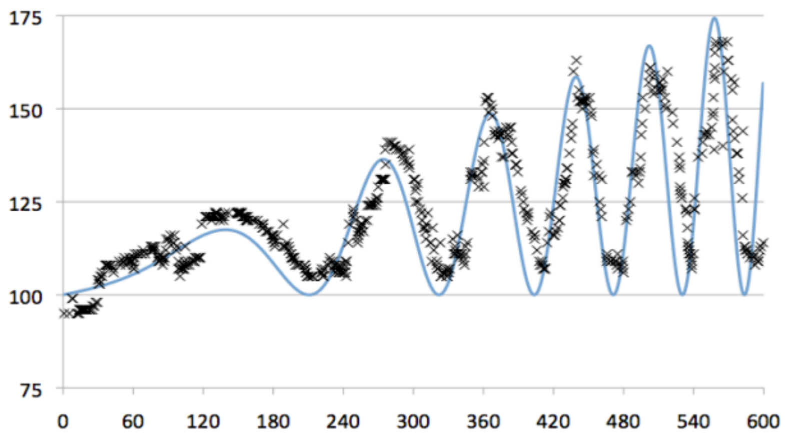


Figure 3.4: Function used to dynamically vary the supply and demand curves during the market session in BSE (Image taken from [8])

This data was then normalised and used to train a 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 3.5. 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 variables 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.* [34]. Regardless, DeepTrader produced impressive results and so will be replicated using the LSTM module, but future work could explore using sequential data to train the network.
The training configuration used by Wray was reported in full and so was straightforward to follow for the execution of this project (see [33] for full details), although some parameters were modified because they seemed to achieve better results (see Section 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

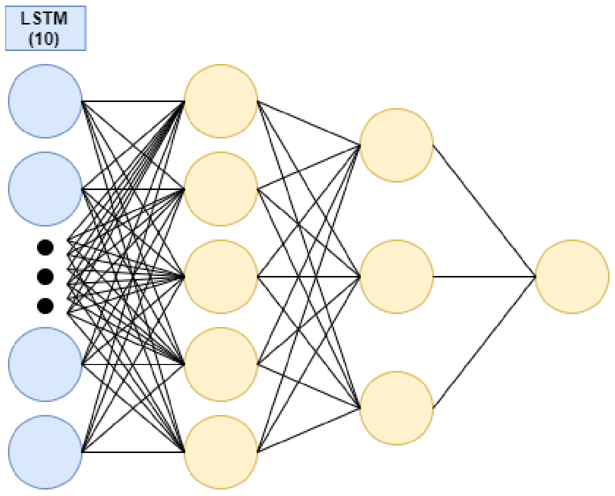


Figure 3.5: Diagram of the network architecture used by Wray to train DeepTrader.

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.

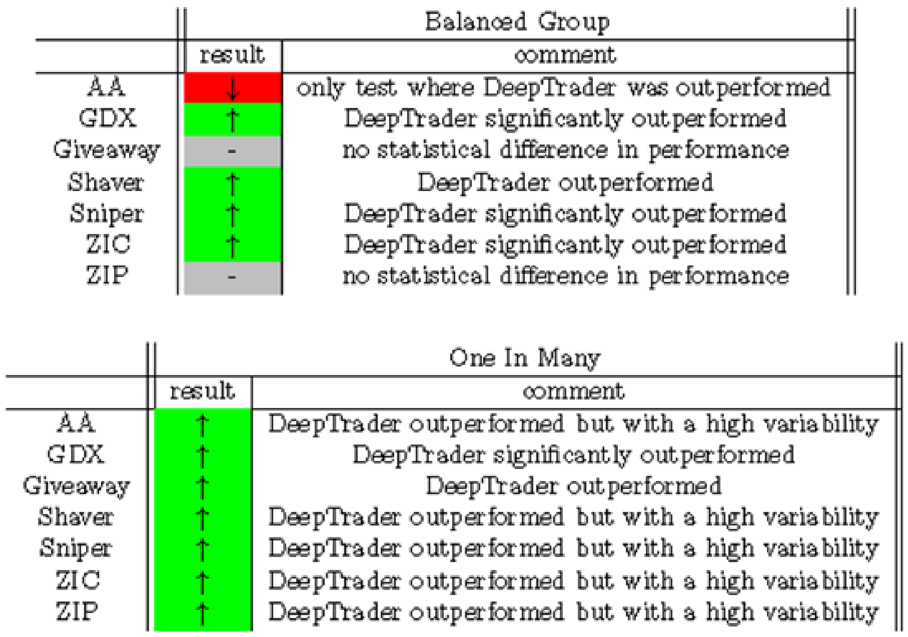
On real world exchanges the strategies of other traders and the market configuration are unknown, and therefore it is hard to draw an absolute comparison between two traders. Two types of experiment were carried out to directly measure DeepTrader's performance against each other trader: one-in-many tests, 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 and distributing balanced customer orders to each strategy, 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 provided a range of values for which it was 90% certain they contained the mean, and so if the lower bound of the first trader's range was above the upper bound of a second trader's range, it was 90% certain that the first trader had outperformed the second. Using this method took into account the variation of DeepTrader's performance across all market sessions, which was important given a high variability in its performance. 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.

$$c_{lower} = \bar{x} - \frac{Z^*_{0.9}\sigma}{\sqrt{n}} \qquad\qquad c_{upper} = \bar{x} + \frac{Z^*_{0.9}\sigma}{\sqrt{n}} \tag{3.14}$$

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, $Z^*_{0.9} = 1.645$.

A summary of these confidence interval results achieved by DeepTrader can be seen in Figure 3.6, 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 that DeepTrader could outperform, although doesn't indicate how much they are outperformed 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 3.6: Summary of the Wray's experimental results, [33]), comparing DeepTrader to all other traders in one-in-many and balanced group tests.

Chapter 4

Project Execution

The execution of this research project was divided into two distinct sections:

- The first component aimed to replicate Wray's work and re-create a version of DeepTrader which could attain similar performance to the original, called DeepTrader2 (DT2). This section is a re-implementation of existing work, but also contains 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 in BSE.

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 unsuccessful, 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 for each session - 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 in Section 3.3).

Once BSE was modified to rank the traders, 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 input contributed equally and to prevent any predictions being skewed by a certain feature, min-max normalisation was carried out on each input feature (using equation 4.1 below) to ensure all values were within 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 for this work was based on the specification in Wray's thesis [33], and is shown in Figure 3.5 above. It contained 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 created later which changed the number of features passed to the neural network, the input shape also changed accordingly.

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. In this case it resulted in marginally worse performance from DeepTrader2 when reimplemented back in to BSE, likely due to the fact that it only had 80% of the training data each epoch. 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. While including and excluding the dropout layer was experimented with, the amount of data randomly removed was not varied so this could potentially be experimented with.

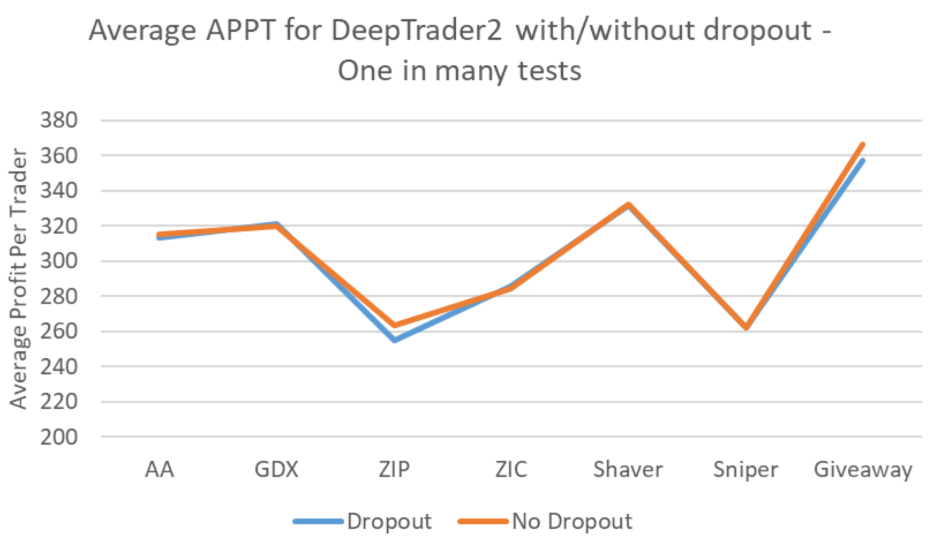


Figure 4.1: Average APPT for 60 one in many tests carried out for each trading strategy vs 2 variations of DeepTrader2, one with dropout and one without.

4.1.3 Neural Network Tuning

Ranking Traders

As mentioned above, during data collection each trader was ranked according to how profitable it had been within an individual trading session. This then translated to a weight which could be associated with every trade which that trader made during the session, allowing the trades with a greater weight to more heavily influence the predictions made by the neural network, in a process called sample weighting. 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 into BSE. This can be seen in Figure 4.2 where there is a distinct drop in performance when the DeepTrader2 variation trained on ranked traders is tested against AA, GDX, and ZIP in the one in many tests. While there is a slight improvement against Shaver and Sniper traders, it was clear from this analysis that weighting the samples did not create a more profitable trader.

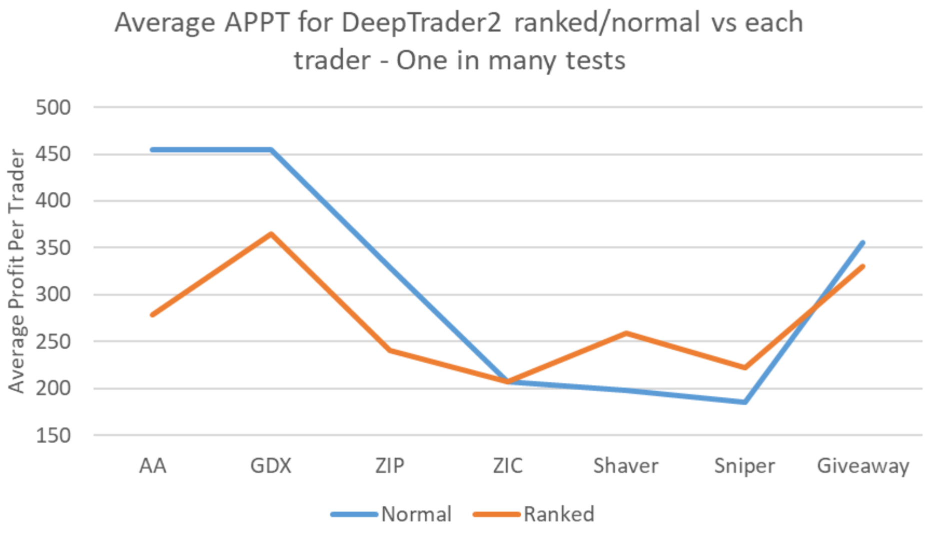


Figure 4.2: Results of 60 one in many tests carried out for each other trading strategy vs 2 variations of DeepTrader2 - one trained on weighted data (ranked), one trained on unweighted data (normal).

Overfitting Data

To avoid overfitting to the training data a validation set, comprising of 10% of the overall data collected, was used to test the model and ensure that it could still make accurate predictions on new data. Although the DLNN achieved a very low loss both on the testing and training data, once reimplemented into 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 slightly 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 Wray's 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 instead to determine an optimal training time. While Figure 4.3 doesn't show a distinct difference between 10 and 20 epochs, when compared later in different market conditions the effect was much more apparent, and so the final model was only trained for 10 epochs.

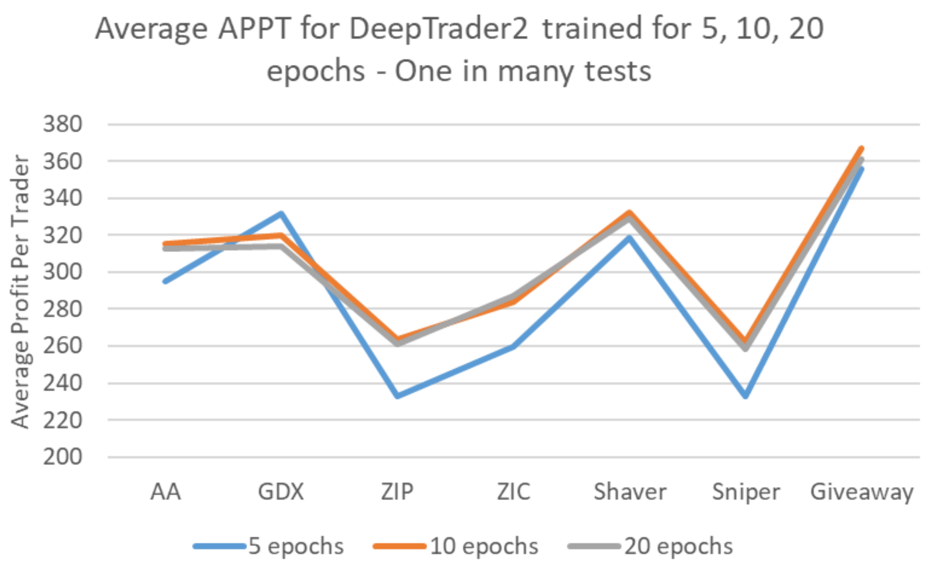


Figure 4.3: Average APPT for 60 one in many tests carried out for each trading strategy vs 3 variations of DeepTrader2, trained for 5, 10, and 20 epochs.

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. The results below show that this optimisation does translate to a slightly better trader performance when 'live' trading in BSE, and so it was used for the final DeepTrader2 configuration.

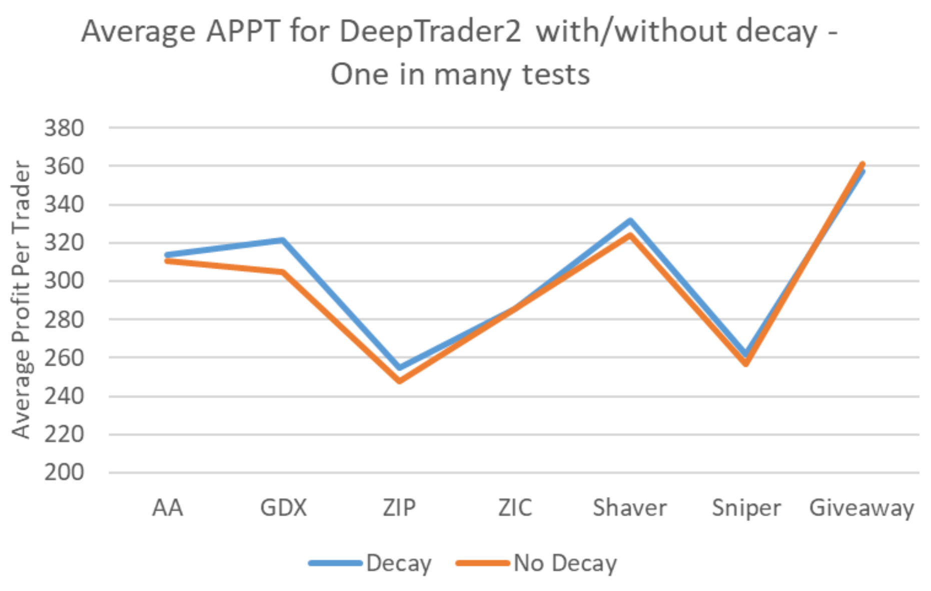


Figure 4.4: Average APPT for 60 one in many tests carried out for each trading strategy vs 2 variations of DeepTrader2; one with decay and one without.

Filtering Traders

When ranking traders and weighting their trades within the dataset proved to be unsuccessful, the next logical step was to filter the trading data based on performance, creating smaller datasets containing only traders with a certain ranking. This isolated only the best trades to train the network with, and while the dataset sizes did reduce as they became more exclusive it was a beneficial trade off as the trader performance noticeably improved, as shown below in Figure 4.5.

