Dissertation Type: Research

University of BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

DeepTrader: A Deep Learning Approach to Training an Automated Adaptive Trader in a Limit-Order-Book Financial Market

Aaron Wray

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

Friday 29th May, 2020

Declaration

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

[illegible]

Aaron Wray, Friday 29th May, 2020

Contents

1 Contextual Background **1**
1.1 Introduction . . . 1
1.2 Motivation . . . 1
1.3 Previous Work . . . 2
1.4 Deep Learning In Finance . . . 4
1.5 Project Approach . . . 5

2 Technical Background **7**
2.1 The Limit Order Book . . . 7
2.2 Bristol Stock Exchange . . . 9
2.3 Comparing Strategies . . . 9
2.4 Automated Trading Agents . . . 9
2.5 Deep Learning . . . 15
2.6 Summary . . . 17

3 Project Execution **19**
3.1 Bristol Stock Exchange Modifications . . . 19
3.2 Data Collection . . . 20
3.3 Data Preprocessing . . . 20
3.4 Network Architecture . . . 21
3.5 Training . . . 21
3.6 DeepTrader . . . 22
3.7 Experiments . . . 22
3.8 Summary . . . 28

4 Critical Evaluation **29**
4.1 Confidence Interval . . . 29
4.2 Comparisons with AA . . . 29
4.3 Comparisons with GDX . . . 30
4.4 Comparisons with Giveaway . . . 30
4.5 Comparisons with Shaver . . . 31
4.6 Comparisons with Sniper . . . 32
4.7 Comparisons with ZIC . . . 32
4.8 Comparisons with ZIP . . . 33
4.9 Summary of Results . . . 33
4.10 Real World Application . . . 34

5 Conclusion **35**
5.1 Outcomes . . . 35
5.2 Future Work . . . 36
5.3 Final Words . . . 37

A Appendix **41**

Executive Summary

Deep learning neural networks (DLNNs) have proven to be remarkably successful in a number of domains that have previously been challenging for traditional machine learning methods. For example, image classification, speech recognition and natural language processing are areas in which DLNNs have shown superior results compared to other machine learning techniques and in finance, two of the biggest use cases for DLNNs are fraud detection and automated trading.

Over the past 20 years, a number of researchers in industrial and academic research labs around the world have proposed adaptive algorithms for trading, buying and selling in financial exchanges. Trading algorithms implement specific strategies and almost all of the well-known trading algorithms in the literature use either traditional machine-learning or adaptive-control mechanisms. These generally rely on only a small number of observable factors typically focusing almost exclusively on the history of prices quoted in the market. In contrast to traditional machine learning, adaptive algorithms typically use techniques from statistical machine learning to alter their responses to future market events on the basis of their past experience in the market.

In regards to the nature of the market mechanism, most of the world's major financial markets are continuous double auction (CDA) systems with a limit order book (LOB). The LOB is a tabular display of the prices and aggregated quantities of all competitive orders (bids or offers) currently posted by traders active in the market.

This project explores the use of DLNNs to train automated adaptive traders which trade on a CDA system that uses a LOB. This is a novel approach because the trading strategy will utilise a lot more data that is available on a LOB than just price information. Specifically, the hypothesis of this research is:

> Does the use of DLNNs - specifically recurrent neural networks (RNNs) - and the additional parameters given by a LOB such as the quantities supplied or demanded at each price, and extended time series information on past bids and offers, let us train and create a trading agent that is comparably better than existing trading agents in the public literature?

This project explores this hypothesis with the intention of answering it under the given time constraints of a master's degree and the limited resources and support received due to the Coronavirus pandemic. The following was completed in compiling this dissertation:

- A total of 150 hours was spent collecting material on and learning about different types of DLNNs, their applications and existing trading agents within the public literature.
- An RNN trading agent has been developed and trained using data created from market experiments.
- An additional 2000 lines of source code has been written for gathering and cleaning data to be used to train the RNN.
- Approximately 50,000 market experiments have been conducted in the cloud to obtain data for training.
- The performance of the RNN trading agent has been compared with traditional trading strategies, commonly referred to by the acronyms AA, GDX, GVWY,SHVR, SNPR, ZIC and ZIP.

- My results show that the RNN trading agent outperforms or equals the performance of all the following trading strategies: GDX, GVWY, SHVR, SNPR, ZIC and ZIP. My results for AA are ambiguous: in one set of experiments, AA outperforms my trader; in another set, my trader out performs AA. Investigation of this is one direction of future research.

Supporting Technologies

This section contains third-party resources and technologies that are used in this project:

- The code base of the entire project is written in Python version 2.7 and 3.6.3.
- To install packages in Python, the Python Package Installer PIP version 20.0.2 was used.
- The following Python libraries: TensorFlow(2.1.0), Keras(2.3.0), MatPlotLib(3.1.2), Numpy(1.18.2), Seaborn(0.10.0), Boto3(1.13.3) and Progress(1.5).
- The project uses Cliff's Bristol Stock Exchange (BSE) to run market simulations and as a test bed to compare trading algorithms.

Acronyms and Notation

Notation and Acronyms used throughout this document are as follows:

Acronyms

AA	:	Adaptive Aggressive
AI	:	Artificial Intelligence
BSE	:	Bristol Stock Exchange
CDA	:	Continous Double Auction
CNN	:	Convulutional Neural Network
CSV	:	Comma Separated Values
GD	:	Gjerstadt & Dickhaut
GDX	:	Gjerstadt & Dickhaut eXtended
DLNN	:	Deep Learning Neural Network
FCNN	:	Fully Connected Neural Network
LOB	:	Limit-Order-Book
LSTM	:	Long Short-Term Memory
MBC	:	Market Based Control
MBRA	:	Market Based Resource Allocation
MGD	:	Modified Gjerstadt & Dickhaut
RNN	:	Recurrent Neural Network
ZI	:	Zero Intelligence
ZIC	:	Zero Intelligence Constrained
ZIP	:	Zero Intelligence Plus
ZIU	:	Zero Intelligence Unconstrained

Notation

$Relu(x)$	:	Rectified Linear Unit of x
η	:	learning rate

Acknowledgements

Firstly, I would like to thank my supervisor who provided me with guidance to help me reach this milestone. Secondly, I would like to thank my girlfriend Frankie for her love and support. Finally, I would also like to acknowledge my brother and parents who provided me with the love, advice and encouragement to help me achieve all of my academic accomplishments.

Chapter 1

Contextual Background

1.1 Introduction

Auctions are a commonplace mechanism used in trading systems particularly in the financial markets. They are used for the allocation of resources, and for price discovery. The continuous double auction (CDA) is the most commonly used mechanism in financial markets, which also very often make use of a Limit Order Book (LOB) - a public record of currently active bids and offers. LOBs are implemented globally in many financial exchanges for trading commodities, derivatives, securities and other financial instruments.

Sales traders play a significant role within financial exchanges. The main function of the sales trader is to buy and sell financial instruments on behalf its clients to obtain the best prices for them whilst receiving a commission on each trade they execute. A key skill to being a successful sales trader is to be adaptive - having the ability to observe the conditions within a market and alter their behaviour accordingly to maximise profit.