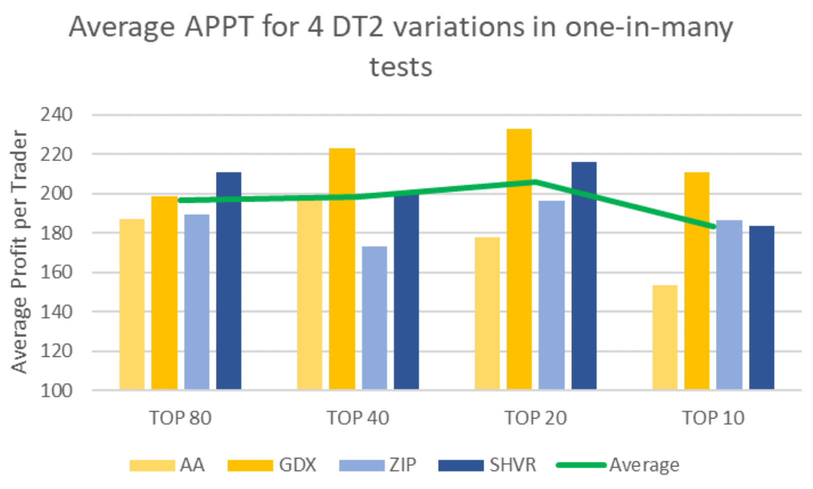


Figure 4.5: Graph showing the average APPT for 4 DeepTrader2 variations in one-in-many tests against 4 other traders. APPT averaged across 3 market sessions with different supply/demand curves

To produce Figure 4.5, the neural network was trained on trades from the top 80 (the entire dataset), top 40, top 20, and top 10 traders to create four variations of DeepTrader2. One-in-many tests were then carried out with each DeepTrader2 variation compared against AA, GDX, ZIP, and Shaver (SHVR) across three different market conditions - the average APPT across these three market conditions is the value plotted for each comparison above. In each situation, the traders outperformed almost every other trading strategy, with the top 40 and top 20 traders outperforming every other strategy in all three market conditions tested. The average APPT across all four traders was taken to determine which variation had the highest overall profitability, which, as indicated by the peak of the green line, was the variation trained on data collected from the top 20 traders.

Final Configuration

After exploring these different combinations of learning rate, epochs, decay, and data optimisation, 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
- Dataset filtered to include only the 20 best performing traders

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 evaluate the original 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.

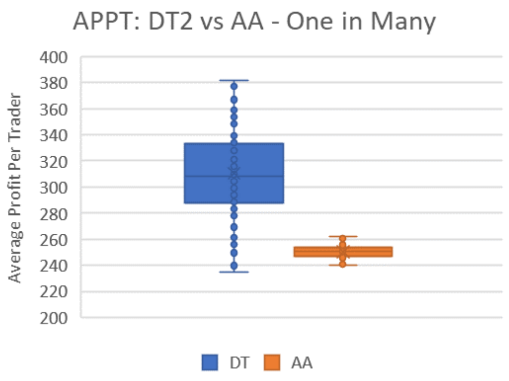


Figure 4.6: Box plot showing the average profit per trader (APPT) over 100 sessions: $2 \times$ DeepTrader2, $78 \times$ AA

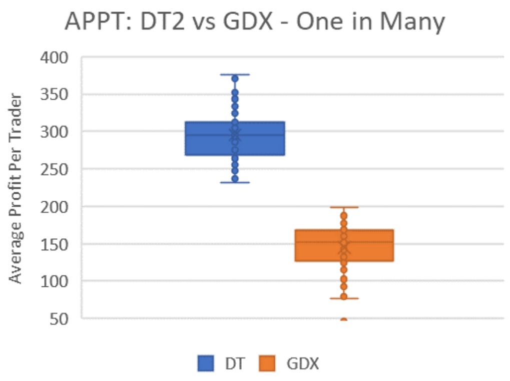


Figure 4.7: Box plot showing the average profit per trader (APPT) over 100 sessions: $2 \times$ DeepTrader2, $78 \times$ GDX

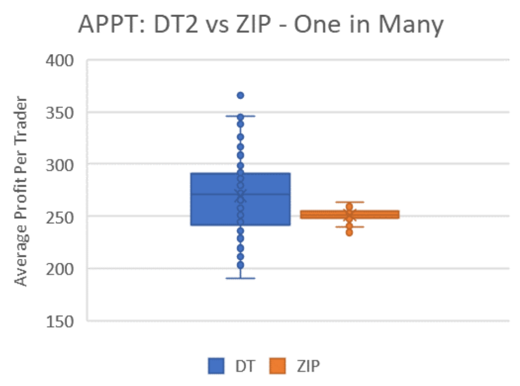


Figure 4.8: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × ZIP

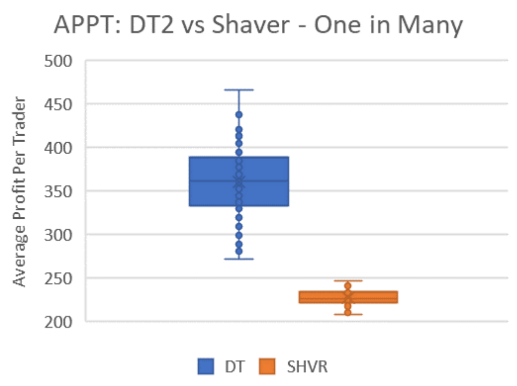


Figure 4.9: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Shaver

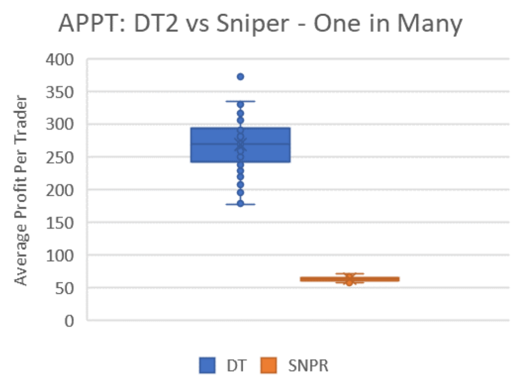


Figure 4.10: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Sniper

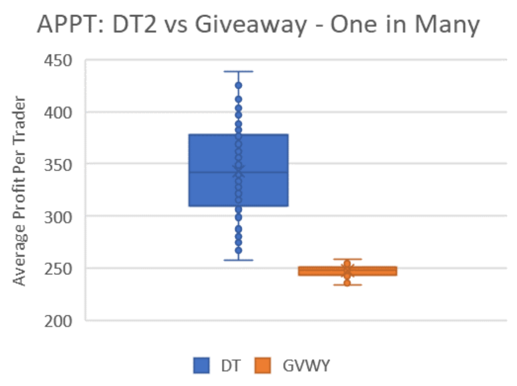


Figure 4.11: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × Giveaway

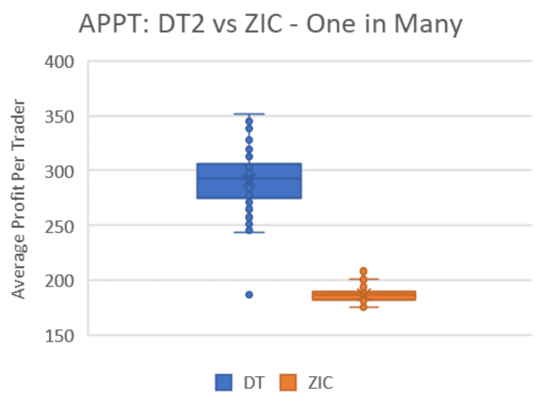


Figure 4.12: Box plot showing the average profit per trader (APPT) over 100 sessions: 2 × DeepTrader2, 78 × ZIC

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.

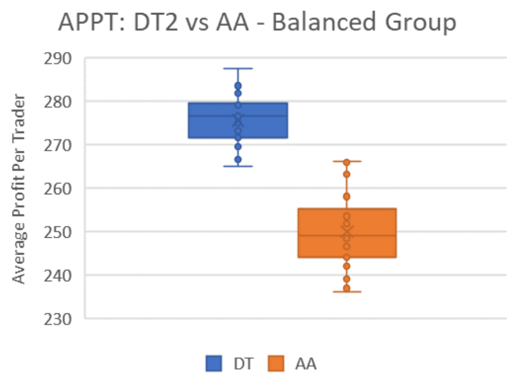


Figure 4.13: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × AA

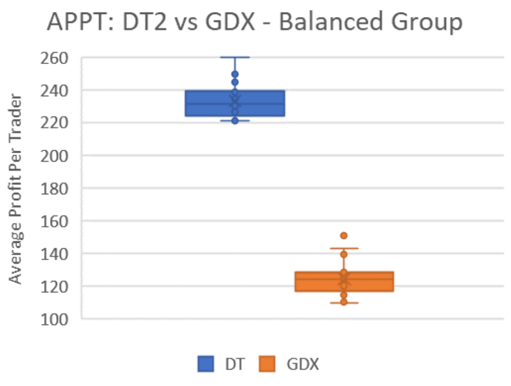


Figure 4.14: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × GDX

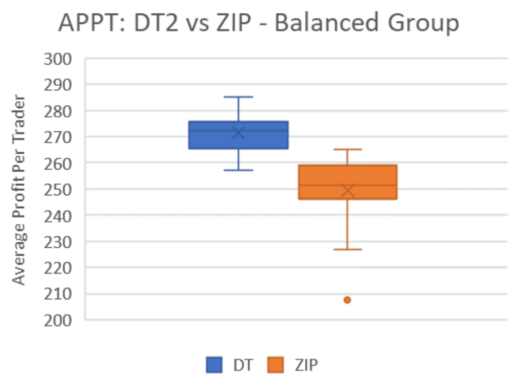


Figure 4.15: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × ZIP

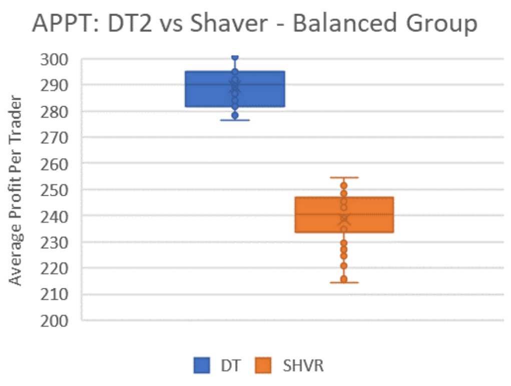


Figure 4.16: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Shaver

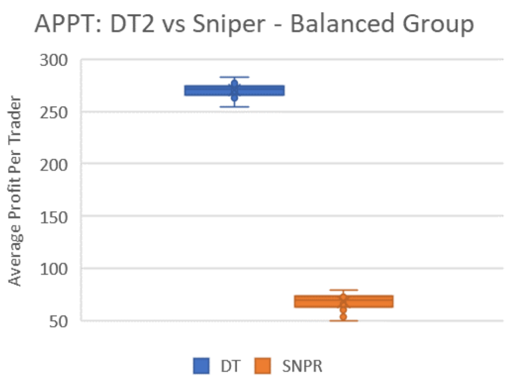


Figure 4.17: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Sniper

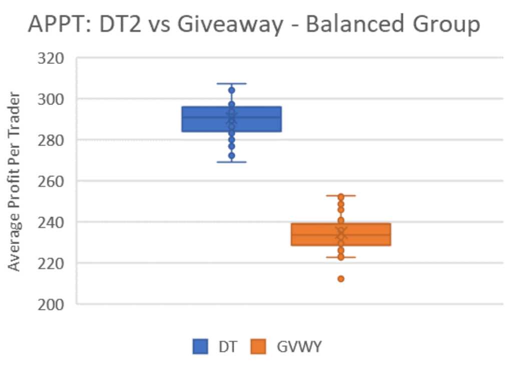


Figure 4.18: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × Giveaway

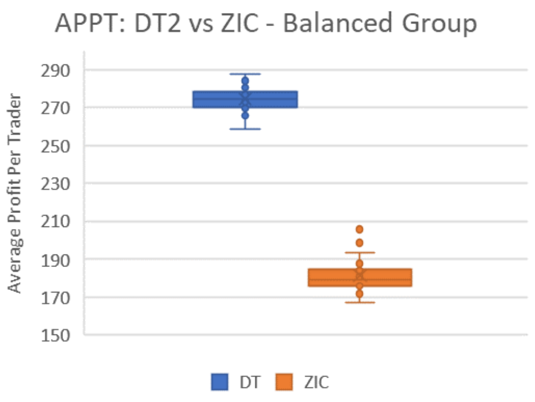


Figure 4.19: Box plot showing the average profit per trader (APPT) over 100 sessions: 40 × DeepTrader2, 40 × ZIC

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 Section 3.4.3). For this comparison however, a 99% confidence interval analysis was carried out ($Z = 2.576$), with the aim of further confirming DeepTrader's performance. When comparing two traders, 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 99% 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 99% 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	249.3	251.8	301.5	319.4
GDX	137.0	153.2	287.1	303.2
ZIP	250.1	252.8	260.2	279.1
Shaver	225.3	229.7	350.2	370.1
Sniper	63.1	64.8	259.3	278.0
ZIC	185.1	188.4	284.3	297.9
Giveaway	246.3	249.0	331.9	353.7

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 99% 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 DeepTrader2 successfully outperformed every other trader in all balanced tests, actually outperforming the original DeepTrader. Because of the time taken to run these tests, only 30 tests were run as opposed to Wray's analysis which ran 100 tests, however the confidence interval analysis takes this in to account when calculating its lower and upper bounds.

Traders	Trader X		DeepTrader2	
	c_{lower}	c_{upper}	c_{lower}	c_{upper}
AA	246.2	253.7	273.0	278.0
GDX	119.7	129.3	228.8	237.6
ZIP	242.8	255.8	268.1	274.9
Shaver	233.8	243.6	285.6	293.0
Sniper	64.7	71.5	267.8	273.2
ZIC	177.6	184.9	271.5	277.3
Giveaway	230.5	239.1	286.0	294.7

Table 4.2: Confidence interval analysis of DeepTrader2's average profit for 30 balanced tests.

4.2.3 Overall Comparison

Overall, DeepTrader2 performed well in all experiments, surpassing the performance demonstrated by the original DeepTrader. While both versions of DeepTrader outperformed all other trading strategies in the one-in-many tests, the original DeepTrader was unable to match DeepTrader2's performance in the balanced group tests against AA, Giveaway, and ZIP. A more comprehensive comparison of the two traders is explored in Section 5.1.

4.3 Feature Importance

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

4.3.1 Permutation Importance

Permutation importance was calculated once the network had been trained and involved systematically disrupting the input to each feature, essentially causing each feature to then compromise any predictions that the model tried to make. All 13 tests with 'corrupted' data should theoretically perform worse than the benchmark test with the original uncorrupted data, and by plotting how much worse each variation performed the importance of each feature could be inferred - larger disparities in loss meaning that feature had a greater impact on model performance, and smaller disparities meaning that feature didn't really affect model performance.

In these experiments, the final version of DeepTrader2 was evaluated against the validation dataset for 20 epochs to set a baseline loss value with which to compare against. Then, each feature in the validation dataset was randomly shuffled one at a time; the first column was shuffled, then the network was tested and its loss value recorded, then the first column was reset, the second column shuffled and the model was tested again, and repeat. The results from this experiment can be shown below in Figure 4.20, with the features showing the highest loss having the greatest impact on model performance.

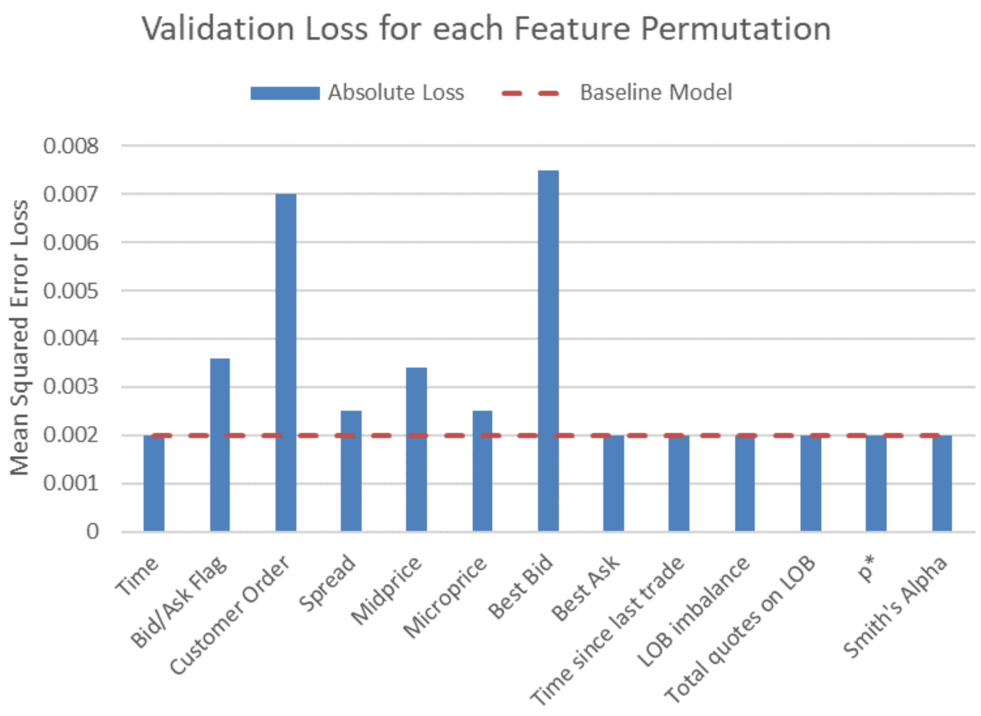


Figure 4.20: Loss values recorded from 13 tests on the validation dataset, each with a single feature's values randomised - baseline model included for comparison, which contained no randomised values.

From Figure 4.20 it can be seen that the best bid feature had the most significant impact on prediction accuracy, with customer order price 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 each other feature was randomised there was no perceived change in loss, with the final loss value being the same as the baseline model, and therefore individually they each had no effect on the predictions that the model made.

4.3.2 Initial Feature Importance Analysis

From the results shown in Figure 4.20, Table 4.3 was produced to rank each feature, showing the most important down to the joint least important features.

The significance of several of these features was fairly intuitive, although for other features it was 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 indicated 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

Rank	Validation Loss	Feature Removed
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: DeepTrader2 training features ranked in order of importance.

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 was 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 DeepTrader2 Variations

Further to this individual feature analysis, the next step was to consider if the six features (ranks 1 - 5 in Table 4.3) which seemed to actually influence the model predictions were all that was necessary for the model to perform well. This would also confirm that the seven other features (rank 6 in Table 4.3), which appeared to have no bearing on the predictive capability of the model, were indeed irrelevant. This involved creating, training, and testing two more versions of DeepTrader2; DeepTrader2$_{\text{Best6}}$, containing only the top 6 features detailed above, and DeepTrader2$_{\text{Worst7}}$, containing only the bottom 7 features.

4.4.1 Training DeepTrader2$_{\text{Best6}}$ & DeepTrader2$_{\text{Worst7}}$

The two variations of DeepTrader2 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. For the results in Table 4.4 the validation loss is higher than the loss shown in Figure 4.20. This is because the models were only trained for 10 epochs, instead of the 20 epoch tests used to demonstrate feature importance, in line with the final configuration outlined in Section 4.1.3.

Trader	Features	Training Loss	Validation Loss
DeepTrader	All Features	0.0043	0.0058
DeepTrader$_{\text{Best6}}$	Ranks 1-5 in Table 4.3	0.0041	0.0056
DeepTrader$_{\text{Worst7}}$	Rank 6 in Table 4.3	0.0059	0.0076

Table 4.4: Training and validation loss results from training each DeepTrader2 variation for 10 epochs

As can be seen in Table 4.4, the original DeepTrader2 model and DeepTrader2$_{\text{Best6}}$ achieved similar training loss, with DeepTrader2$_{\text{Best6}}$ even marginally outperforming the original on both training and validation datasets. DeepTrader2$_{\text{Worst7}}$ performed measurably worse than both other DeepTrader2 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 suggests that both DeepTrader2 and DeepTrader2$_{\text{Best6}}$ will achieve a similar performance, whereas DeepTrader2$_{\text{Worst7}}$ can be expected to perform much worse.

4.4.2 BSE Experiments

In the same way as DeepTrader2, the two variations were reimplemented back in to BSE to trade live against the other trading strategies. To evaluate their performance, one-in-many and balanced group tests were carried out, however for this comparison I was only interested in the relative performance of each DeepTrader2 variation and not the overall profit. Therefore, only these values have been plotted, but they have also been normalised so that comparisons could also be drawn across different market conditions.

Every APPT value from each experiment was divided by the maximum APPT recorded in those market conditions, which acted as an estimate for the optimal allocative efficiency of that market. Therefore, measuring each trader's performance relative to this gave an indication of their market efficiency in each experiment. The expected results from these comparisons were that DeepTrader2 and DeepTrader2$_{\text{Best6}}$ would achieve a similar performance, since they were both using the most useful market features to determine trade price. In contrast, DeepTrader2$_{\text{Worst7}}$ was expected to perform significantly worse in each market condition.

One-in-Many Tests

For the one-in-many tests shown in Figure 4.21, the same market conditions were used as in Section 4.1. From visually comparing the three DeepTrader2 variations, it was not clear which trader performed the best overall; DeepTrader2$_{\text{Best6}}$ performed the best against GDX, ZIP, and Giveaway, but DeepTrader2$_{\text{Worst7}}$ actually outperformed the other two against AA and ZIC. The expectation was that each set of box plots would look similar to the three Sniper plots, with DeepTrader2 and DeepTrader2$_{\text{Best6}}$ relatively similar and DeepTrader2$_{\text{Worst7}}$ significantly worse, but this was not consistent across all traders.

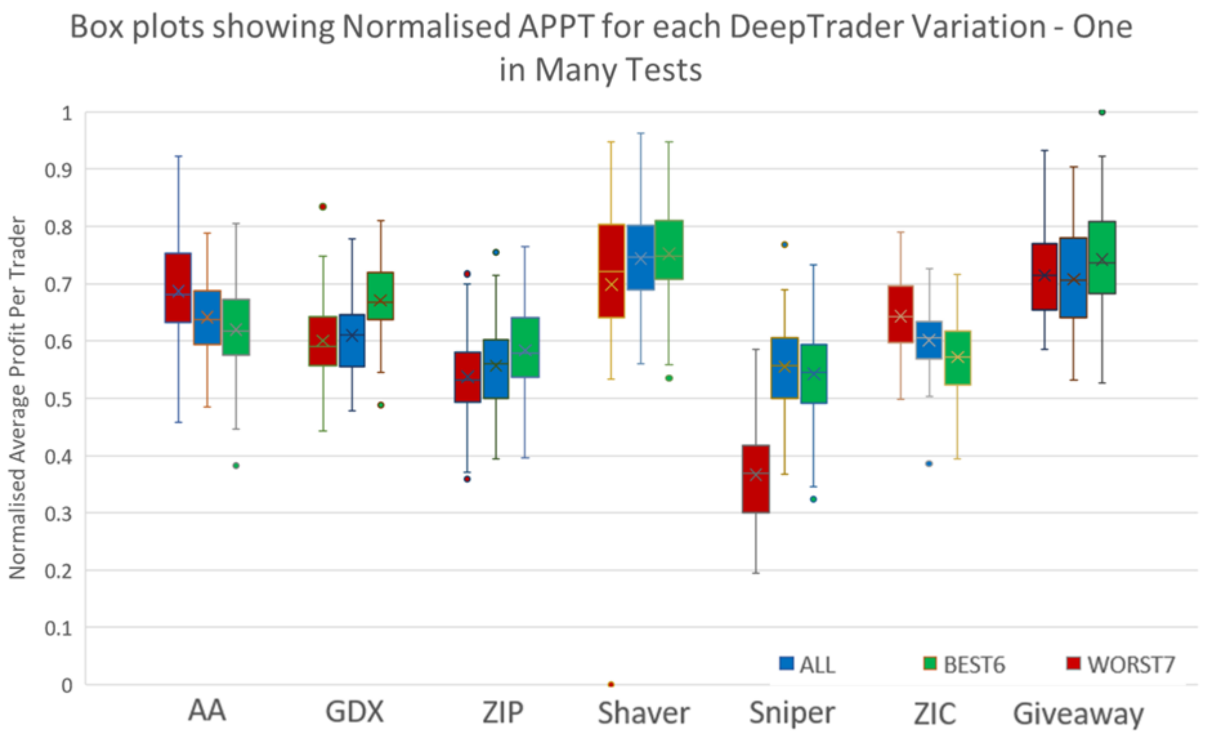


Figure 4.21: Box plots showing the normalised APPT for each DeepTrader2 variation vs each other trading strategy (30 market sessions for each one in many tests)

Balanced Group Tests

Due to time constraints, the DeepTrader2 variations were only compared against four of the seven trading strategies in the balanced group tests. However, this still provided a good indication of ability because the four best performing strategies were chosen. The balanced tests provided slightly more insight - it was fairly clear than DeepTrader2$_{\text{Best6}}$ was the superior trader in markets populated entirely with GDX traders, however this was not consistent across other traders. When tested against AA, DeepTrader2$_{\text{Worst7}}$ was the slightly better performing strategy, although in all other experiments it performed measurably worse than the other two (more in line with expectation).

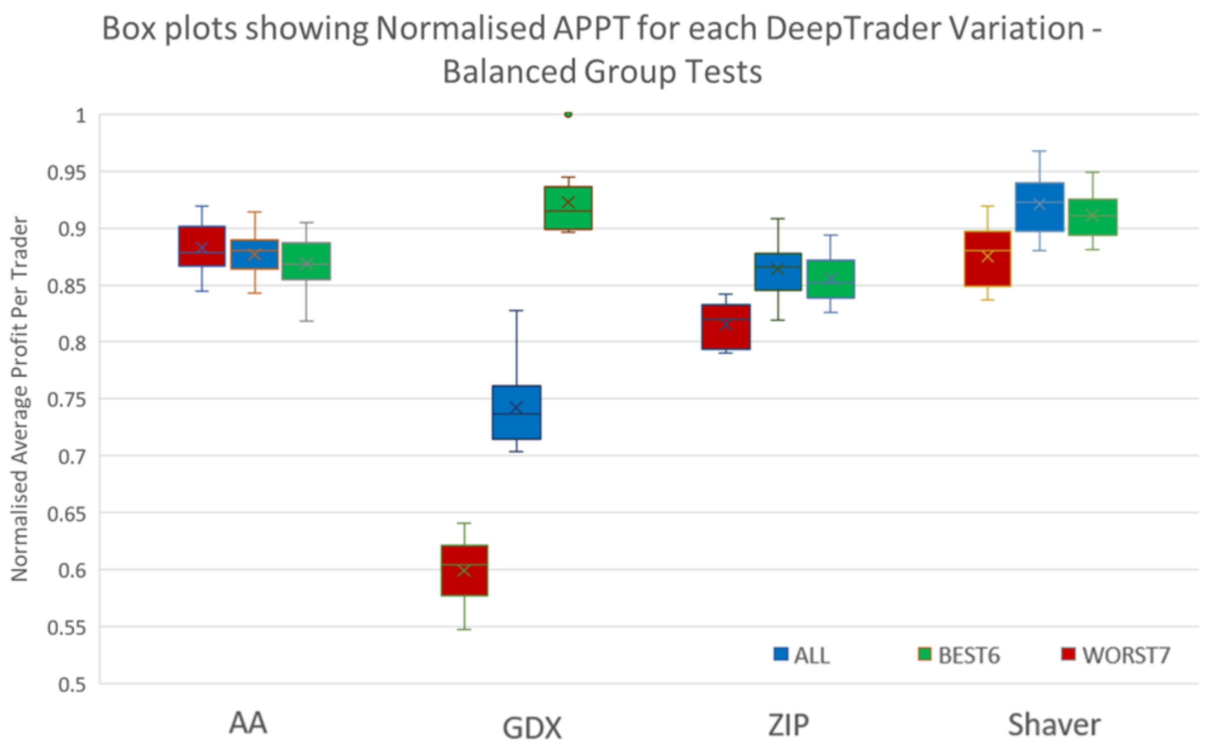


Figure 4.22: Box plots showing the normalised APPT for each DeepTrader2 variation vs 4 other trading strategies (10 market sessions for each balanced)

Figures 4.21 and 4.22 provided an initial insight in to the different performances, and while they revealed similar trends in the data they were by no means conclusive. Section 5.2 provides a more statistical and in-depth analysis of their performance across several market conditions.

Chapter 5

Analysis of Results

5.1 DeepTrader2 Analysis

5.1.1 DeepTrader2 vs DeepTrader

When comparing the absolute average profit achieved by DeepTrader and DeepTrader2, both sets of values varied by a considerable margin, especially for the one-in-many tests. This was likely caused by the different market conditions used when the results were recorded because supply and demand curves greatly affect the profit that traders can extract from the market. This difference was apparent when the APPT of the seven known trading strategies were compared because each of these experiments was populated with 97.5% known trading strategies, and only 2.5% DeepTrader traders. Therefore it could be assumed that the DeepTrader traders didn't have a large impact on the APPT of the other strategy, and so if the market conditions had been the same the APPT of each strategy would have been similar.

This assumption was actually verified by the tests run in Section 4.1.3 - when each DeepTrader2 variation was compared in one-in-many tests in constant market conditions, the comparison trader profit didn't vary by more than 1%. In the original DeepTrader experiments the average profit for the comparison trading strategies was between 10-20% higher than for those same strategies in the DeepTrader2 experiments, suggesting very different market conditions. Therefore, in order to compare the performances of DeepTrader and DeepTrader2, each trader's respective APPT values needed to be normalised.

One in Many Tests

In the one-in-many tests, this normalisation was calculated by dividing each of the profits for DeepTrader by the highest value (460.1), which was the upper bound for DeepTrader vs Shaver, and dividing each of the profits for DeepTrader2 by 366.5, which was the upper bound for DeepTrader2 vs Shaver. The results of this can be seen in Table 5.1, with each comparison colour-coded to indicate which trader achieved the highest APPT. However, this wasn't an entirely fair comparison because everything was measured relative to the upper bound of each DeepTrader version vs Shaver - if DeepTrader had performed exceptionally well against Shaver, this would cause the other values to appear relatively lower than DeepTrader2 and give the illusion that it had performed worse. Therefore, the only case where it could be confidently said that DeepTrader2 outperformed DeepTrader is against Sniper, in which not only were the normalised profit values are higher but DeepTrader2's absolute average profit was also higher.

This table does provide a good measure of each traders variance, which is shown in the range column, and in this case represents the range of values for which it is 90% confident that the mean lies. This translates to how much variability there is in the traders performance, which ideally should be minimised to ensure a consistently profitable trader. Therefore, the occasions where the range is smaller have been highlighted green, and it can be seen that DeepTrader2 was able to achieve a much smaller variation in its performance than DeepTrader in all but one test. Also, it should be noted here that 90% confidence intervals were used so that these comparisons could be made, as opposed to Tables 4.1 and 4.2 which used 99% confidence intervals.