In recent times, there has been a great rise in the use of Deep Learning Neural Networks (DLNNs) to solve a variety of problems. DLNNs are a machine learning method inspired by the information processing that occurs within the brain. They have been shown to be effective in numerous applications such as, object detection [22], speech recognition [9] and fraud detection [18]. Another notable use of neural networks is for time-series analysis where data is analysed at discrete points in time to establish a trend and predict future behaviour [8].

This project's aim is to apply the use of a DLNN in a unique way to the problem of creating an automated sales trader by using all of the parameters provided by the LOB, to build a trading strategy that is competitive and possibly superior to existing strategies previously published within the public domain.

Note that some of the extended LOB information that this research uses, is not available to the DLNN at the outset in the real world. The data is not usually collected by the trader in the course of trading and therefore will need to be acquired over a period of time once the strategy is implemented. Nevertheless, the purpose of this research is to ascertain whether a trading advantage over other automated traders could be obtained by collecting and eventually using this data in a DLNN.

1.2 Motivation

1.2.1 Economics

Trade is one of the major building blocks of the world economy. It facilitates global economic growth, the reduction of poverty and increases productivity across the world. It also allows for a greater amount of competition for goods, which leads to lower prices. However, the world economy is a complex system. It consists of an immeasurable number of trades across multiple markets happening concurrently every second. Hence a sophisticated yet dynamic system is required to manage and regulate how these trades are performed across the markets. One of the most effective and commonplace methods of managing these interactions is through the use of financial exchanges.

As financial exchanges act as a marketplace for trading as well as regulating how the trades are executed, they play a vital role in the economy. However for obvious reasons, they are not 100% efficient. Trades that occur on an exchange are a result of human decisions as well as automated trading agents, which all operate within their own self interest. This can lead to inconsistencies, such as inefficient pricing and resource misallocation.

Consequently, there are opportunities for arbitrage within the existing system which if exploited successfully can result in financial gain. Moreover, if price discovery can be optimised within these constantly evolving marketplaces this will lead to a more efficient economy and a better society which everyone can prosper and benefit from.

1.2.2 Computer Science

The allocation of resources is not a problem limited to the field of economics. The terms Market Based Resource Allocation (MBRA) or Market Based Control (MBC) describe a set of problems that require fast, robust and distributed control over a resource in a dynamic environment. In the field of Computer Science for example, this is a recurring problem that takes many forms such as sharing out bandwidth in a network or allocating resources to multiple processes in an operating system competing for limited hardware resources.

A Computer Scientist studies the theory and practical applications of computation, and common problems involving MBRA have been researched extensively. Their job does not stop at finding a solution to a problem, but also about pursuing the one that is most optimal. Artificial Intelligence (AI) is one way of doing just that, a field now widely used to address the problem that is generally considered to have been established at a Dartmouth Summer Research Project in 1956 [14].

The term machine learning a subset of AI is used to describe methods for computer algorithms to learn from data, and then perform a set of tasks, not necessarily repetitive, without being explicitly programmed to. Machine learning therefore provides a platform to improve existing solutions and solve problems that were thought to be too complex or unpredictable in the past. However, one of the major downsides to the use of AI was thought to be the amount of data required to train AI algorithms as well as the resources required to process it in order to produce desired results. Fortunately, the necessary amount of data and computational power required to train DLNNs is now readily available and, as a consequence, the last few decades has seen a re-emergence in AI, specifically the deep learning a subset of AI. With these developments, DLNNs now present a new and increasingly viable approach to building a strategy for the sales trader problem.

1.3 Previous Work

1.3.1 The Beginning: Experimental Economics

In 1962, Vernon Smith published an article in the Journal of Political Economy on the experimental study of competitive market behaviour [19]. The article outlined a number of market simulations based on a CDA mechanism explained further in Section 2.1, where an arbitrary instrument was traded. The supply and demand curves used in theses experiments were realistic, similar to those encountered by real life commodity traders, but were predetermined by Smith. He achieved this by setting the limit prices for each trader - the price that a buyer cannot buy above and for a seller, the price that a seller cannot sell below.

The experiments consisted of a small number of human traders, which were divided at random into two subgroups of buyers and sellers. The role of each trader was similar to that of a sales trader i.e. to maximise profit, which in this case is the difference between the price they traded at and their limit price. Each experiment was carried out over a succession of trading “days”, which lasted between 5 and 10 minutes, depending on the number of traders.

During each trading day, buyers and sellers would shout out their bids and offers within their given limit prices, much like posting prices to a LOB, except in Smith’s early experiments the bids and offers were not recorded. A respective buyer or seller could choose to accept the shouted out bid or offer, or alternatively, a trader could shout out a better price. If a bid/offer was accepted by a trader the trade price was recorded by both parties and both traders would then leave the market, as a trader could only buy or sell a single unit of the instrument being traded.

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) in a CDA, even with a small number of traders. This was measured by using Smith's α. Smith's α is a measure of how well traders in the market collectively discover the equilibrium price. In 2002, Smith received the Nobel Prize for establishing the field of experimental economics and variations of his experiments have been used to test and compare algorithms throughout research on automated trading agents [3, 6, 7, 16, 21]. Likewise, a version of his experiments will be conducted and analysed in this project to support the validity of this research.

1.3.2 The Rise of the Automated Trading Agent

In 1990, a competition was hosted at the Santa Fe Institute for designing the most profitable automated trading agent on a CDA [16]. Thirty contestants competed for cash prize incentives totalling $10,000. The prize money won for 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 negotiations. Once the bid/ask price was within an adequate range, the Kaplan's agent would then enter and "steal the deal". Aptly, his program was named Sniper. During the end of a market session, Kaplan's Sniper would also take advantage of the time left and attempt to make a deal rather than not make one at all.

In 1993, Gode and Sunder investigated 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 within a given limit price.

Gode and Sunder's series of experiments were performed in a similar format to Smith's but with different combinations of both human and ZI traders. Three key metrics that were looked at were 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 utility and is expressed as a percentage whereas the single agent efficiency is the profit earned by an agent divided by its expected profit.

Gode and Sunder claimed that their research showed that the ZIC traders were able to equilibriate the market, and that most of the intelligence is in the market and not in the traders, as the allocative efficiency was close to 100%, irrespective of the traders within the market. Additionally, on the basis of the single agent efficiency the experiments indicated that the ZIC agents were surprisingly human like, which implied that human traders aren't much better than random.

Continuing on from the work of Gode and Sunder, Cliff identified that there were certain market conditions where ZIC traders would fail to equilibriate [3]. This finding led him to create an automated trading agent which could converge to the market equilibrium. Consequently in the same paper, he outlined a new automated trader, called Zero Intelligence Plus (ZIP). ZIP is unique when compared with all of the other aforementioned traders, as it incorporates the idea of adaptiveness.

The ZIP trader utilises a profit margin along with a given limit price to calculate an ask/offer price. The profit margin is determined by a learning rule and is adjusted depending on the conditions of the market. If trades are occurring above the calculated price, the profit in margin is increased/decreased depending on whether the trader is a buyer/seller.

In 1998, Gjerstadt and Dickhaut co-authored a paper that approached the sales trader problem from a new perspective [10]. They developed a price formation strategy in a CDA that utilized recent market activity to form a belief function. The frequencies of bids, asks, accepted bid and accepted asks from a history of 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 a certain price.