Traders	DeepTrader			DeepTrader2		
	c_{lower}	c_{upper}	Range	c_{lower}	c_{upper}	Range
AA	0.852	0.905	0.053	0.831	0.863	0.032
GDX	0.772	0.824	0.052	0.791	0.819	0.028
ZIP	0.670	0.715	0.045	0.719	0.752	0.033
Shaver	0.952	1.000	0.048	0.965	1.000	0.035
Sniper	0.183	0.206	0.023	0.717	0.749	0.032
ZIC	0.776	0.815	0.039	0.782	0.806	0.024
Giveaway	0.847	0.896	0.049	0.916	0.954	0.038

Table 5.1: Normalised confidence interval analysis of DeepTrader2 vs the original DeepTrader in one in many tests

As another way of comparing strategies, the difference in APPT was calculated between each DeepTrader variation and the comparison trading strategies, and was presented as a percentage of the comparison trader's overall profit. This provided a standardised measure of how much each DeepTrader strategy outperformed, or was outperformed by, each other trading strategy, and can be seen below in Table 5.2. This comparison was also fairer because each calculation was independent of market conditions, instead standardising each trader's performance within their individual experiments and then comparing those standardised results.

Traders	DeepTrader				DeepTrader2			
	DT APPT	Trader APPT	Diff.	Diff. (%)	DT2 APPT	Trader APPT	Diff.	Diff. (%)
AA	404.2	301.2	103.0	34.2	310.5	250.5	59.9	23.9
GDX	367.3	123.5	243.8	197.4	295.2	145.1	150.1	103.4
ZIP	318.5	277.4	41.1	14.8	269.6	251.4	18.2	7.2
Shaver	449.1	270.5	178.6	66.0	360.1	227.5	132.6	58.3
Sniper	89.71	74.3	15.4	20.8	268.6	63.9	204.7	320.1
ZIC	366.1	224.1	142.0	63.4	291.1	186.7	104.4	55.9
Giveaway	400.9	301.9	99.0	32.8	342.8	247.6	95.2	38.4

Table 5.2: Percentage difference in average APPT between DeepTrader (DT) and DeepTrader2 (DT2) vs each other trading strategy in one in many tests

The results shown in Table 5.2 have been colour-coded to provide a more visual representation of each trader's performance. An arbitrary threshold of 10% was used to determine whether one version of DeepTrader had outperformed the other, with green cells indicating a 10% higher profit was achieved, and red cells indicating a 10% lower profit - grey means the difference was within the 10% threshold and therefore their performance was considered statistically comparable. Using the first row as an example, DeepTrader achieved 103 more profit than AA, and so its APPT was 34.2% greater than AA's. By comparison, on average DeepTrader2 achieved 59.9 more profit than AA, which was only 23.9% more than AA, and therefore it was concluded that DeepTrader had performed better than DeepTrader2 in one-in-many tests against the AA trading strategy.

Table 5.2 confirmed the earlier conclusion that DeepTrader2 outperformed DeepTrader by a significant amount when tested against Sniper, but it also revealed that there was a statistically significant difference in performance when compared against AA and GDX. Because of the 10% threshold the other results were not considered statistically comparable, although if they were included then DeepTrader would have outperformed DeepTrader2 in five of the seven experiments. Therefore, whilst Table 5.1 indicated that DeepTrader2 achieved more consistent results, overall it was concluded that DeepTrader had performed slightly better overall when measuring by how much each trader had dominated the comparison trading strategies.

Balanced Group Tests

When considering the balanced group tests, DeepTrader outperformed most trading strategies but was only able to equal ZIP and Giveaway, and performed worse than AA. In comparison, DeepTrader2 outperformed every other strategy in the balanced group tests, as shown above in Table 4.2. Using the same confidence interval normalisation as Table 5.1 to compare DeepTrader and DeepTrader2 produced Table 5.3, where DeepTrader was normalised using the upper bound for the DeepTrader vs Shaver experiment (413.6), and DeepTrader2 was normalised using the upper bound for the DeepTrader2 vs Giveaway experiment (293.1).

Traders	DeepTrader			DeepTrader2		
	c_{lower}	c_{upper}	Range	c_{lower}	c_{upper}	Range
AA	0.595	0.609	0.014	0.935	0.945	0.010
GDX	0.676	0.700	0.024	0.786	0.805	0.019
ZIP	0.535	0.650	0.115	0.919	0.934	0.015
Shaver	0.983	1.000	0.017	0.979	0.995	0.016
Sniper	0.588	0.602	0.014	0.917	0.929	0.012
ZIC	0.255	0.261	0.006	0.930	0.942	0.012
Giveaway	0.550	0.610	0.06	0.981	1.000	0.019

Table 5.3: Normalised confidence interval analysis of DeepTrader2 vs the original DeepTrader in balanced group tests

Again, the range values for DeepTrader2 were measurably smaller, suggesting a more repeatable performance than DeepTrader in the balanced group tests as well. Whilst this comparison was useful to see the variation in each strategy's performance, for DeepTrader the upper and lower bounds were skewed by the high profit achieved when tested against the Shaver trading strategy. DeepTrader2 had much more consistent results across each experiment, and so the normalised values appeared to be higher. Therefore, the same analysis as Table 5.2 was carried out to find the percentage difference in average profit between each variation of DeepTrader, shown in Table 5.4.

Traders	DeepTrader				DeepTrader2			
	DT APPT	Trader APPT	Diff.	Diff. (%)	DT2 APPT	Trader APPT	Diff.	Diff. (%)
AA	249.1	281.7	−32.6	−11.6	275.5	250.0	25.5	10.2
GDX	284.8	116.9	167.9	143.6	233.2	124.5	108.7	87.3
ZIP	245.0	263.1	−18.1	−6.9	271.5	249.3	55.5	8.9
Shaver	410.1	400.5	9.6	2.4	289.3	238.7	50.6	21.2
Sniper	246.0	82.6	163.4	197.8	270.5	68.1	202.4	297.2
ZIC	106.7	75.4	31.3	41.5	274.4	181.1	93.3	51.5
Giveaway	240.0	234.1	5.9	2.5	290.3	234.8	55.5	23.6

Table 5.4: Percentage difference in average APPT between DeepTrader (DT) and DeepTrader2 (DT2) vs each other trading strategy in balanced group tests

Table 5.4 has been colour-coded in the same way as Table 5.2 using a 10% threshold. In the balanced tests, while DeepTrader acquired a much greater absolute profit when tested against Shaver than DeepTrader2, Shaver also managed to achieve a high average APPT. This meant that, relatively, DeepTrader was a less dominant strategy in this experiment and each trader's high absolute APPT was likely an artefact of the market conditions. As shown in Table 5.4 this only translated to a 2.3% difference between DeepTrader and Shaver, which indicated that DeepTrader was only a marginally better trader than Shaver.

It was clear from this analysis that DeepTrader2 could outperform each trading strategy by a greater margin than DeepTrader, the only exception being for DeepTrader vs GDX in which it outperformed GDX by a considerable amount more than DeepTrader2. This was consistent across both the one-in-many and balanced tests, indicating that DeepTrader was more profitable than DeepTrader2 in a market purely populated by DeepTrader and GDX agents. However, it was still concluded that DeepTrader2 performed better overall in the balanced group tests.

To summarise, while DeepTrader performed marginally better in the one-in-many tests, DeepTrader2 proved to be a considerably more profitable trading agent in 6 of the 7 balanced group tests. DeepTrader2 also demonstrated statistically comparable or greater performance across all but 2 of the 14 experiments, and managed to outperform the comparison strategy in every experiment - contrary to DeepTrader which only succeeded in 11 of the 14 experiments. The reasons behind this improved performance could be due to any of the small modifications explored in Section 4.1.3, but it was most likely caused by the data filtering to include trades from only the top 20 performing traders.

5.1.2 Varying Market Conditions

In addition to the comparison with the original DeepTrader, DeepTrader2 was also evaluated in several different market conditions to further assess its versatility. This progressed upon Wray's work because the supply and demand schedules used in his original experimental setup were never stated, and therefore it wasn't clear whether DeepTrader could actually perform outside of those unknown experimental conditions.

The best way to compare performance across market conditions was to compare the relative difference in profit, in the same way as Table 5.4. The market conditions tested were defined by their supply and demand ranges, which in BSE were the ranges of prices from which customer orders were randomly generated and distributed to the traders. The ranges used can be seen in Tables 5.5 and 5.6, and they were chosen to model the following market conditions:

- C1: Lower supply prices than demand prices (used for all previous experiments).
- C2: Even supply and demand across a wide range of prices.
- C3: Slightly lower supply prices than demand prices.
- C4: Slightly higher supply prices than the demand prices.
- C5: Smaller range of supply prices than demand prices.
- C6: Larger range of supply prices than demand prices.
- C7: Even supply and demand across a more restricted range of prices.

One in Many Tests

In the one-in-many tests, an arbitrary threshold of 10% was chosen to determine whether DeepTrader2 had outperformed each other trader, in which case the cell was coloured green. Cells were also coloured lighter green for the range of 5% to 10% to indicate a less statistically significant performance, grey was used for the range -5% to 5% to represent no statistical difference in performance, and red was used for anything below 5%. Ideally a confidence interval analysis would be carried out for each comparison, but due to time constraints only a limited number of tests were able to be performed for which a confidence interval analysis is less accurate.

	Supply Range		Demand Range		APPT Difference (%)						
	Min	*Max*	*Min*	*Max*	AA	GDX	ZIP	Shaver	Sniper	ZIC	Giveaway
C1	75	95	100	120	19.3	50.8	6.7	36.8	76.7	36.9	27.6
C2	50	150	50	150	9.7	43.9	10.3	3.6	67.8	23.1	23.0
C3	90	110	100	120	22.3	76.8	13.2	24.6	72.5	42.0	37.1
C4	110	130	100	120	−13.8	42.4	20.7	32.4	62.0	46.9	28.7
C5	100	120	90	130	8.8	65.1	9.7	10.6	70.1	41.8	24.9
C6	90	130	100	120	20.1	82.6	14.5	20.6	68.5	46.3	28.2
C7	100	120	100	120	20.2	72.5	15.5	26.0	69.5	45.3	37.0

Table 5.5: Percentage difference in average APPT for DeepTrader2 vs each other trading strategy - 30 one in many tests performed for each of the 7 different market conditions, defined by their supply and demand ranges.

Table 5.5 demonstrates that DeepTrader2 could consistently outperform the other 7 trading strategies in almost all market conditions. The main exception was for condition 4 (C4) when tested against AA, in which DeepTrader2 performed significantly worse. This was likely because C4 was the furthest supply and demand profile possible from that of the dataset that DeepTrader2 was trained on (C1), and therefore AA was better able to adapt to the different market conditions. However this does not mean that DeepTrader2 was unable to adapt, because in these same conditions against the other traders it was still able to perform exceptionally well.

Interestingly, DeepTrader2's performance against Shaver was significantly impacted when a large even range of supply and demand prices were used (C2), but there was no similar effect when a smaller, even range of supply and demand prices were used (C7). However, this could be an anomaly due to the small number of tests carried out, but there did seem to be a large drop in relative profits between C7 and C2.

Balanced Group Tests

For the balanced group tests shown in Table 5.6, the arbitrary threshold was changed to 5% to determine whether DeepTrader2 had outperformed the other trader. This was because the balanced group results were averaged across 40 agents per session over 10 sessions, whereas the one-in-many tests only used 2 agents per session over 30 sessions, meaning the balanced group results had less variation. Also, from comparing Tables 5.1 and 5.3, the balanced tests had a much lower range and the results were much more consistent, therefore the threshold could be lowered.

	Supply Range		Demand Range		APPT Difference (%)						
	Min	*Max*	*Min*	*Max*	AA	GDX	ZIP	Shaver	Sniper	ZIC	Giveaway
C1	75	95	100	120	9.3	46.6	8.2	17.5	74.8	33.9	19.1
C2	50	150	50	150	5.8	32.8	4.1	0.2	76.7	31.4	15.2
C3	90	110	100	120	2.5	59.6	0.9	4.6	69.1	37.9	14.7
C4	110	130	100	120	−17.0	58.5	−0.9	3.8	48.6	36.7	5.3
C5	100	120	90	130	−2.7	61.4	1.7	2.5	64.6	36.0	7.0
C6	90	130	100	120	−2.6	65.2	−2.7	1.0	67.0	38.4	6.2
C7	100	120	100	120	−0.6	70.9	1.8	6.6	63.0	38.2	9.8

Table 5.6: Percentage difference in average APPT for DeepTrader2 vs each other trading strategy - 10 balanced group tests performed for each of the 7 different market conditions.

In Table 5.6 it was clear that DeepTrader2's performance was not as consistent across all the market conditions as it was for the one-in-many tests. For the experiments using GDX, Sniper, ZIC, and Giveaway, DeepTrader2 demonstrated slightly worse but still impressive results, and it could be concluded that DeepTrader2 is able to consistently outperform these strategies in various market conditions (for both one-in-many and balanced tests). For DeepTrader2 versus Shaver the results indicated that the strategies were either equal, or DeepTrader2 showed a slight advantage, although this was not conclusive. However, when compared to AA and ZIP it was obvious from these results that there was no advantage for DeepTrader2 when competing against them, potentially even losing to AA (especially in C4).

One observation from these results was that there was no single condition which DeepTrader2 consistently performed the worst in; while C4 showed poor results against AA, C6 showed poor results against ZIP and C2 resulted in the worst performance against Shaver. There were evidently less favourable market conditions for DeepTrader2, although no single worst condition against all other trading strategies. This indicated that DeepTrader2 could adapt to these different market conditions, and it was instead a matter of which other traders were present in the market which determined its performance.

In summary, when evaluated against each trading strategy in varied market conditions, DeepTrader2 still demonstrated a significantly better performance than 4 of the 7 trading strategies, and a marginally better performance than Shaver. Further analysis was definitely required, especially to explore other market conditions similar to C4 in which DeepTrader2 has a significant disadvantage. Although when taking in to account both the one-in-many and balanced group tests, it was concluded that overall DeepTrader2 could outperform 12 of the other trading strategies across the market conditions tested here, and also match the performance of ZIP and AA.

This conclusion actually correlated well with DeepTrader's initial performance because ZIP and AA were two of the strategies that it was unable to outperform. This suggested that DeepTrader2 was displaying similar behaviour, since it had the same strengths and weaknesses, and therefore it was an accurate replication. This also suggested that both ZIP and AA could exploit a weakness in the two traders that was inherent to the method used by both Wray and myself to create them.

5.2 Feature Variation Analysis

Following on from Section 4.4, this Section aims to provide a more in-depth analysis to try and clarify the earlier results. While it appeared there was no obviously superior trader, this was only explored for a single market condition, and so here several more market conditions were experimented with to see if there was a noticeable difference.

5.2.1 One in Many Tests

Whilst all seven trading strategies were compared in Figure 4.21, due to time constraints only four of the seven strategies were used for these experiments. To visualise the results, Table 5.7 was used which shows the average profit of each DeepTrader2 variation, but relative to the median of the three variations in that specific market condition. For example, for the three experiments where DeepTrader2, DeepTrader2$_{\text{Best6}}$, and DeepTrader2$_{\text{Worst7}}$ were tested against AA in market condition 1 (C1), DeepTrader2 achieved the median average profit of the three traders. DeepTrader2$_{\text{Best6}}$ performed 3.4% worse than DeepTrader2, and DeepTrader2$_{\text{Worst7}}$ performed 7.1% better.

Table 5.7 was also colour-coded with red and green to show whether a trader outperformed the other two, and grey to indicate that the values were not statistically different. An arbitrary 5% threshold was used to determine whether one trader had outperformed another, meaning that one trader had to achieve an average profit at least 5% higher than the median to be considered significant.

	AA			GDX			ZIP			Shaver		
	Worst7	All	Best6	Worst7	All	Best6	Worst7	All	Best6	Worst7	All	Best6
C1	7.1	0.0	−3.4	−1.6	0.0	10.0	−3.4	0.0	4.8	−2.2	0.0	1.1
C2	0.0	3.6	−5.0	−2.1	0.0	32.3	0.0	1.9	−5.4	−17.2	0.0	5.1
C3	−2.4	1.4	0.0	−1.0	0.0	3.9	−1.0	0.0	4.9	0.0	−1.3	2.1
C4	−27.7	0.0	17.0	0.0	1.7	−48.2	−1.6	17.9	0.0	0.0	2.8	−6.1
C5	−15.3	0.0	0.5	−0.1	0.0	21.1	2.5	0.0	−6.8	0.0	−5.9	10.9
C6	−14.6	9.3	0.0	−2.9	0.0	3.2	−1.6	7.5	0.0	−7.9	2.0	0.0
C7	−9.8	1.4	0.0	0.0	−0.7	12.2	−4.0	0.0	9.3	−1.9	0.0	0.8

Table 5.7: Percentage difference in average APPT for DeepTrader2 variations vs each other trading strategy across the 7 market conditions - One in many tests.

The results in Table 5.7 reinforced several findings from Figure 4.21. DeepTrader2$_{\text{Best6}}$ continued to show greater dominance in most market conditions against GDX than the other two variations, which both achieved very similar performance. Also, there was not much consistent variation within the ZIP and Shaver experiments, making it difficult to determine a clearly superior trader. In contradiction of Figure 4.21, which shows trader performance for market condition 1 (C1), DeepTrader2$_{\text{Worst7}}$ performed significantly worse than the other two traders against AA in over half of the other market conditions tested, and therefore actually had a much worse overall performance.

The most surprising result was that against GDX, ZIP, and Shaver in several market conditions, there was no statistically significant difference between DeepTrader2 and DeepTrader2$_{\text{Worst7}}$. This was unexpected considering the disadvantage that DeepTrader2$_{\text{Worst7}}$ had from the supposedly important features being removed. As a further analysis method, each DeepTrader2 variation's profit was averaged across all 7 market conditions against each trader to provide an measure of their overall performance. Whilst excessive averaging of the data loses a lot of information, this gave an idea of the magnitude of the differences in profit between the DeepTrader2 variations.

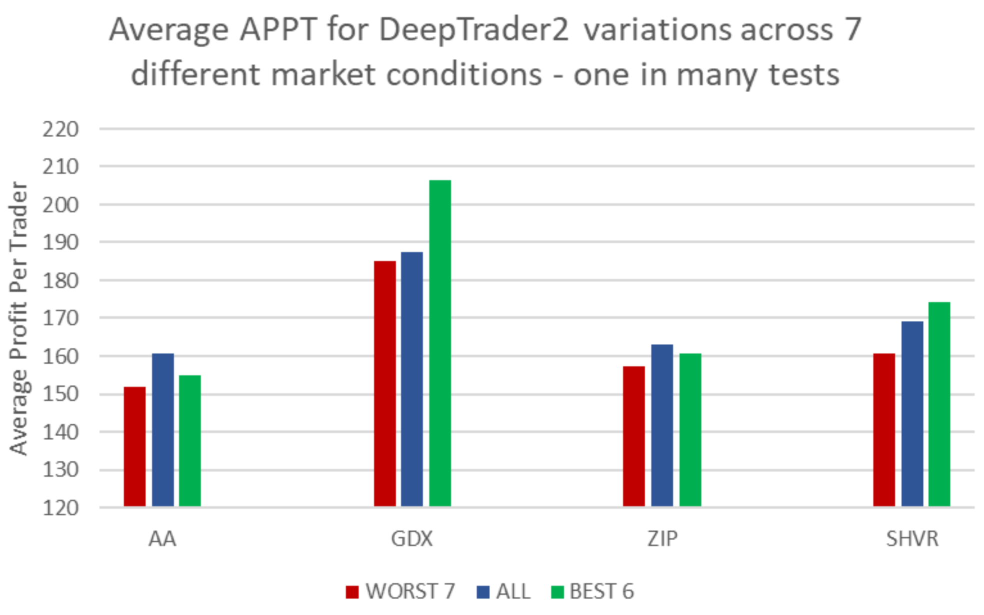


Figure 5.1: Average APPT for DeepTrader2, DeepTrader2$_{\text{Best6}}$, and DeepTrader2$_{\text{Worst7}}$ averaged across 28 one in many tests in 7 different market conditions.

From Figure 5.1, it became apparent that DeepTrader2$_{\text{Worst7}}$ had the worst overall performance against every trader, whilst DeepTrader2$_{\text{Best6}}$ alternated with DeepTrader2 for best performance. This correlated well with Table 5.7 because DeepTrader2$_{\text{Worst7}}$ had the most cells coloured red, and DeepTrader2$_{\text{Best6}}$ had the most cells coloured green, however the difference between average profits was still much less than expected. The only visually distinguishable difference was in DeepTrader2$_{\text{Best6}}$'s performance against GDX, which clearly outperformed the other two traders significantly. When the average profit of each trader was summed across all 28 experiments, DeepTrader2$_{\text{Best6}}$ achieved the highest profit, followed by DeepTrader2, then DeepTrader2$_{\text{Worst7}}$.

5.2.2 Balanced Group Tests

In the same way as the one-in-many tests, the balanced group test results were calculated and can be seen in Table 5.8. Because the variation across balanced group tests was generally lower due to the higher number of trading agents, the threshold was lowered from 5% to only 2.5% difference in average profit required to be considered significant.

	AA			GDX			ZIP			Shaver		
	Worst7	All	Best6	Worst7	All	Best6	Worst7	All	Best6	Worst7	All	Best6
C1	0.7	0.0	−0.9	−19.3	0.0	24.3	−4.7	1.0	0.0	−3.9	1.1	0.0
C2	0.0	2.4	−2.2	−26.4	0.0	9.3	−1.0	2.0	0.0	−3.5	8.5	0.0
C3	0.0	1.9	−0.6	−15.6	0.5	0.0	0.0	5.5	−0.6	−0.7	4.6	0.0
C4	0.0	15.7	−4.9	0.0	1.1	−28.0	0.0	6.8	−29.2	0.0	1.1	−21.3
C5	−8.8	2.4	0.0	−5.0	3.2	0.0	0.0	5.5	−4.7	0.0	4.7	−5.5
C6	−7.0	4.8	0.0	−2.8	4.1	0.0	0.0	0.0	−3.0	0.0	2.1	−3.4
C7	−3.5	4.9	0.0	−3.3	6.9	0.0	0.0	1.3	−10.3	0.0	2.3	−5.1

Table 5.8: Percentage difference in average APPT for DeepTrader2 variations vs each other trading strategy across 7 market conditions - Balanced Group tests.

When compared to the one-in-many tests, Table 5.8 revealed a considerably greater number of significantly worse performances by both DeepTrader2$_{\text{Best6}}$ and DeepTrader2$_{\text{Worst7}}$. DeepTrader2$_{\text{Best6}}$'s dominance over GDX essentially disappeared once other market conditions were tested, suggesting that the impressive results shown in Figure 4.22 were an artefact of the C1 market conditions. While DeepTrader2$_{\text{Best6}}$ showed a statistically similar performance to DeepTrader2 against ZIP and Shaver in the one-in-many tests, this also disappeared in the balanced group testing environment.

Perhaps the most noticeable drop in performance was from DeepTrader2$_{\text{Worst7}}$, which performed significantly worse in every market condition except one against GDX. When considering the fact that DeepTrader2$_{\text{Best6}}$ had access to considerably more useful market data than DeepTrader2$_{\text{Worst7}}$, the results against ZIP and Shaver were very unexpected. The original DeepTrader2 trader performed worse than the other two variations in only 2 of the 28 experiments, on all other occasions either matching or beating their performance. These results strongly indicated that DeepTrader2 was the best performing of the three, and that the two variations both couldn't perform in balanced group tests in different market conditions.

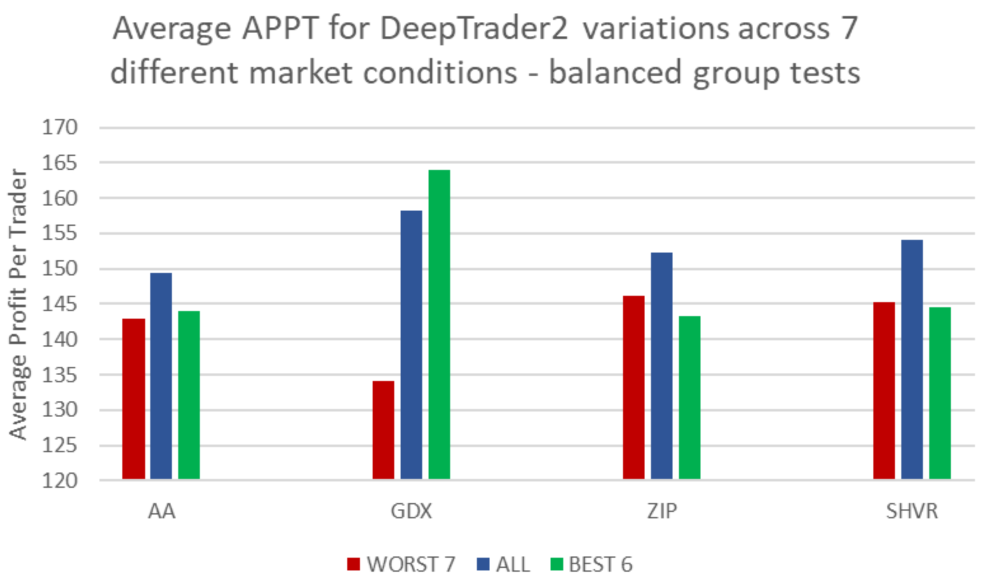


Figure 5.2: Average APPT for DeepTrader2, DeepTrader2$_{\text{Best6}}$, and DeepTrader2$_{\text{Worst7}}$ averaged across 28 balanced group tests in 7 different market conditions.

DeepTrader2$_{\text{Best6}}$ continued the trend of outperforming the other two traders against GDX despite its weak performance in C4, however this was not enough to compensate for its relatively low profit against the other 3 trading strategies. DeepTrader2$_{\text{Worst7}}$ continued to perform poorly, although still not as badly as expected from the hypothesis except in the GDX experiments. When the average profit from all 28 experiments was summed, DeepTrader2 achieved the highest, then DeepTrader2$_{\text{Best6}}$, then DeepTrader2$_{\text{Worst7}}$.

Overall, when both the one-in-many and balanced group tests were considered, there was still no definitive conclusion that could be drawn. The instinctive finding from this analysis was that the variation was not significant or consistent enough to confirm the original hypothesis. Overall, the one-in-many experiments demonstrated that DeepTrader2$_{\text{Best6}}$ achieved a marginally higher profit, followed by DeepTrader2, and then DeepTrader2$_{\text{Worst7}}$, while the balanced group tests showed that DeepTrader2 performed the best, followed by DeepTrader2$_{\text{Best6}}$ and then DeepTrader2$_{\text{Worst7}}$.

One interpretation of these results could be that, on average, DeepTrader2 and DeepTrader2$_{\text{Best6}}$ achieved the joint best performance, with DeepTrader2$_{\text{Worst7}}$ achieving the worst. This would verify the initial hypothesis that removing the worst features created a worse trader, but none of the analysis so far could produce statistically significant proof that this was the case. Therefore, to decide between these two contradicting conclusions, a Kruskal-Wallis test was carried out to test if these variations in performance were statistically significant.

5.2.3 Kruskal-Wallis Tests

The Kruskal-Wallis test is a non-parametric, ranking-based method of comparing multiple sets of values to determine the likelihood that they are sampled from the same distribution. This was applicable in this case because it could provide a way of determining whether there was a quantifiable difference in performance between the three trader variations. The requirements for this test are that the datasets being compared are categorical in nature, independent of each-other, and the dependent variable being measured is continuous, all of which this data fulfilled.

The first step in the Kruskal-Wallis method was to rank every value, independent of category, in ascending order. The test statistic, H, can then be calculated using Equation 5.1 and compared to a critical value H_c, which determines whether the null hypothesis should be rejected or not - the null hypothesis being that the groups of data were sampled from populations with a statistically similar median. In this equation, N refers to the total number of samples, n_k is the total number of samples in sample group k (where m is the total number of sample groups), and R_k is the sum of the ranks within sample group k.

$$H = \frac{12}{N(N-1)} \sum_{k=1}^{m} \frac{R_k^2}{n_k} - 3(N-1) \tag{5.1}$$

To carry out this comparison for DeepTrader2$_{\text{Best6}}$, DeepTrader2$_{\text{Worst7}}$, and DeepTrader2, the averaged profits for each trader from the 28 one-in-many experiments and the 28 balanced group experiments were used to calculate values for H. A significance level of $\alpha = 0.01$ was used, which meant that if the null hypothesis was accepted then there was a 99% chance that the samples were drawn from similar populations. Table 5.9 below shows the results of the two Kruskal-Wallis tests carried out on the results from the one-in-many tests and the balanced group tests.

Tests	H value	Result
One-in-Many	0.366	Hypothesis accepted
Balanced Group	0.5115	Hypothesis accepted

Table 5.9: Results from the Kruskal-Wallis test comparing the average profits across 7 market sessions for DeepTrader2, DeepTrader2$_{\text{Best6}}$, and DeepTrader2$_{\text{Worst7}}$

Since the hypothesis was accepted in both cases, it could finally be concluded that there was no statistical difference between the performance of the three DeepTrader2 variations. Whilst these results contradicted the original hypothesis that DeepTrader2$_{\text{Best6}}$ and DeepTrader2 would perform better than DeepTrader2$_{\text{Worst7}}$, this confirmation that the features used didn't have a statistically significant effect on the trader's performance was a valuable finding. Now that this has been established, further work could explore its ramifications for future DLNN traders, discussed in Section 6.2.

Chapter 6

Conclusion

6.1 Contributions

When compared against the original aims of this research, outlined in Section 1.2, it can be argued that both aims have been met but each to a different extent. The replication of DeepTrader was initially only a stepping stone so that the primary aim could be carried out, and while the experimentation within this work was intended, the improved performance offered by DeepTrader2 was an additional contribution of this work. Whilst the feature analysis produced results which were not as expected, the process followed was well documented and the finding was an interesting and valuable result. This provides a platform for future work to expand upon, and so the aim of conducting a preliminary investigation in to the feature importance is considered successfully fulfilled.

In addition to the main aims being achieved, this research also contributed valuable results and subsequent analysis - a full list of contributions is included below:

- A successful replication of DeepTrader was achieved along with an in depth comparison between DeepTrader and DeepTrader2, verifying that the two traders were comparable in performance.
  - This result validated Wray's original proof of concept because if DeepTrader can be independently reproduced, as it has been here, then its success couldn't be an artefact of his experimental setup.
- Further to this, a more in-depth analysis conducted across multiple different market conditions was carried out. This expanded upon the original analysis done by Wray (in which the market conditions were not specified or modified) providing evidence of DeepTrader2's superiority in various conditions.
- Feature importance results were presented which identified the features which contributed most to the accuracy of neural network predictions during training.
- Subsequent results demonstrated that using the 'best' or 'worst' features to create variations of DeepTrader didn't result in a difference in performance that was statistically significant.
  - Whilst these findings were unable to shed light on which features were important to produce profitable trade prices, it was an interesting result that which features were used was irrelevant.
- A well documented methodology for creating DeepTrader2, including decisions made and the rationale behind them, allowing any future replications to follow and expand upon this research efficiently - i.e. without wasting time on experiments which have already been tried.

6.2 Future Work

In addition to work which would build upon the contributions outlined in Section 6.1, this section also explores other paths which could have been taken over the course of this research, discovered during the project or retrospectively identified during the analysis.

Feature Isolation

Since it has become evident that the features used to train a DLNN trader aren't as sensitive as it was believed, a more extreme approach could be to isolate each feature individually, creating 13 different DeepTrader variations each with only a single feature input. This would overcome the issue of training loss not being a good indicator of trader performance, since they would all simply be evaluated in BSE. Then, using these results, features could be combined in different combinations to derive the best ones, and although this would involve a long, exhaustive test of different permutations, it may be the essential next step to try and understand how DeepTrader actually trades so successfully.