With this strategy, the function selects an ask/offer price that would maximise a trader's expected gain based on the data. The strategy produced efficient allocations and was found to achieve competitive equilibrium within markets. A name was was not given to this strategy in the paper, but in research following from this, it was coined GD (by taking the first initial from both of the author's last names).

A few years after this, IBM researchers Das and Tesauro modified GD by interpolating the belief function to smooth the function for prices that did not occur in the selected number of recent trades. They named the new trading agent MGD (modified GD) [6]. The IBM paper was the first to explore the interaction between automated trading agents and human traders in a methodical manner, using CDA markets that were close to ones implemented in financial exchanges across the world.

The trading agents used in their experiments, were ZI traders, Kaplan's Sniper, ZIP, GD and MGD. Both ZIP and MGD consistently surpassed human traders during the experiment. This was an important finding, as at this time sales trading was a job that was only carried out by humans. However since then, a limited amount of research has been carried out about developing an automated trading agent. The majority of this research utilized old-fashioned machine learning techniques and heuristic methods.

One of the few areas of later research was published in 2002 by Tesauro. He followed up on his work on trading strategies by publishing a paper with Bredin that stipulates further alterations to the GD algorithm [23]. In the paper, he introduces a new trading strategy, GD eXtended (GDX), which exploits dynamic programming, to learn functions that express long term reward within a CDA.

Further on from this in 2006, Vytelingum created an adaptive aggressive trading agent, named AA, as a part of his PhD research [25]. The key element of this trading strategy is aggressiveness. A more aggressive trader places a bid/ask that is more likely to be accepted, where as a less aggressive trader will aim to seek a larger gain. This trading strategy estimates the market equilibrium by using a weighted moving average and calculates the volatility of the market by using Smith's α.

An aggressiveness function is updated based on the price volatility in the market. A more volatile market will give rise to a greater change in bidding behaviour if there is a small change in aggressiveness, whereas a less volatile market will cause a smaller change to bidding behaviour if there is only small change in aggressiveness. Inspired by the work done by Tesauro and Das, De Luca and Cliff ran a series of experiments on the OpEx market simulator, which suggested that AA dominates all known trading strategies [7]. However, some years on from this, Snashall and Cliff performed a brute force exhaustive search of all possible permutations of different trading strategies, consisting of over a 1,000,000 market sessions, in order to show that AA doesn't dominate [20]. It did show however, that AA was the best performing algorithm over all experiments.

1.4 Deep Learning In Finance

As previous mentioned, deep learning is the field that concentrates on solving complex problems through the use of "deep" (many-layered) neural networks which is otherwise known as DLNNs. There are many different types of DLNN, each having their own advantages, disadvantages and particular use cases. Nevertheless, DLNNs can be categorised into three main areas: Fully Connected Neural Networks (FCNNs), Convolutional Neural Network (CNNs) and Recurrent Neural Networks (RNNs). A brief discussion of FCNNs and CNNs will be given below, however the latter of the three, RNNs, will be given detailed treatment in this project mainly due to the advantages of using an RNN when processing temporal data.

It is commonplace to implement RNNs 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. Put differently, a neural network will be created that will receive a limit price from customer orders, consider the conditions in the market by extracting information from the LOB and finally, given all of this information will produce a price to transact at. Of course in this model, the DLNN is constrained by the limit price from the customer's order, however if the DLNN is to assist the trader in determining a price, it must work within these limitations.

To the best of my knowledge, there are only two pieces of work that are closely related enough to discuss here. The first is DeepLOB, which uses CNNs which are traditionally used in image processing.

Interestingly in this example, CNNs are used to capture the spatial structure of a LOB [27]. In addition to this after processing the information from the LOB, it uses an RNN to examine and take into account information over long periods of time.

Finally, a paper written by le Calvez and Cliff, the most relevant piece of work to this project demonstrates the use of a FCNN to successfully replicate the behaviour of ZIP in a CDA that uses a LOB [12]. It makes the claim of being the the first ever demonstration to successfully replicate a human-like, or super-human, adaptive trader operating in a realistic emulation of a real-world financial market and is a proof-of-concept to show that over time, a DLNN can learn to be better than all of the above.

1.5 Project Approach

One of the fundamental challenges of this project is using an RNN to learn to implement a new trading strategy. Essentially the main ambition lies in applying an RNN in appropriate conditions to create an optimal solution for the sales trader problem. Throughout the literature, claims have been made by the creators of different automated trading agents, that their trading strategy is the best [10, 23, 25]. However, recent research has demonstrated that the best trading strategy mostly depends on the strategies chosen by the rest of the agents within a market [20]. Additionally, a vast amount of data is required for an RNN to learn. This is not always readily available to researchers without investing significant resources in terms of cost and data manipulation processing.

In order to address the data volume requirements of the RNN for the project, the Bristol Stock Exchange (BSE), a LOB based simulation of a financial market, will be the test bed for this project. BSE provides the environment to collect training data needed to train an RNN, by producing information from the LOB throughout a market session. However, choosing the sets of trading strategies that will generate this data, as well as selecting a set of features from the training data need to be given careful consideration as the choices are crucial to the validity of the results across all conditions.

Two further challenges remain in obtaining fair and optimal assessment of any new trading strategy compared to other trading strategies namely identifying circumstances where the strategy underperforms compared to others or whether the strategy is able to equilibriate the market.

To this end, the high-level objective of this project is to use an RNN to create a automated trading agent which is competitive compared to a range of other automated trading strategies and performs well under a number of different trading conditions for the sales trader problem. Essentially, the approach to achieving this will be:

1. Research and survey literature on existing automated trading agents.
2. Build on work completed as part of the Applied Deep Learning Unit at Bristol, to further research into the use of RNNs.
3. Implement a data pipeline for the collection and pre-processing of training data from BSE modelling a range of trading conditions.
4. Create a profit making RNN which performs well under differing trading conditions. This will include defining its architecture and methodology, and how it should be initialised.
5. Compare the performance of the new strategy with existing automated trading agents under the various trading conditions, using *balanced-group* and *one-in-many* experiment designs.

Chapter 2

Technical Background

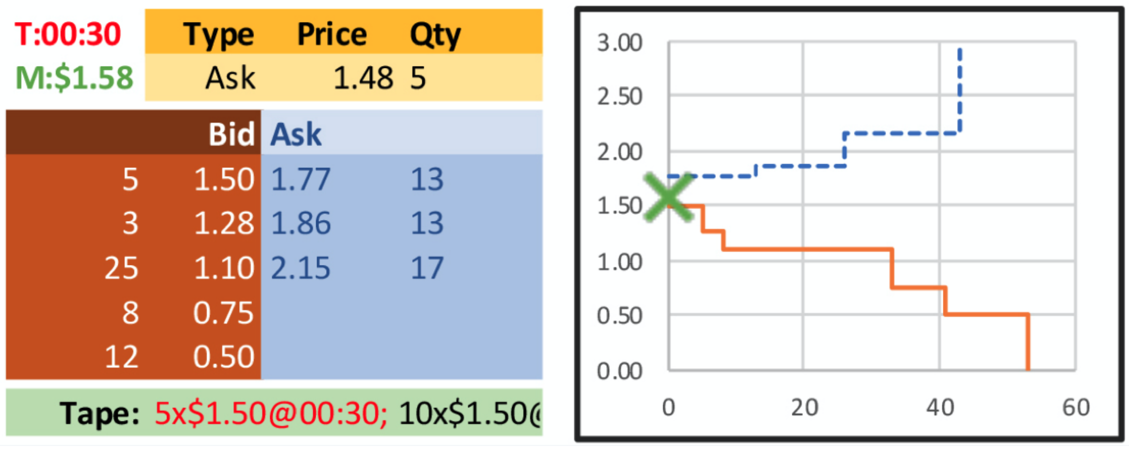
2.1 The Limit Order Book

An English auction, also known as an open ascending price auction, starts with an auctioneer announcing a starting price. Bidders attempt to outbid each other by bidding at the price given by the auctioneer. The auctioneer continuously increases the bid price, until there is only one person with the highest bid. The person with the highest bid wins the auction.