Exhaustive DeepTrader Analysis

While DeepTrader2 has been tested in several different market conditions, these were only a few arbitrarily chosen variations to try and introduce some variability. For DeepTrader to be considered as a superior trader, a full, exhaustive analysis needs to be conducted, which may still reveal that it doesn't perform well under all market conditions. In addition to this, various market configurations could also be tested since DeepTrader has only been subjected to one-in-many and balanced group tests so far. Whilst they are verified methods of evaluating trading strategies, they are not representative of real exchanges where the other traders could be any combination of other trading strategies. Therefore, an exhaustive analysis of DeepTrader's performance in different market configurations is also required to validate its status as a superior trading strategy.

Improved Training Data

While data pre-processing was carried out to try and improve the training data quality, work could be carried out to explore this further because this is a crucial part of training a successful model. Future versions of DeepTrader could employ the same filter so that only the top 25% of trades are used as training data, but the data could also be collected across multiple different market conditions to expose the DLNN to as much varied data as possible. Increasing the amount of data could also allow the filter to become more and more exclusive without losing model accuracy - for example if the amount of data collected was doubled, perhaps only the top 10 traders could be used to trade the model instead of the top 20. Another potential next step could also be to extract LOB data from traders on real exchanges and try to move towards a real-world implementation of DeepTrader, however collecting this data is often very expensive.

6.3 Final Evaluation

In conclusion this research has been largely successful, contributing further analysis of the existing work carried out by Wray, and then a subsequent investigation in to the dynamics of DeepTrader which produced unexpected and interesting results.

Whilst disappointing that the performance of the two DeepTrader2 variations (DeepTrader2$_{\text{Best6}}$ and DeepTrader2$_{\text{Worst7}}$) didn't verify the initial hypothesis, it was an interesting result and raises a lot more questions about what makes the DLNN trader successful. However, it would have been useful to be able to identify the market features which contribute to profitable trade prices. In retrospect, the loss value for DeepTrader2$_{\text{Worst7}}$ shown in Table 4.4 should have been significantly higher than the value that was recorded considering it had lost the best 6 features. This shows that the model was still able to derive relationships between the input and output, which then translated to being able to predict good trade prices regardless of the features used.

An important outcome from this was the fact that DeepTrader2$_{\text{Worst7}}$ was able to trade almost as profitably as DeepTrader2 despite not having access to its own customer orders. This was a very interesting result, because as discussed in Section 3.2 the other algorithmic traders relied on the customer order prices to calculate their trade prices. This could also have implications for future research involving real exchanges, because a significant limitation of Wray's work in other applications was that outside of BSE, it is impossible to know another trader's limit order. Therefore, DeepTrader couldn't be trained on data from a real world exchange, but if customer order prices aren't as vital as once thought, this could now be possible.

Another interesting finding was that while filtering the training data to only use trades from the top 25% of traders, the increase in performance wasn't as drastic as expected. The ability of the original DeepTrader to trade so well on sub-optimal data in the first place was surprising, but the fact that optimising that data only had a marginal effect on performance was even more surprising.

Another route which was explored but not discussed here was a feature variation analysis in which each of the 13 inputs to DeepTrader2 was fixed at either 0 or 1 at a time to see the effect it had on its performance. The aim of this was to confirm that the results from Section 4.3.1 translated to a similar change in trader performance. However, in practice, the variation was either non-existent, or so small that it was undetectable against the noise of the trader performance, but now this result makes sense after the results in Section 5.2.

This was very much a preliminary exploration in to the mechanisms behind the success of DeepTrader, and is hopefully the first of many iterations on Wray's original work. If I were to do this project again, I would have allocated more time to experimenting with the features used for DeepTrader2 instead of trying to further improve upon the original DeepTrader, which consumed a greater amount of time than was planned. However, this was still a valuable exercise because DeepTrader2 has expanded upon the original proof of concept and demonstrated its versatility through its impressive performance across different market conditions.

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. In *IJCAI International Joint Conference on Artificial Intelligence*, 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] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] Gerald Tesauro and Jonathan Bredin. Strategic sequential bidding in auctions using dynamic programming. In *Proceedings of the International Conference on Autonomous Agents*, pages 591–598, 01 2002.

[29] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, EC '01, page 206–209, New York, NY, USA, 2001. Association for Computing Machinery.

[30] 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.

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

[32] Perukrishnen Vytelingum. The Structure and Behaviour of the Continuous Double Auction, 2006. Ph.D. Thesis, University of Southampton, Southampton, UK.

[33] 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.

[34] 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

DeepTrader Code

```
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
```

Listing A.1: Code copied from Wray for the implementation of DeepTrader back in to BSE [33].

Appendix B

Conference Paper

Automated Creation of a High-Performing Algorithmic Trader via Deep Learning on Level-2 Limit Order Book Data

Aaron Wray
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
line 5: email address or ORCID

Matthew Meades
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
line 5: email address or ORCID

Dave Cliff
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
line 5: email address or ORCID

***Abstract*— We present results demonstrating that an appropriately configured deep learning neural network (DLNN) can automatically learn to be a high-performing algorithmic trading system, operating purely from training-data inputs generated by passive observation of an existing successful trader *T*. That is, we can point our black-box DLNN system at trader *T* and successfully have it learn from *T*'s trading activity, such that it trades at least as well as *T*. Our system, called *DeepTrader*, takes inputs derived from Level-2 market data, i.e. the market's *Limit Order Book* (LOB) or *Ladder* for a tradeable asset. Unusually, DeepTrader makes no explicit prediction of future prices. Instead, we train it purely on input-output pairs where in each pair the input is a snapshot *S* of Level-2 LOB data taken at the time when *T* issued a quote *Q* (i.e. a bid or an ask order) to the market; and DeepTrader's desired output is to produce *Q* when it is shown *S*. That is, we train our DLNN by showing it the LOB data *S* that *T* saw at the time when *T* issued quote *Q*, and in doing so our system comes to behave like *T*, acting as an algorithmic trader issuing specific quotes in response to specific LOB conditions. We train DeepTrader on large numbers of these *S/Q* snapshot/quote pairs, and then test it in a variety of market scenarios, evaluating it against other algorithmic trading systems in the public-domain literature, including two that have repeatedly been shown to outperform human traders. Our results demonstrate that DeepTrader learns to match or outperform such existing algorithmic trading systems. We analyse the successful DeepTrader network to identify what features it is relying on, and which features can be ignored. We propose that our methods can in principle create an explainable copy of an arbitrary trader *T* via "black-box" deep learning methods.**

Keywords—component, formatting, style, styling, insert *(key words)*

I. Introduction

The motivation for our work is best explained by a brief sketch of where we hope to end up, a little two-paragraph story of a plausible near-future:

Imagine a situation in which a highly skilled human trader operating in a major financial market has a device installed on her trading station, a small black box, with a single indicator lamp, that takes as input all data provided to the trader via her screen and audio/voice lines. The black box records a timestamped stream of all the market data that the trader is exposed to at her station, and also records a timestamped tape of all orders (quotes and cancellations) that she sends to the market: while it is doing this, the black box's indicator lamp glows orange, signaling that it is in Learning Mode.

After a while, maybe a few weeks, the indicator lamp on the black box switches from orange to green, signaling that it is now in Active Mode. At this point, the box starts to automatically and autonomously issue a stream of orders to the market, trading in the style of the human trader whose activity it has been monitoring. The box has learnt purely by observation of the inputs to the trader (market data and other information) and her outputs (various order-types) and its trading performance matches or exceeds that of the human trader. At this point the services of the human trader are no longer required.

In the language of research psychologists, our approach sketched in this story is a *behaviorist* one: we are concerned only with the "sensory inputs" and "motor outputs" of the human trader, we do not care about (or, at least, we make no pre-commitment to) modelling her internal mental states, or her internal reasoning processes; we do not need to interview her in some "knowledge elicitation" process (cf. e.g. [6]) to find out what analysis she performs on the incoming data, what sequence of decisions leads her to issuing a particular order; we do not require our black box to internally compute a GARCH model, or even a MACD signal: all we ask is that when presented with a stream of specific market-data inputs, the outputs of our box is a stream of orders that lead to trading performance at least as good as the human trader that the box learned from.

In this paper, we demonstrate a proof-of-concept of such a system, called DeepTrader. We have not yet put it in a metal box with a single indicator lamp, but we've got the software working.

At the heart of DeepTrader is a Deep-Learning Neural Network (DLNN: see e.g. [9][12][22][28]), a form of machine learning (ML) that has in recent years been demonstrated to be very powerful in a wide range of application domains. DLNNs are instances of supervised learning, where training the ML system involves presenting it with a large training-set of 'target' input/output pairs: initially, when presented with a specific input, the output of the DLNN will be a long way from the target output, but an algorithm (typically based on the back-propagation of errors, or "backprop", introduced by [16]) adjusts the DLNN's internal parameters on the basis of the errors between the actual output for this specific input and the target output associated with that input, so that next time this input is presented, the difference between the actual and target outputs will hopefully be reduced. This process is iterated many times, often hundreds or thousands of cycles over training-sets involving many tens of thousands of target input/output pairs, and if all is well this leads to the errors reducing to acceptably small levels. Once the errors are small enough, the DLNN is hopefully not only producing close-to-target outputs for all of the input/output pairs in the training set, but it is also capable of generating appropriate outputs when presented with novel inputs that were not in the training set: i.e., it has generalized. For this reason, evaluating how well a DLNN has learned usually involves testing it post-training, on a test-set of input/output pairs that were not used in the training process.

In the fictional story we opened this section with, the input-output pairs in the training and test set would come from observing the human trader working her job in a real financial market: every time a significant event occurs in the market, an observable behavior of interest, that event or action is the desired output vector; and the associated input vector is some set of multivariate data that is believed to be necessary and sufficient for explaining the observable behavior of the trader – i.e. it is whatever data the trader is thought to have been exposed to and acting upon at the time the event occurred. In our work reported here, each input vector is calculated from a timestamped snapshot of a financial exchange's Limit Order Book (LOB) (also known as the Ladder in some trading circles), i.e. the array of currently active bids and offers at the exchange, represented as the prices at which there are limit orders currently resting, awaiting a counterparty, and the quantity (total size of orders) available at that price. The output vector, the action to be associated with each input, could be an order (a fresh quote, or a cancellation of a previous order) issued from the trader to the exchange, and/or it could be a trade executing on the exchange.

More generally, as a source of input-data for DeepTrader, we need a market environment, which we'll denote by M; and to generate the target outputs used in the training-set we need a training trader, which we'll denote by T. We think it arguable whether we actually need a test-set, as a standalone collection of fresh input-output pairs: in principle, once DeepTrader's DLNN training process has produced an acceptable drop in error-levels on the training set, then it could just be set to work on live trading in the market M – whether it makes a profit or a loss in that trading would then be the final arbiter of whether the learning was successful or not. Such an approach would suit risk-seeking developers who have sufficient funds available to take the financial hit of whatever losses an under-generalized DeepTrader makes before it is switched off: for the risk-averse, positive results from a test-set could provide useful reassurance of generalization before DeepTrader goes live.

Real professional human traders are typically very busy people who don't come cheap, and also there will most likely be some regulatory and internal-political hurdles to overcome if we did want to record the necessary amounts of data from a human trader, which would only serve to delay us. So, for our proof-of-concept reported here, we have instead used high-performing algorithmic trading systems (or "algos" for short) as our T, our training trader. Specifically, the algos that we use include two that have been repeatedly shown to outperform human traders in experiments that evaluated the trading performance of humans and algos under controlled laboratory conditions. These two "super-human" algos are known by the acronyms AA (for *Adaptive Aggressive*: [25][26]) and ZIP (for *Zero Intelligence Plus*: [4]). Given that these out-perform human traders, we reason that if DeepTrader's DLNN can be trained to match or exceed the trading behavior of these algorithms in the role of T, then the likelihood is that it will also do very well when we deploy the same methods albeit using data from a human T – this is a topic we return to in the discussion section at the end of this paper. Another advantage conferred by using algo traders as T at this stage is replicability: the source-code for the traders is in the public domain, and so anyone who wishes to replicate or extend the work we report here can readily do so.

Having identified a T to produce target outputs, we also need an M, a market environment to generate the inputs associated with each target output. Again, as this is a proof-of-concept study, instead of using data from a real financial market (with its associated nontrivial costs and licensing issues, and the difficulty of doing direct replication) we instead use a high-fidelity simulation of a contemporary electronic exchange. For the T in this study we use the long-established public-domain market-simulator BSE [2][5] as our source of input data. BSE is an open-source GitHub project written in Python, which was first made public in 2012, and provides a faithful detailed simulation of a financial exchange where a variety of public-domain automated trading algorithms interact via a Continuous Double Auction (CDA: the usual style of auction for all major financial exchanges, where buyers can issue bids at any time and sellers can issue asks/offers at any time) for a simulated asset: traders can issue a range of order-types (and cancellations), and BSE publishes a continuously-updated LOB to all market participants: it is timestamped snapshots of that LOB data that form the input data for training and testing the DLNN in DeepTrader. BSE includes a number of pre-defined algorithmic traders including AA and ZIP, so the Python source-code we used for our T traders can be found alongside the source-code we used for our M market, in the BSE GitHub repository [2].

The rest of this paper, much of which is drawn from [27], is structured as follows. In Section II we summarize the background to this work. Section III then describes our methods. In Section IV we present results which demonstrate that DeepTrader learns to outperform pre-existing algo traders in BSE, and matches or exceeds the trading ability of the two "super-human" algorithms AA and ZIP. Section V (drawn from [15]) further analyses those results; we discuss our plans for further work in Section VI, and offer conclusions in Section VII.

II. Background

Our work reported here uses a public-domain simulator of a contemporary electronic financial exchange running a CDA with a LOB, so in that sense our work is very much about AI in present-day and future financial trading systems, but the roots of our work, and of the simulator we use, lie in academic economics research that commenced more than 50 years ago.

In 1962, Vernon Smith published an article in the prestigious *Journal of Political Economy* (JPE) on the experimental study of competitive market behaviour [19]. The article outlined a number of laboratory-style market simulation experiments where human subjects were given the job of trading in a simple open-outcry CDA where an arbitrary asset was traded, while the experimenters looked on and took down their observations. The supply and demand curves used in these experiments were realistic, but were predetermined by Smith, who allocated each trader a private limit price: the price that a buyer cannot pay more than, or the price that a seller cannot sell below. Different buyers might be given different limit prices, and the array or schedule of limit prices would determine the shape of the demand curve in the experimental market; ditto for the schedule of sellers' limit prices and the resultant market supply curve. In this sense, Smith's experimental subjects were like sales traders in a brokerage or bank, running customer orders: some external factor sets a limit price, and the trader's job is to do their best to buy or sell within that limit. If a buyer can get a deal for less than her limit price, the difference is a saving; if a seller can get a deal for more than her limit price, that's profit. Economists use 'utility' or 'surplus' to refer to both the buyer's difference and the seller's difference, but as we're focused on applications in finance we'll use 'profit' for both.

The experiments run by Smith demonstrated a rapid convergence of a market to its theoretical equilibrium price (the price where the quantity of goods supplied is equal to the quantity of goods demanded, where the supply curve intersects the demand curve) in a CDA, even with a small number of traders. This was measured by using Smith's 'a' metric, a measure of how well transactions in the market converge on the equilibrium price. In 2002, Smith received the Nobel Prize in Economics for his work establishing the field of experimental economics, and variations of his experiments have become *de facto* standards for test and comparison of trading algorithms.