A Dutch auction, also known as an open descending price auction, is the reverse of an English auction. It starts with the auctioneer announcing a start offer-price or "ask". However, in this auction the auctioneer progressively reduces the ask price and the first person to accept it wins the auction.

The CDA is a combination of both the English and Dutch auctions, it allows for buyers to be matched with sellers and vice versa. This type of auction very often involves a LOB. A LOB is an anonymised data structure that is consists of two halves: a bids half for buyers to post their bid prices and an asks (offers) half for sellers to post their ask prices. On the buyers half of the LOB prices are listed in descending order with the highest bid price at the top. On the sellers half of the LOB prices are listed in ascending order with the lowest ask price at the top.

In this way, the best bid and best ask are always at the top of the book. For every price listed the quantity demanded or supplied is also listed beside that price. The LOB may also display the microprice (see Section 2.1.1 for definition) of the traded instrument. Additionally, financial exchanges will also very often use a tape to keep a record of each transaction. For each trade, the tape contains a timestamp, the parties involved and the price it took place at.


T:00:30
M:$1.58

Type	Price	Qty
Ask	1.48	5

	Bid	Ask	
5	1.50	1.77	13
3	1.28	1.86	13
25	1.10	2.15	17
8	0.75		
12	0.50		

Tape: 5x$1.50@00:30; 10x$1.50

Figure 2.1: Visualisation of a LOB/tape (left) and the supply/demand curves (right) for a security [5].

The difference between the best ask price and the best bid price is known as the "spread". The term "Crossing the Spread" is used for when a trade is taking place. A trade occurs if a buyer posts a bid price that is higher than the lowest offer price. This is known as "lifting the ask" - the trader wants to buy at the current best ask price. Similarly, if a seller posts an offer price on the LOB lower than the highest bid price, crossing the spread in this direction is known as "hitting the bid" - the trader wants to sell at the current best bid price. The information available on a LOB is often referred to as level 2 market data.

Figure 2.1 displays a LOB for a traded asset 30 seconds into a market session (on the left). There are several bids/asks that have been posted to the LOB. However, a trader has hit the bid by posting an ask of 1.48 (shown at the top) which is less than the best bid of 1.50. Therefore, the trader sells the security at best bid price of 1.50. The transaction is processed by the exchange and information is shown on the tape at the bottom.

2.1.1 Terminology

The LOB is an extensively researched data structure that contains a lot of information [1, 2, 3, 25, 27]. The intention of this section is to provide further clarification, definitions and where appropriate, formulae for key terms associated with both the LOB and for the trading strategies that use it.

The key terms are listed below:

- Competitive equilibrium price p^* - first referred to in Section 1.3.1, is the price at which the supply and demand curves intersect. Stated differently, it is where the quantity demanded of an asset is equal to the quantity supplied.
- Bid-offer spread - stated in Section 2.1 is calculated using the best bid p_b and best ask p_a see in Equation (2.1).

$$spread = p_a - p_b \tag{2.1}$$

- Utility u - is the for profit received by a trader. In BSE this is the absolute difference between trade price p and limit price l see in Equation (2.2).

$$u = |p - l| \tag{2.2}$$

- Midprice - is calculated using the best bid p_b and best ask p_a see in Equation (2.3). This is the only information that is given by the LOB in level 1 market data.

$$midprice = \frac{(p_a + p_b)}{2} \tag{2.3}$$

- Microprice - introduced in Section 2.1 is an adjustment to the mid-price to take the quantity demanded or supplied of an asset into account. The microprice is considered to be the more accurate value of an asset than the midprice. It is calculated using the best bid p_b, the quantity of the best bid q_b, the best ask p_a and the quantity of the best ask q_b see equation (2.4).

$$microprice = \frac{(p_b \times q_a) + (p_a \times q_b)}{(q_a + q_b)} \tag{2.4}$$

- Smith's α - described in Section 1.3.1 and 1.3.2 it is used to determine the price volatility in a market. It is the root mean square deviation of the prices p of trades T around the theoretical equilibrium price p^* as shown in Equation (2.5).

$$\alpha = \sqrt{\sum_{t \in T} \frac{p_t - p^*}{|T|}} \tag{2.5}$$

- Limit Order Book Imbalance I - determines whether the LOB is bid or ask heavy. The Imbalance is used as indicator for price direction and is calculated using the bid quantity q_b and the ask quantity q_a as in Equation (2.6).

$$I = \frac{q_b - q_a}{q_b + q_a} \tag{2.6}$$

- Allocative Efficiency - first mentioned in Section 1.3.2. Indicates how well the market performed in reaching its competitive equilibrium and is often used as a metric in homogeneous population tests. To calculate the allocative effiency a the total utility earned by all traders u and the theoretical maximum possible utility u^* are needed see Equation (2.7). Section 2.3 addresses this in further detail.

$$a = \frac{u}{u^*} \tag{2.7}$$

2.2 Bristol Stock Exchange

The Bristol Stock Exchange (BSE) is a minimal simulation of a limit order book financial exchange. It provides access to level 2 market data and was created by Cliff for teaching and research purposes [4]. There are several clear distinctions that can be made to distinguish the BSE from a real financial exchange which are outlined below.

Firstly, there is zero latency. If a trader posts a bid or ask, other traders are able to react to it instantaneously. There is no time between a quote being processed by the exchange and traders being able to respond to it. Secondly, it only permits a trader to submit one type of order, that is a limit order. A limit order is an order to buy/sell at a security at a specific price or better. Additionally, BSE only allows a buyer or seller to post a bid/ask with a quantity of one for a traded asset. If a trader is given a new order to execute from a client and posts a quote for it the old quote on the LOB will be deleted, in place of the new one. For each traded asset, the minimum price which can be quoted is 1 and the maximum price that can be quoted is 1000.

Finally, from the rules given by Smith's experiments, a buyer can only accept offers or post bids less than or equal to its given limit price [19]. Likewise, a seller can only accept bids or post offers greater than or equal to its given limit price. Therefore, an individual trader is unable to trade at a loss. A trade's profit is the absolute difference between the limit price and the trade price for a security. Despite these aforementioned differences, BSE is an entirely stochastic system that can be used to design experiments and analyse empirical data.

2.3 Comparing Strategies

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. Nevertheless, there are methods which can be used to compare trading agents. Tesauro and Das [6] present three separate tests for comparing trading agents two of which will be used in this project:

- Homogeneous population tests: a test where the market is populated entirely by one trading strategy. Although, this test defines an unrealistic scenario it is used to validate a training strategy and for future reference. Furthermore, it allows us to determine whether a specific trader is able to equilibrate the market. To calculate the allocative effiency of a market, knowledge of the supply and demand curves at all times are required. This would involve calculations after every market event. For this reason, it was thought to be beyond the scope of the project to perform these tests.
- One-in-many tests: a test where one trader is using a different strategy to the rest. This test is used to explore a trading strategy's vulnerability to invasion and defection. This test will be used to compare DeepTrader with existing strategies in this project.
- Balanced-group test: a test where buyer and sellers are split evenly with two types of strategy. In addition, every agent of one type of strategy has an identical limit price. This test is considered to be the fairest way to directly compare two strategies. This test will be used to compare DeepTrader with existing strategies in this project.

2.4 Automated Trading Agents

Section 1.3.2 provided a brief overview of automated trading agents within the literature and only some of the agents used in this project. However, this is not sufficient. A more detailed treatment of trading strategies is required to adequately assess results and make comparisons. The objective of this section therefore, is to deliver a greater insight into all of the automated trading agents used within this project.

2.4.1 Giveaway

The Giveaway trader is an adaption of Gode and Sunder's ZIU agent [21]. ZIU was designed to pick a completely random quote price and as a consequence, it very often trades at a loss. In the BSE, an individual trader is unable to trade at a loss, so the strategy was modified to "giveaway" and only trade at its limit price. The code for the agent is shown in Listing 2.1.

```
# The Order class used to create and handle orders
import Order
# The Trader class used to implement a specific trading strategy
import Trader

class Trader_Giveaway(Trader):
    def getorder(self, time, countdown, lob):
        # checks if the agent has a customer order
        if len(self.orders) < 1:
            # does nothing if there is no customer order
            order = None
        else:
            # sets the quote price as the limit price
            quoteprice = self.orders[0].price
            order = Order(self.tid, ...)
            self.lastquote=order
        return order
```

Listing 2.1: The code for the Giveaway trader

2.4.2 Zero-Intelligence Constrained

Another trading agent introduced by Gode and Sunder which is also used within this project is ZIC [21]. ZIC was designed to pick a random quote price that is also within its given limit price and BSE's minimum and maximum price. The code for the agent is shown in Listing 2.2.

2.4.3 Shaver

The Shaver trading strategy is considered to be an extension of ZIC. The ZIC trader does not utilize any information from the LOB, whereas the Shaver attempts to do just that. The Shaver's approach to the sales trader problem is to always be on the top of the LOB, by having the best bid or ask. To achieve this it "shaves" by slightly increasing the current best bid or decreasing the current best ask. In the event that doing this raises it above the limit price for a bid or below the limit price for an ask, the trader acts like Giveaway and just sets the quote price to be the limit price. The code for the agent is shown in Listing 2.3.

2.4.4 Sniper

The trading strategy that won the Santa Fe Institute trading agent competition was Kaplan's Sniper [16]. Kaplan's Sniper has been modified for BSE and will be referred to as Sniper in this project. Much like the original, Sniper lurks in the background and will attempt to "steal a deal". The implementation used for this project is an extension of the Shaver trading agent. Sniper utilizes a countdown parameter which indicates the amount of time left in the market session. The countdown is used to vary amount the Sniper "shaves" the best bid or ask by. When a market session is close to ending, the amount the strategy shaves by is reduced in an attempt to make a trade. The code for the agent is shown in Listing 2.4.

```
class Trader_ZIC(Trader):
    def getorder(self, time, countdown, lob):
        # checks if the agent has a customer order
        if len(self.orders) < 1:
            # does nothing if there is no customer order
            order = None
        else:
            # assigns the limit price given by the customer order
            limit = self.orders[0].price
            # assigns the order type e.g. bid/ask
            otype = self.orders[0].otype

            if otype == 'Bid':
                # generates a random quote price for a bid
                # The system minimum for a quote price in BSE is 1
                quoteprice = random.randint(1, limit)
            else:
                # generates a random quote price for an ask
                # The system maximum for a quote price in BSE is 100
                quoteprice = random.randint(limit, 1000)

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

Listing 2.2: The code for the ZIC trader

2.4.5 Zero-Intelligence Plus

As mentioned previously, Cliff identified that there were conditions that Gode and Sunder's ZI traders failed to equilibriate the market [3]. In response to these findings he developed his own trading strategy which could. Namely, ZIP. ZIP directly responds to market events by altering its profit margin M using a learning rule and does this whether or not it has an order. M is used to calculate a percentage multiplier. The multiplier and the limit price L are then used to determine a quote price P seen in Equation (2.8).

$$P = (1 + M) \times L \tag{2.8}$$

For a buyer $M \geq 0$ and for a seller $M \leq 0$, so that the P is always within the given limit price. M is updated throughout a market session using the Widrow-Hoff with momentum learning rule. If trades occur at prices greater than P then M is increased. Alternatively, if trades occur at prices less than P, M is decreased. This is done irrespective of a ZIP trader having an order. Thus, ZIP is an adaptive trader as the strategy takes conditions of the market and changes its behaviour.

2.4.6 Gerstadt Dickhaut eXtended

The paper "Strategic Sequential Bidding in Auctions Using Dynamic Programming" by Tesauro and Bredin outlines the GDX agent[23]. The GDX agent is an extension of the MGD agent that utilizes dynamic programming which computes a belief function f using a history H of n recent trades. The belief function is then used to estimate the probability for each possible bid/ask price p to be accepted at that price.

The belief function is calculated using: $able(p)$ - the number of accepted bids $\leq p$ in H, $ole(p)$ - the number of offers made $\leq p$ in H and $rgbe(p)$ - the number of rejected bids $\geq p$ in H as seen in Equation (2.9). The belief function also shows a zero probability for bids lower than the previous lowest trade and for offers higher than the previous highest trade.

```
class Trader_Shaver(Trader):
    def getorder(self, time, countdown, lob):
        # checks if the agent has a customer order
        if len(self.orders) < 1:
            # does nothing if there is no customer order
            order = None
        else:
            # assigns the limit price given by the customer order
            limitprice = self.orders[0].price
            # assigns the order type e.g. bid/ask
            otype = self.orders[0].otype

            if otype == 'Bid':
                # checks to see if there any bids on the LOB
                if lob['bids']['n'] > 0:
                    # sets the quote price to be the best bid plus one
                    quoteprice = lob['bids']['best'] + 1
                    # checks if the quote price is within the given limit price
                    if quoteprice > limitprice:
                        # assigns the quote price to be the limit price
                        quoteprice = limitprice
                else:
                    # no bids on the lob so posts the minimum price
                    quoteprice = 1
            else:
                # checks if there are any asks on the LOB
                if lob['asks']['n'] > 0:
                    # sets the quote price to be the best ask minus one
                    quoteprice = lob['asks']['best'] - 1
                    # checks if the quote price is within the given limit price
                    if quoteprice < limitprice:
                        # assigns the quote price to be the limit price
                        quoteprice = limitprice
                else:
                    # no bids on the lob so posts the maximum price
                    quoteprice = 1000
            order = Order(self.tid, ...)
            self.lastquote = order
        return order
```

Listing 2.3: The code for the Shaver trader

$$f(p) = \frac{able(p) + ole(p)}{able(p) + ole(p) + rgbe(p)} \tag{2.9}$$

The strategy interpolates the belief function over the prices that aren't in H using a cubic spline approach and then uses the belief function on p and the utility u at p to calculate the expected gain μ as shown in Equation (2.10). The trader sets the quote price to be a value of p that maximises μ.

$$\mu = f(p) \times u \tag{2.10}$$

2.4.7 Adaptive Aggressive

AA is a trading agent developed by Vytelingum in 2006 for his PhD research [25]. AA was previously was thought to dominate [7] until it was shown by Vach and Cliff in 2015 that this may not be true [24]