Winding forward roughly 30 years, in 1990 a competition was hosted at the Santa Fe Institute for designing the most profitable automated trading agent on a CDA [17]. Thirty contestants competed for cash prize incentives totaling $10,000. The prize money won by each contestant was in proportion to the profit that their agent received in a series of different market environments. The highest ranked algorithm, designed by Todd Kaplan, was a simple agent that would hide in the background and hold off from posting a bid/ask price whilst letting other traders engage in bidding negotiations. Once the bid/ask spread was within an adequate range, Kaplan's agent would enter and "steal the deal". Aptly, Kaplan's program was named *Sniper*. If the market session was about to end, Sniper was programmed to rush to make a deal rather than not make one at all.

Subsequent to this, in 1993 Gode & Sunder published a JPE paper investigating the intelligence of automated traders and their efficacy within markets [21]. They developed two automated trading agents for their experiments, the Zero-Intelligence Unconstrained (ZIU) and the Zero-Intelligence Constrained (ZIC). The ZIU trader generates completely random quote prices, whereas the ZIC trader quotes random prices from a distribution bounded by the trader's given limit price, so the ZIC's are constrained to not enter loss-making deals. Gode & Sunder's series of experiments were performed in a similar style and spirit to Smith's: they ran some human-trader experiments to establish baseline data, and then ran very similar experiment with markets populated only by ZIU traders, and then only by ZIC. In each market they recorded three key metrics: allocative efficiency; single agent efficiency; and profit dispersion. The allocative efficiency is a measure of the efficiency of the market. It is the total profit earned by all traders divided by the maximum possible profit, expressed as a percentage. Gode & Sunder's key result was that the allocative efficiency of ZIC markets was statistically indistinguishable from that of human markets, and yet allocative efficiency had previously been thought to be the marker for intelligent trading activity. Ever since, ZICs are used as a de facto standard benchmark for a lower-bound on automated traders.

Extending the work of Gode & Sunder, in 1997 Cliff [4] identified that there were certain market conditions where ZIC traders would fail to exhibit human-like market dynamics. This finding led Cliff to create an automated trading agent with some elementary added AI, one of the first adaptive automated traders, called Zero Intelligence Plus (ZIP). The ZIP trader calculates its own profit margin which, along with its given limit price, it uses to calculate its bid or ask price. The profit margin is determined by a simple machine-learning rule and is adjusted depending on the conditions of the market. If trades are occurring above the calculated price, the profit margin is increased/decreased depending on whether the trader is a buyer/seller.

At roughly the same time as Cliff was publishing ZIP, in 1998 Gjerstadt & Dickhaut co-authored a paper that approached the sales trader problem from a new perspective [16]. They developed a price formation strategy in a CDA that analyzed recent market activity to form a belief function. The frequencies of bids, asks, accepted bid and accepted asks, from a set number of the most recent trades were used to estimate the belief or probability that an ask or bid would be accepted at any particular price. With this trading strategy, which came to be widely referred to as the GD strategy, the function selects an ask/offer price that would maximize a trader's expected gain based on the data. The strategy produced efficient allocations and was found to achieve competitive equilibrium within markets.

Then in 2001 a team of IBM researchers modified GD by interpolating the belief function to smooth the function for prices that did not occur in the selected number of recent trades, and they named the new trading agent MGD (Modified GD) and published results in a paper at the prestigious International Joint Conference on AI (IJCAI) that generated worldwide media coverage [7]: the IBM team was the first to explore the direct interaction between automated trading agents and human traders in a methodical manner, using LOB-based CDA markets that were close to ones implemented in financial exchanges across the world, where the traders in the market were a mix of human traders and automated algorithmic traders (specifically: IBM's

MGD, Kaplan's Sniper, Gode & Sunder's ZIC, and Cliff's ZIP). Famously, the IBM team demonstrated that MGD and ZIP could consistently outperform human traders in these realistic market scenarios – that is, MGD and ZIP are `super-human' traders. And the rest, as they say, is history: the IBM work got the attention of many investment banks and fund-managmenet companies, and in following years the world of finance started to see ever increasing levels of automation, with more and more human traders replaced by machines.

Academic and industrial R&D continued after the landmark IBM study, and two significant subsequent developments were the extension of MGD into GDX, and a new ZIP-related trading algorithm called AA. Details of GDX were published in 2002 by Tesauro & Bredin [23]: GDX exploits dynamic programming to learn functions that better incorporate long term reward, and at the time it was published IBM claimed it as the world's best-performing public-domain trading strategy. Details of AA were published by Vytelingum in his PhD thesis [25] and subsequent article in the prestigious *Artificial Intelligence* journal [26]. The key element of AA is *aggressiveness*: a more aggressive trader places a bid/ask that is more likely to be accepted, while a less aggressive trader will aim to seek a larger gain. This trading strategy estimates the market's equilibrium price by using a weighted moving average and estimates the volatility of the market by using Smith's α metric.

Inspired by the IBM experiments pitting human traders against robot traders, a decade later in 2011 De Luca and Cliff ran a series of experiments, reported at IJCAI in [8], which suggested that AA dominates all known trading strategies and also outperforms humans, making AA the third trading strategy to be demonstrated as super-human. However, recently Snashall and Cliff [20] performed a brute-force exhaustive search of all possible ratios or permutations of different trading strategies for markets populated by a specific number of traders, consisting of over 1,000,000 market sessions, in order to show that AA doesn't always outperform GDX or ZIP: there are some circumstances in which AA can be dominated by GDX, or by ZIP.

While AA, GD, GDX, MGD, and ZIP were all early instances of AI in finance, in virtue of their use of machine learning (ML) to adapt to circumstances and outperform human traders, they all used relatively simple and traditional forms of ML. In the past decade there has been an explosion of interest in Deep Learning, the field that concentrates on solving complex problems through the use of "deep" (many-layered) neural networks, i.e. DLNNs.

It is commonplace to implement recurrent DLNNs for time series forecasting and a vast amount of research has been completed in this area particularly in spot markets where traders attempt to predict the price of a resource in the future. Predictions are often made to assist in generating a signal on whether a trader should buy, hold or sell the resource that they are trading. Although this project employs a DLNN, there is a clear distinction on how it is being used. Rather than being used to predict a future price, this DLNN will be applied to the sales trader problem directly: a DLNN (specifically, a Long Short Term Memory, or LSTM DLNN: see [12]) is created that receives a limit price from customer orders, considers the conditions in the market by extracting information from the LOB, and finally given all of this information produces a price to quote in the next order, a desired price to transact at.

To the best of our knowledge, there are only two pieces of work that are closely related enough to discuss here. The first is DeepLOB [28] which uses a form of DLNN traditionally used in image processing, to capture the spatial structure of a LOB, coupled with an additional recurrent DLNN that incorporates information gathered over long periods of time. The second is by Le Calvez and Cliff [14] which demonstrates preliminary results from the use of a DLNN to successfully replicate the behavior of a ZIP trader, but which used only the best bid and ask prices. As Sirignano and Cont [18] have recently and elegantly demonstrated, deeper (Level-2) LOB data can be highly informative about short-term market trends, so a natural question to explore given Sirignano and Cont's result is: can we extend the methods reported by Le Calvez and Cliff to instead use Level-2 LOB data? That is what we explore in this paper.

III. Methods

Comparing the performance of trading strategies is not a straightforward task. As previously mentioned, the performance of a strategy is reliant upon the other traders within the market and in real-world financial markets, it is implausible to know what algorithms other traders are using, as this information is confidential. Traders tend not to disclose their strategies in order to remain profitable, for obvious reasons. Nevertheless, there are well-established experiment-methods which can be used to compare trading agents. IBM's Tesauro and Das [24] present three separate experiment designs for comparing trading agents, two of which will be used here: in one-in-many tests (OMTs), one trader is using a different strategy to the all rest -- this test is used to explore a trading strategy's vulnerability to invasion and defection at the population level; and in balanced-group tests (BGTs) buyers and sellers are split evenly across two types of strategy, and for every trader using strategy A is matched with a trader using strategy B, with the matched-pair ech being given the same limit price. BGTs are generally considered to be the fairest way to directly compare two strategies.

BSE was used to generate and collect all of the data required to train the LSTM network for DeepTrader and then test its performance against existing trading strategies. BSE allows control of the supply and demand schedules for a market session: we specified a range of schedules with varying shapes to both the supply and the demand curves, to generate data from a wide range of market conditions.

BSE produces a rich flow of data throughout a market session, including a record of the profit accumulated by each trader: when we present our results in Section 4, we focus on average profit per trader (APPT) because this is metric is reassuringly close to the profit and loss (P&L) figure that real-world traders (humans or machines) are judged by.

The LOB maintained by BSE is updated and published to all traders in the market whenever a new limit order is added to it, whenever a market order executes, or whenever an order is cancelled (thereby taking liquidity off the LOB). The published LOB is represented within BSE by a data-structure made up of an order-book for bids, and an order-book for asks. Each of these two order-books contains a list of the prices at which orders are currently resting on the book, and the quantity/size available at each such price. From this LOB data, it is possible to calculate various derived values such as the bid-ask spread, the mid-price, and so on. BSE also publishes a 'tape" showing a time-ordered list of timestamped market events such as orders being filled (i.e., transactions being consummated) or being cancelled. The clock in BSE is usually set to zero at the start of a market session, so the time t shows how much time has elapsed since the current market session began.

DeepTrader takes as input 14 numeric values that are either directly available on BSE's LOB or tape outputs, or directly derivable from them: these 14 values make up the 'snapshot' that is fed as input to DeepTrader's LSTM network for each trade that occurred within a market session. The 14 values are as follows (the +/– prefixes on input values are used in Section V):

1. – The time t the trade took place.
2. + Flag: did this trade hit the LOB's best bid or a lift the best ask?
3. + The price of the customer order that triggered the quote that initiated this trade.
4. + The LOB's bid-ask spread at time t.
5. + The LOB's midprice at time t.
6. + The LOB's microprice at time t.
7. + The best (highest) bid-price on the LOB at time t.
8. – The best (lowest) ask-price on the LOB at time t.
9. – The time elapsed since the previous trade.
10. – The LOB imbalance at time t.
11. – The total quantity of all quotes on the LOB at time t.
12. – An estimate P^* of the competitive equilibrium price at time t, using the method reported in [25][26].
13. – Smith's α metric [19], calculated from P^* at time t.
14. The price of the trade.

The first 13 items in the list are the inputs to the network: if any of them is undefined at time t then zero is used. Item 14 is the output (target) variable that the network is training toward: this is the price that DeepTrader will quote for an order. With respect to Item 3 on this list, it is important to note that when DeepTrader is trading live in the market, it only has access to the limit-prices of its own customer orders.

Each of these 13 input variates can have values within differing ranges. As a single input consists of 13 different features and the contribution of one feature depends on its variability relative to other features within the input. If for example, one feature has a range of 0 to 1, while another feature has a range of 0 to 1,000, the second feature will have a much larger effect on the output. Additionally, values in a more limited range (e.g. 0 to 1) will result in faster learning. Therefore, when training a multivariate neural network such as in DeepTrader, it is common practice to normalize all features in the training dataset such that all values are within the same scale. We used min-max normalization: for further details see [25].

The BSE GitHub repository [2] includes source code for seven different trading strategies, four of which (AA, GDX, ZIC, & ZIP) have already been introduced. The remaining three are SNPR, a trader directly inspired by (but not identical to) Kaplan's Sniper; GVWY, a "giveway" trader that simply quotes at its own limit price, giving away all potential profit; and SHVR, a "shaver" trader whose strategy is simply always to undercut the current best ask by one penny, and/or to always beat the current best bid by one penny – this strategy is intended as a minimal model of a pesky high-frequency trader.

To create a large dataset to train the model, many market-session configurations were devised where the proportions and types of traders were varied. Each market session had 80 traders (40 buyers and 40 sellers). Additionally, each market session involved four different trading strategies. For each trading strategy, the number of buyers and sellers was always the same but there were five different proportion-groups of traders used. These proportion-group were: (20, 10, 5, 5), (10, 10, 10, 10), (15, 10, 10, 5), (15, 15, 5, 5), and (25, 5, 5, 5). Each number within a group denotes the number of buyers and sellers for a specific trading strategy within a market session. For example, the (20, 10, 5, 5) proportion group, indicates that there were 20 buyers and sellers of trading strategy 1; 10 buyers and sellers of trading strategy 2; 5 buyers and sellers of trading strategy 3; and 5 buyers and sellers of trading strategy 4 within this group.

Given that there are four trading strategies in each session selected from a total pool of 7 available strategies, there is a total of 35 different combinations (i.e. 7 combined 4).

Furthermore, there are 35 different permutations for each of the proportion groups listed. This led to a total of 1225 (=35x35) different market configurations where the proportions and types of traders were varied. Each market configuration was executed 32 times with different random-number sequences for additional variability giving a total of 39,200 different market sessions that were run to create the training data for DeepTrader.

Each individual market session takes approximately 30 seconds to complete, so running all 39,200 on a single computer would take approximately 13.5 days. For this reason, the decision was made to use Amazon's *Elastic Compute Cloud* (EC2) service to parallelize data generation and collection processes amongst 32 virtual machines (VMs). The Python library *Boto3* v.1.13.3 [10] was used to create, manage and terminate the VMs. Work was automatically split amongst the VMs by making every VM run each market configuration once and via a separate custom utility, created for this project.

Typically, neural networks have *training*, *validation* and *test* datasets. The training set is used to train the model, it is the data that a neural network learns from; whilst the validation dataset is used for tuning a model's hyperparameters, and the test dataset is used to evaluate a model's final performance. For this project, as the performance of the neural network is determined by how well it trades during a market session, the test data is generated dynamically as the trader interacts with the simulated market.

The LSTM network created consists of three distinct hidden layers. The first hidden layer is an LSTM layer containing 10 neurons. The final two hidden layers are both fully connected layers containing 5 and 3 neurons respectively. Each hidden layer uses the Rectified Linear Unit (Relu) as an activation function: further details are given in [27].

The training process is limited by the size of memory on the machine used to train the network: the training dataset was so large that using all data points at once is not practicable because it exceeds the memory limits of conventional commodity servers, and we did not have a national-scale supercomputer readily available. Therefore, was training was executed in batches. Each batch consisted of 16,384 data points and the Adam optimizer [13] is used to train the network. As is common in DLNN applications, the network's learning rates require careful selection, and Adam uses an adaptive learning rate method that calculates different learning rates based on the weights in the network, with the intention of finding a workable tradeoff between overfitting (if the learning rate is set too high) and long processing times (if it is set too low).

The function that was used to calculate the error (loss) within the network was the mean squared error (MSE), as described in more detail in [25]. An epoch in training is the network being presented with each data-point within the training dataset once. We trained DeepTrader's LSTM network for 20 epochs, and the error measure typically fell rapidly in the first 10 epochs and was thereafter asymptotic approaching a very low value for the remainder of the training process. So, in total, DeepTrader would be trained via exposure to LOB data from 20 x 39,200 = 784,000 individual market sessions, and each of those sessions would typically involve roughly 20 LOB snapshots, so the total number of snapshots used in training was around 15 million.

IV. Results

Figures 1 to 4 show box-plots summarizing results from our experiments. Each experiment involves n=100 independent and identically distributed trials in the particular market, with a different sequence of random numbers generated for each trial. In all these figures, the vertical axis is average profit per trader (APPT) and the box is plotted such that distance between the upper and lower edges is twice the inter-quartile range (IQR); the horizontal line within the box is the median, and any data-points that are more than 1.5 times the IQR from the upper or lower quartiles are regarded as outliers and plotted individually. Figures 1 and 2 show results from the balanced group tests (BGTs), while Figures 3 and 4 show results from the one-in-many tests (OMTs). As is described in detail in Chapter 4 of [25], as a test of the significance of the differences observed between the APPT for DeepTrader and the APPT for whatever pre-existing algorithm it is being tested against, we calculated 90% confidence intervals (CIs) around the mean and judged the difference in distributions to be significant if the CIs of the two trading strategies were non-overlapping.