```
class Trader_Sniper(Trader):

    def getorder(self, time, countdown, lob):
        # parameters used to calculate how much to shave the best bid/ask
        lurk_threshold = 0.2
        shavegrowthrate = 3
        # the amount to shave the best ask/bid by
        shave = 1.0 / (0.01 + countdown / (shavegrowthrate * lurk_threshold))
        # checks if the agent has a customer order
        if len(self.orders) < 1:
            # does nothing if there is no customer order
            order = None
        else:
            # gets the limit price from the customer order
            limitprice = self.orders[0].price
            # gets the type of order e.g. bid/ask
            otype = self.orders[0].otype

            if otype == 'Bid':
                # checks to see if there are any bids on the LOB
                if lob['bids']['n'] > 0:
                    # generates quote price by adding on amount to shave by
                    quoteprice = lob['bids']['best'] + shave
                    # if quote price greater than limit price "giveaway"
                    if quoteprice > limitprice :
                        quoteprice = limitprice
                else:
                    # no bids on the LOB set bid to be min
                    quoteprice = 1
            else:
                # checks to see if there are any asks on the LOB
                if lob['asks']['n'] > 0:
                    # generates quote price by subtracting amount to shave by
                    quoteprice = lob['asks']['best'] - shave
                    # if quote price less than limit price "giveaway"
                    if quoteprice < limitprice:
                        quoteprice = limitprice
                else:
                    # no bids on the LOB set asks to be max
                    quoteprice = 1000
            order = Order(self.tid, ...)
            self.lastquote = order
        return order
```

Listing 2.4: The code for the Sniper trader

and altogether disproved by Snashall and Cliff in 2019 [20]. Despite this, AA is still considered to be the best performing strategy.

AA calculates an estimate of the competitive equilibrium $\hat{p}^*$ of a market session from all trades T using the weighted moving average as in Equation (2.11) [25].

$$\hat{p}^* = \frac{\sum^T (w_i \times p_i)}{|T|},\ where \sum^T w_i,\ w_{i-1} = \rho w_i \tag{2.11}$$

AA uses an aggressiveness model to calculate the trade price and identifies agents as one of two types. A trader is intra-marginal if it is a buyer and its limit price is higher than the estimated equilibrium price or if it is a seller and its limit price is lower than the competitive equilibrium price. Conversely, a trader is extra-marginal if it is a buyer and its limit price is lower than the estimated equilibrium price or if its a seller and its limit price is higher than the estimated equilibrium price. Intra-marginal traders are more likely to make trades within a market than extra-marginal ones.

The behaviour of the model not only depends on whether the trader is intra-marginal or extra-marginal, it also uses an aggressiveness level r. A fully aggressive intra-marginal trader ($r = -1$) makes the bid/ask its limit price in order to trade. An active intra-marginal trader ($r = 0$) will attempt to trade at the equilibrium price. A fully passive intra-marginal trader ($r = 1$) will quote a price in order to obtain the maximum profit available for a buyer that is the minimum price in the system and for a seller that is the maximum price.

An extra-marginal trader cannot be aggressive only active ($r = 0$) where it will trade at its limit price and passive ($r = -1$) where the agent will try to obtain maximum profit. The aggressiveness level is updated after every bid/ask is posted to the LOB using short term learning rules. Therefore, like GDX and ZIP it is an adaptive strategy.

2.5 Deep Learning

Introduced in Section 1.4, Deep Learning is a subset of Machine Learning that utilizes artificial neural networks to extract high level features from input data. The aim of a neural network is to infer a function that performs a specific task. Learning this way is often described as learning "deep", hence the name.

Supervised learning involves labelling the data, giving the neural network both input-output pairs so that a neural network can learn the mapping between them. Unsupervised learning requires the data to be unlabelled and only give a network input data, so that the neural network is able to draw inferences and learn the structure of the input data. This project concentrates on the former, as the aim is to use the information provided by running experiments on BSE to learn a mapping from the conditions given by a market and a set limit price to its target - a quote price.

2.5.1 The Basics

In 1958, a form of artificial neural network was invented by Rosenblatt who created the perceptron - a neural network with one hidden layer [15]. The perceptron was designed for image/pattern recognition.

Artificial neural networks are made up of three distinct sections: the input layer which is for the input data points, one or more hidden layers which contain a set of neurons and the output layer which produces an output to the problem. A single neuron represents an individual unit of computation.

A neuron is an activation function f and determines whether a neuron is active or not given a set of inputs x. Each input is given a specific weight w and the activation function is used to compute an output y on these weighted inputs. This can be seen in Equation (2.12) where i represents the output from one training example seen by the network. One or typically many neurons are used in combination to produce an output. For simplicity, Section 2.5.2 takes a single layer perceptron with one neuron into consideration. Figure 2.2 shows the example network.

$$y^{(i)} = f(w^T x^{(i)}) \tag{2.12}$$

2.5.2 The Delta Rule

A network attempts to learn the mapping between input and outputs by updating the weights after seeing each training example. The goal is to obtain the most optimal weights to solve a problem achieved by minimising the error within a network. This is done by performing gradient descent optimisation to adjust the weights within a network.

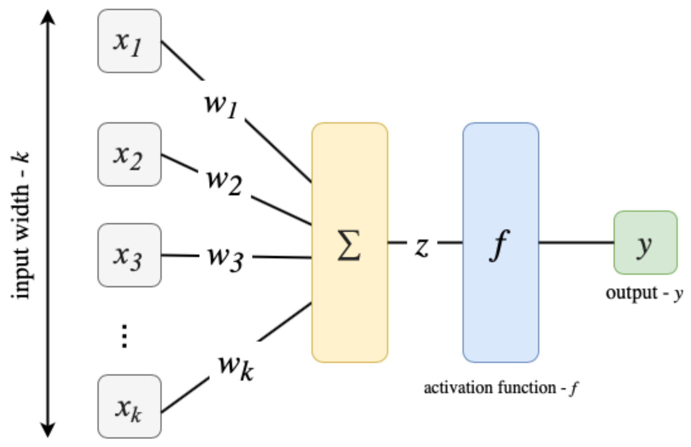


Figure 2.2: Displays a single layer perceptron with one neuron.

After the output, also known as the forward activity, has been calculated for a training example, an error function is used to calculate the error E within the network. The error function compares the output of the network y, with its true value v, to calculate the error. The error function may compute the error in different ways to calculate the network error. In this example, the error function used is the mean squared error and it is given for each training example i as shown in Equation (2.13).

$$E = \frac{1}{2}(v^{(i)} - y^{(i)})^2 \tag{2.13}$$

To update the weights, gradient descent optimisation is performed. This is done by computing the gradient of the error function with respect to each weight inside of the network. To do this, a few steps are required as described below.

Firstly, the derivative of the weighted inputs z with respect to a weight w_k is the input x_k and the derivative of the weighted inputs with respect to the input x_k is a weight w_k as shown on the left and right of Equation (2.14) respectively.

$$\frac{\partial z}{\partial w_k} = x_k \qquad\qquad \frac{\partial z}{\partial x_k} = w_k \tag{2.14}$$

Secondly, the activation function of a neuron is typically nonlinear, so that the model can solve problems that are nonlinear. For this example, the sigmoid function is chosen (see the left of Equation (2.15)). Thus, the derivative of the output of the network with respect to the weighted inputs can be determined as shown in the Equation on the right of Equation (2.15).

$$y = \frac{1}{1+e^{-z}} \qquad\qquad \frac{\partial y}{\partial z} = y(1-y) \tag{2.15}$$

The chain rule can then be used with the equations in (2.14) and (2.15) to find the derivative of the output with respect to each weight in the network see Equation (2.16).

$$\frac{\partial y}{\partial w_k} = \frac{\partial z}{\partial w_k}\frac{\partial y}{\partial z} = x_k y(1-y) \tag{2.16}$$

Additionally, the derivative of the error function with respect to the output is required.

$$\frac{\partial E}{\partial y} = (v-y) \tag{2.17}$$

One more application of the chain rule with Equations (2.16) and (2.17) gives the derivative of the error function with respect to each weight as in Equation (2.18).

$$\frac{\partial E}{\partial w_k} = \frac{\partial y}{\partial w_k}\frac{\partial E}{\partial y} = -x_k y(1-y)(v-y) \tag{2.18}$$

Finally, applying Equation (2.18) gives the rule to update the weights inside of the network as shown in Equation (2.19). The learning rate η is a configurable hyperparameter which determines how quickly the network learns.

$$\Delta w_k = -\eta\, x_k\, \underbrace{y(1-y)(v-y)}_{\sigma\text{ - error derivative}} \tag{2.19}$$

The learning rate has to be selected carefully. Small values can lead to long training times and large values can lead to overfitting. Overfitting is a common problem in Machine Learning and is when a model can only solve a problem for examples that exist within its training set. The model does not generalise well for unseen data.

Equation (2.19) can be reduced to Equation (2.20) which is also known as the Delta Rule [26].

$$\Delta w = -\,\eta\, x\, \sigma \tag{2.20}$$

2.5.3 The Backpropagation Algorithm

Section 2.5.2 discusses a network with one hidden layer. However, in order to learn "deep" and extract more features from input data, more hidden layers are required. Unfortunately though, for neural networks with more than one hidden layer, the problem of finding the error within in a network with respect to each weight or computing the steepest descent, becomes more complicated. Each hidden unit can have an effect on the output and therefore the effect of each hidden layer on the output has to be combined in some way.

An efficient way of solving this problem is by using the Backpropagation algorithm which utilizes dynamic programming [13]. It computes the error derivatives of one layer and then uses that to compute the error derivatives for the layer below "backpropagating" the error throughout a network.

Continuing on from Section 2.5.2, the sigmoid activation and mean squared error functions are used to achieve this. Equation (2.21) shows the new rule for updating weights where i and j represents a neuron from an individual layer. Additionally, the network sums the partial derivative for batches of training examples to compute the weight update.

$$\Delta w_{ij} = -\sum_{batch} \eta\, y_i\, y_j\, (1 - y_j) \frac{\partial E}{\partial y_j} \tag{2.21}$$

2.5.4 Long Short-Term Memory Recurrent Networks

Recurrent Neural Networks RNNs are designed for processing sequential data. RNNs are different from a regular neural network because they share the weights between all previous members of the output. In other words, weights are shared across time which makes RNNs ideal for processing temporal data. However, this leads to an even "deeper" computational graph, as not only do we have to consider the effect of each parameter on the output, the parameters from a previous time t have to also be considered.

Training an RNN is even more complicated then a regular fully connected neural network. To compute the gradient of the error function with respect to each weight in an RNN, the forward activity of the network at each time t has to be computed. The forward activity is then stored and used to backpropagate the error through all time steps. This algorithm is known as backpropagation through time.

In 1991, Hochreiter identified that RNNs suffer from the fundamental problem of deep learning: vanishing/exploding gradients [17]. Gradients propagated over time tend to vanish or explode at an extremely large rate such that the weights no longer have an affect on learning. Thus, causing a failure for a network to learn.

In 1997, Hochreiter and Schmidhuber published a paper on Long Short-Term Memory (LSTM) networks demonstrating a form of gated RNN which overcame the problem of vanishing/exploding gradients [11]. An LSTM captures the long term dependencies between time steps by using forget gates. A forget gate uses the sigmoid function (see the left of Equation (2.15)) to give an LSTM the ability to forget old states when necessary. Furthermore, it allows an LSTM to capture both the short and long term dependencies, but more specifically, the long term dependencies between inputs without the effect of vanishing/exploding gradients.

2.6 Summary

This chapter has provided a detailed technical overview of the sales trader problem and some of the existing strategies that have been used to solve it. Additionally, it outlines research completed on Deep Learning. Its intention has been to provide the reader with the technical understanding required to achieve the goal of this project namely to apply a Deep Learning methodology to the sales trader problem through the use of a LSTM network and to compare the new trading agent with existing strategies.

Chapter 3

Project Execution

3.1 Bristol Stock Exchange Modifications

As first introduced in Section 2.2, BSE is the test bed for this project. BSE was used to collect all of the data required to train the LSTM network for DeepTrader and then test its performance against existing trading strategies. BSE is written in Python 2.7 and therefore, for compatibility reasons, the majority of this project is also written in Python 2.7. There are a few notable exceptions namely some data preprocessing work, running BSE in the cloud and preliminary training of the LSTM network. These components can be considered to be independent of developing a trading strategy for BSE and for the reasons of usability and in some cases speed, they are all written in Python 3.6.

3.1.1 Training Data Price Ranges

In order to mimic the real world, BSE enables a user to control the supply and demand schedules for a market session. This includes the maximum and minimum price ranges for a customer's order. To expose DeepTrader to a wide range of supply and demand curves rather than avoiding a repetition.

3.1.2 Feature Selection

BSE produces data throughout a market session, and monitors the profit per trader. This data is recorded for every trader in a market session and the output is written to a file in CSV format. However, this data is insufficient to train an LSTM network to trade on a market.