Fig. 1a shows BGT comparison of APPT scores between DeepTrader and ZIP, and Fig. 1b shows BGT comparison of APPT scores between DeepTrader and AA. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that in this case there is no significant difference between ZIP and DeepTrader, but AA does significantly outperform DeepTrader. However, as we will see in Fig. 3, when AA is pitted against DeepTrader in one-in-many tests, AA is outperformed by DeeTrader: we discuss these AA results later in this section.

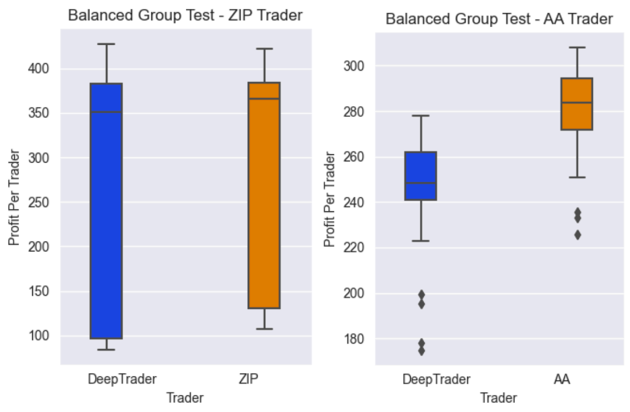


Fig. 1. Box-plots showing average profit per trader (APPT) from balanced-group tests (BGTs) for DeepTrader vs ZIP algorithmic traders (left: Fig1a) and AA algorithmic traders (right: Fig1b).

Fig. 2a shows BGT comparison of APPT scores between DeepTrader and GDX, and Fig. 2b shows BGT comparison of APPT scores between DeepTrader and ZIC. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that in this case DeepTrader significantly outperforms GDX, and ZIC too.

Fig. 3a shows OMT comparison of APPT scores between DeepTrader and ZIP, and Fig. 3b shows OMT comparison of APPT scores between DeepTrader and AA. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that DeepTrader significantly outperforms both ZIP and AA, although from visual inspection of the graphs it is also obvious that DeepTrader has much more variability of response than either ZIP or AA.

Fig. 4a shows OMT comparison of APPT scores between DeepTrader and GDX, and Fig. 4b shows OMT comparison of APPT scores between DeepTrader and ZIC. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that DeepTrader significantly outperforms both GDX and ZIC, although again from visual inspection of the graphs it is also obvious that DeepTrader has much more variability of response than either GDX or SHVR.

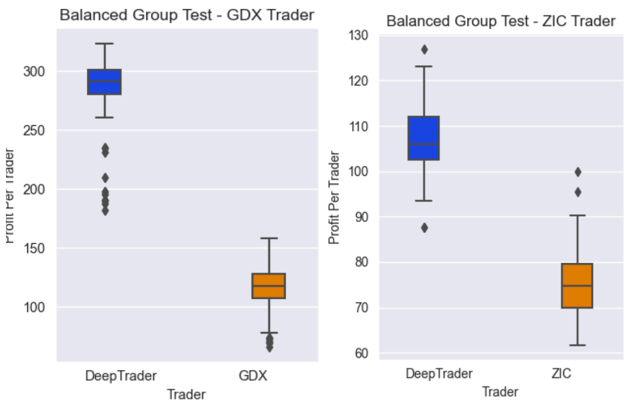


Fig. 2. Box-plots showing APPT from BGTs for DeepTrader vs GDX algorithmic traders (left: Fig.2a) and ZIC algorithmic traders (right: Fig.2b).

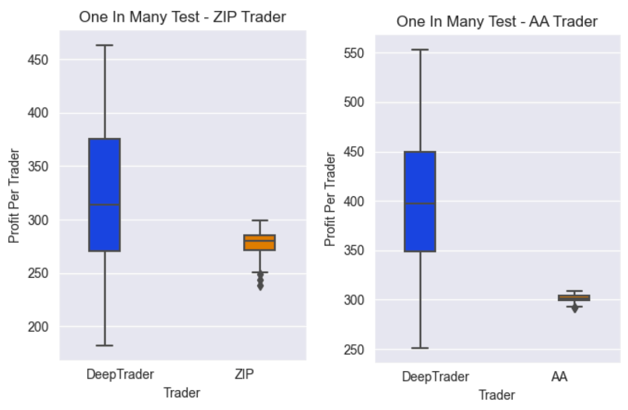


Fig. 3. Box-plots showing APPT from one-in-many tests (OMTs) for DeepTrader vs ZIP traders (left: Fig3a) and vs AA traders (right: Fig3b).

The results presented here demonstrate that DeepTrader achieves what we set out to do: when trained on a series of orders issued by a trader T, where each order is associated with a snapshot of the Level2 market data available to T at the instant that the order was issued, the DLNN in can be trained such that DeepTrader learns a mapping from the inputs (Level 2 market data inputs to DeepTrader) to outputs (quotes issued by DeepTrader) that result in superior trading performance when the final trained DeepTrader system is evaluated by allowing it to live-trade in the market environment M that the original trader T was operating in.

DeepTrader equals or outperforms the following trading algorithms in both the balanced group tests and the one-in-many tests: GDX, SHVR, SNPR, ZIC, and ZIP. Space limitations prevent us from including here further results, presented in [27], which show DeepTrader similarly learning to equal or outperform the rest of BSE's built-in algorithmic traders, i.e. GVWY, SHVR, and SNPR, thereby taking the total number of trading algos that DeepTrader outperforms to six. The most notable aspect of DeepTrader learning to trade at least as well as, or better than, this list of six different algorithms is that it includes ZIP, one of the two "super-human" algo traders for which code is already available in BSE.

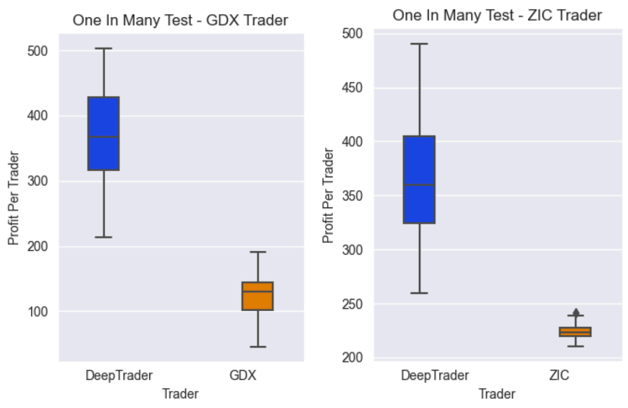


Fig. 4. Box-plots showing APPT from OMTs for DeepTrader vs GDX (left: Fig4a) and vs ZIC (right: Fig4b).

The results for DeepTrader when learning from (and then pitted against) the final algo studied here, Vytelingum's AA, the other super-human algo trader in BSE is somewhat less clear-cut: in the balanced group test, AA gets the better of DeepTrader; but in the one-in-many tests, DeepTrader roundly outperforms AA. As these two sets of tests result in a 1-1 tie, it seems fair to call it a draw.

So, in summary the results presented here (which are expanded upon in [27] and [15]) collectively show DeepTrader having six clear wins and one draw. While seven straight wins would naturally be preferable, these results nevertheless clearly demonstrate that the approach we have developed here has merit, and warrants further exploration.

In particular, having successfully used deep learning to create such a successful trader, the success provokes a natural question: how does DeepTrader work? That is, in what way does it use the 13 input features when trading? This is a question that we start to answer in the next section.

V. Analysis

Thus far our focus has been on ablation studies: that is, removing, disabling or masking specific aspects of the DeepTrader network and then noting the effects of that ablation. In general terms, we perform a sequence of ablation studies and the results from such a sequence can prompt us to hypothesise about how we could increase the efficiency of DeepTrader; we then test such hypotheses by studying the market performance of edited versions of DeepTrader, ones that have been altered to reflect our hypotheses – and this process can be iterated in an attempt at reducing the DeepTrader network to a minimally complex version that retains the ability to produce the desired level of trading behavior.

As is described in more detail in [15] we ran a set of 13 experiments where in each experiment one of the 13 input features was "ablated" by having the values in that column of the training data-set randomly shuffled (so the frequency-distribution of values in that column was unaltered, but any correlation between the value in that column and the other 12 features was very heavily disrupted); this gave us 13 sets of results where we could record the performance hit, the increase in error rate, when a specific one of the 13 features was ablated. A relatively large performance hit for a specific feature is an indication that DeepTrader's good trading behavior is indeed to reliant on that feature, whereas if the performance hit was sufficiently low then we considered that as an indication that the feature in question was, to a first approximation, irrelevant (or, at least, of sufficiently limited significance that it could be safely ignored) – that is, a low performance hit for ablating a feature led us to hypothesize that DeepTrader could operate successfully without access to that feature as an input; to check this, we re-ran our experiments with the feature (and its associated neurons within DeepTrader) absent, to check that the behavior of the resultant trader was consistent with our belief.

Via this method, we eliminated 7 of the 13 features (those prefixed with a – character in this enumerated list of 13 features in Section III), leaving a 6-input DeepTrader network which, when re-trained and tested, proved to show a slim improvement in its validation loss result: i.e. eliminating the seven features identified by ablation studies and using only the remaining six did not cause any loss of performance, and actually gave a very slight improvement. Thus we conclude this paper with the observation that only the six features prefixed with a + symbol in the enumerated list in Section III are required to trade as well as the 'super-human' trading agents ZIP and GDX.

VI. Further Work

Although at the start of this paper we characterized our approach to the use of learning in DeepTrader as *behaviorist*, because we concentrate only on the observable inputs and resultant trading behavior of the system, and although this approach was demonstrated by the results in the previous section to be one that delivers successful results, we do not intend to always treat DeepTrader as an impenetrable black box system. Instead, our next phase of work will be devoted to analysing the internal mechanisms that make a trained DeepTrader so successful. In particular, we will investigate the extent to which each of the 13 inputs listed in Section III contribute to the behavior of DeepTrader in a variety of market conditions: it is possible that some of those inputs play a much more significant role than others, and it is possible that which inputs are significant is not constant across all market conditions: we will report on our findings in this respect in a future publication.

VII. Discussion and Conclusions

We have explored here the problem of using machine learning to automatically create high-performance algorithmic traders that are fit to operate profitably in a contemporary financial exchange based (as are all current major electronic exchanges) on a CDA process mediated by a continuously updated limit order book showing Level 2 market data. Our approach to this problem is behaviorist, in the sense that we seek to use machine learning to replicate or exceed the trading behavior of an existing high-performance trader, and we do this purely by specifying a set of desired outputs for particular inputs: we have made no commitment to any particular approach being incorporated within the trader's internal processing that maps from externally observable inputs to outputs; instead we treat DeepTrader as an opaque black-box.

The novel results presented in this paper demonstrate for the first time this approach being used successfully against a range of pre-existing algorithms, including both AA and ZIP which had previously been shown to outperform human traders. Given that AA and ZIP are already known to exceed the capabilities of human traders in LOB-based CDA markets, it seems plausible to conjecture that the methods used here could in principle be extended to operate on training data that comes from observation of a human trader rather than an algorithmic trader. The basic approach, of associating snapshots of the LOB with orders issued by the trader, should work independently of whether the trader issuing the order is a person or a machine. And, in that sense, the little story that we started this paper with may not be fiction for much longer.

References

[1] M. Avellaneda and Sasha Stoikov. "High-frequency trading in a limit order book." Quantitative Finance, 8(3):217–224, 2008.

[2] BSE: A LOB-Based Financial Exchange Simulator. Github open-source repository (code & documentation) https://bit.ly/2XdW184.

[3] C. Cao, O. Hansch, and X. Wang. "The information content of an open limit-order book". Journal of Futures Markets, 29(1):16–41, 2009.

[4] D. Cliff. Minimal-Intelligence Agents for Bargaining Behaviors in Market-Based Environments. Technical Report, HP Labs, 1997.

[5] D. Cliff. "BSE: an open-source limit-order-book exchange for teaching and research." In S. Sundaram, editor, Computational Intelligence in Financial Engineering: IEEE Symposium Series on Computational Intelligence (CIFeR), pp1853–1860, 2018.

[6] N. Cooke. "Varieties of knowledge elicitation techniques". International Journal of Human-Computer Studies, 41(6), 1994.

[7] R. Das, J. Hanson, J. Kephart, and G. Tesauro. "Agent-human interactions in the continuous double auction". In Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI2001) Volume 2, pp.1169–1176, San Francisco, CA, USA, 2001. Morgan Kaufmann.

[8] M. De Luca and D. Cliff. "Human-agent auction interactions: adaptive-aggressive agents dominate". In Proc. Int. Joint Conference on Artificial Intelligence (IJCAI2011), Volume 1, pages 178–185, January 2011.

[9] J. Borges Gamboa. "Deep learning for time-series analysis". CoRR, abs/1701.01887, 2017.

[10] M. Garnaat. Python and AWS Cookbook: Managing Your Cloud with Python and Boto. O'Reilly, 2011.

[11] S. Gjerstad and J. Dickhaut. "Price formation in double auctions". Games and Economic Behavior, 22(1):1–29, 1998.

[12] S. Hochreiter and J. Schmidhuber. "Long short-term memory." Neural Computing, 9(8), 1997.

[13] D. Kingma and J. Ba. "Adam: a method for stochastic optimization". Proceedings ICLR, 2015.

[14] A. Le Calvez and D. Cliff. "Deep learning can replicate adaptive traders in a limit-order-book financial market". In S. Sundaram, editor, IEEE Computational Intelligence in Financial Engineering: Symposium Series on Computational Intelligence (CIFeR), pages 1876–1883, 2018.

[15] M. Meades, MSc Thesis, Forthcoming September 2020.

[16] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors". Nature, 323:533-536, 1986.

[17] J. Rust, J. Miller, and R. Palmer, "Behavior of trading automata in a CDA market". In D. Friedman and J. Rust (eds) The Double Auction Market: Theories and Evidence. Addison-Wesley, pp.155-198, 1992.

[18] J. Sirignano and R. Cont, "Universal features of price formation in financial markets: perspectives from deep learning". Quantiative Finance, 19(9):1449-1459, 2019.

[19] V. Smith. "An experimental study of competitive market behavior". Journal of Political Economy, 70(2):111–137, 1962.

[20] D. Snashall and D. Cliff, "Adaptive-aggressive traders don't dominate". In J. van den Herik, A. Rocha, and L. Steels, editors, Agents and Artificial Intelligence, pages 246–269, Springer, 2019.

[21] D. Gode and S. Sunder, "Allocative efficiency of markets with zero-intelligence traders: market as a partial substitute for individual rationality". Journal of Political Economy, 101:119–37, 02 1993.

[22] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection". In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pp.2553–2561. Curran Associates, Inc., 2013.

[23] G. Tesauro and J. Bredin. "Strategic sequential bidding in auctions using dynamic programming." In Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems: ACM Press, 2002.

[24] G. Tesauro and R. Das, "High-performance bidding agents for the CDA". Proc. 3rd ACM Conf. on E-Commerce, pp.206-209, 2001.

[25] P. Vytelingum. The Structure and Behaviour of the Continuous Double Auction. PhD thesis, University of Southampton, December 2006.

[26] P. Vytelingum, D. Cliff, and N. Jennings, "Strategic bidding in CDAs". Artificial Intelligence, 172(14):1700-1729, 2008.

[27] A. Wray. DeepTrader: A Deep Learning Approach to Training an Automated Adaptive Trader in a LOB Financial Market. Master's Thesis, Department of Computer Science, University of Bristol, May 2020.

[28] Z. Zhang, S. Zohren, and S. Roberts. "DeepLOB: deep convolutional neural networks for limit order books". IEEE Trans Sig. Proc., 67(11):3001–3012, 2019.