After reviewing the literature and performing research (see sections 1.3.1 and 2.4), 14 additional features were extracted from BSE and output to the CSV file to train the LSTM network for each trade that occurred within a market session. These are as follows:

1. The time t the trade took place after the market session began in seconds.
2. A binary representation of the customer order type that was used to generate the quote, that initiated the trade (1 for bid and 0 for ask).
3. The limit price of the customer order that was used to generate the quote that initiated the trade.
4. The bid-ask spread on the LOB at time t defined in Section 2.1.1 if undefined set to 0.
5. The midprice of the LOB at time t defined in Section 2.1.1 if undefined set to 0.
6. The microprice of the LOB at time t defined in Section 2.1.1 if undefined set to 0.
7. The best (highest) bid on the LOB at time t if undefined set to 0.
8. The best (lowest) ask on the LOB at time t if undefined set to 0.
9. The difference in time between the previous and current trade, for the first trade this is the trade price.
10. The LOB imbalance at time t defined in Section 2.1.1.

11. The total quantity of all quotes on the LOB a time t.
12. An estimate of the competitive equilibrium price at time t as defined by Vytelingum [25] see Equation (2.11).
13. Smith's α calculated by using an estimate of the competitive equilibrium price at time t defined in Section 2.1.1.
14. The price of the trade.

The first 13 items in the list are the multivariate input to the network. Item 14 is the output (target) variable that the network is training toward.

A single CSV file was generated to store the data for all trades in one market session. A high volume of data was therefore generated for the many market sessions required to train the DLNN. To handle the heavy data and processing requirements, the market sessions were ran in the cloud.

3.2 Data Collection

Section 2.4 details all seven trading agents contained in BSE that were used in this project. To create a large dataset to train the model, many markets 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 used 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 proportions groups 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. As an 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 4 trading strategies in each session selected from a total pool of 7 available strategies, gives a total of 35 different combinations ($^7C_4$) of trading strategies.

Furthermore, there are 35 different permutations for each of the proportion groups listed. This led to a total of 1225 (35 x 35) 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.

This multiple implementation method of BSE is not ideal from a processing perspective as 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 parallelise data collection processes amongst 32 virtual machines (instances). The Python library Boto3 (1.13.3) was used to create, manage and terminate the EC2 instances. The work was split amongst each instance by making every instance run each market configuration once and used a separate custom utility, created for this project, to automate the process.

3.3 Data Preprocessing

The Pickle library was used to convert all CSV files produced from running the market sessions into one large continuous stream of bytes to save time when loading the dataset for training.

All of the data features used in training the DLNN have been previously outlined in Section 3.1.2. However each feature can have values within differing ranges. In this implementation, a single input consists of 13 different features and the contribution of one feature depends on its variability relative to other features inside of 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 this implementation, it is common practice to normalise all features in the training dataset such that all values are within the same scale. The normalisation method chosen here was min-max normalisation. Min-max normalisation scales all data in the range $[0, 1]$, by using the minimum and maximum values for each feature X in the training data. In this way, it does not alter any distribution within the data. The equation used to normalise each feature is shown in Equation (3.1), where i denotes a single training example.

$$X_{norm} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Typically, neural networks have train, validation and test datasets. The train dataset is used to train the model and 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, there is no distinct test dataset. Alternatively, the split of the test and validation datasets will be 90%:10% respectively from the data collected.

3.4 Network Architecture

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 as described by equation (3.2).

$$Relu(x) = max(0, x) \tag{3.2}$$

3.5 Training

The train dataset was too large to use all data points at once. The process is limited by the size of memory on the machine used to train the netowork. Therefore, was training was executed in batches. Each batch consisted of 16384 data points and the Adam optimiser is used to train the network. As previously discussed in in section 2.5.2, 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. The initial learning rate $\eta = 1.5 \times 10^{-5}$ which is a reasonable compromise between overfitting and long processing times.

The function that was used to calculate the error (loss) within the network was the mean squared error (MSE) given in Equation (2.13). An epoch can be considered as the model seeing each data point within the dataset once. The model was trained for 20 epochs and the loss after each training epoch is shown in Figure 3.1 below.

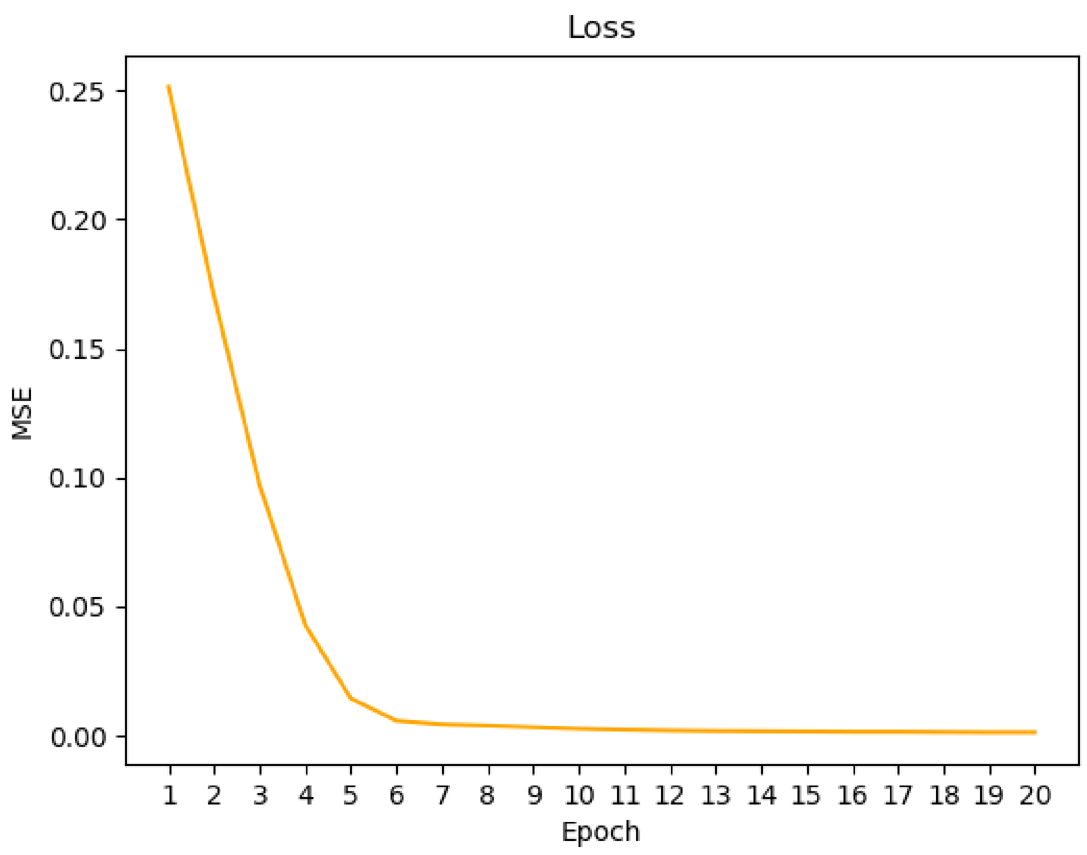


Figure 3.1: MSE loss after each training epoch

3.6 DeepTrader

DeepTrader uses the information provided by the LOB specified in Section 3.1.2 and a customer order's given limit price to calculate a price to trade at. However, it first has to normalise inputs before this information is used, in the same way as the training data. Consequently, the minimum and maximum values for all features within the training data are stored and loaded from a file so that they can be used to normalise the input to the model. Once DeepTrader has determined the price, the output will be in the range [0,1]. Therefore, the output is "denormalised" to produce a trade price that is appropriate for BSE. Listing 3.1 shows the algorithm used for denormalistaion.

3.7 Experiments

Section 2.3 outlined three different methods of comparing trading strategies. The most appropriate methods were deemed to be one-in-many tests and balanced-group tests for reasons highlighted in that section. Additionally, the most significant measure used when analysing and comparing the performance of trading agents is the amount of profit received per type of trader. Therefore, this is the key metric collected during the course of the experiments that are described in this section.

3.7.1 Balanced Grouped Tests

The Balanced Group Tests were carried out using DeepTrader and the other traders that exist within BSE. These tests consisted of 100 iid market sessions and each market session contained 20 buyers and 20 sellers of one type of trader. The ranges for limit prices are set in the same way as when the training data was collected (see section 3.2). Figure 3.2 shows box plots of results for the various balanced group configurations.

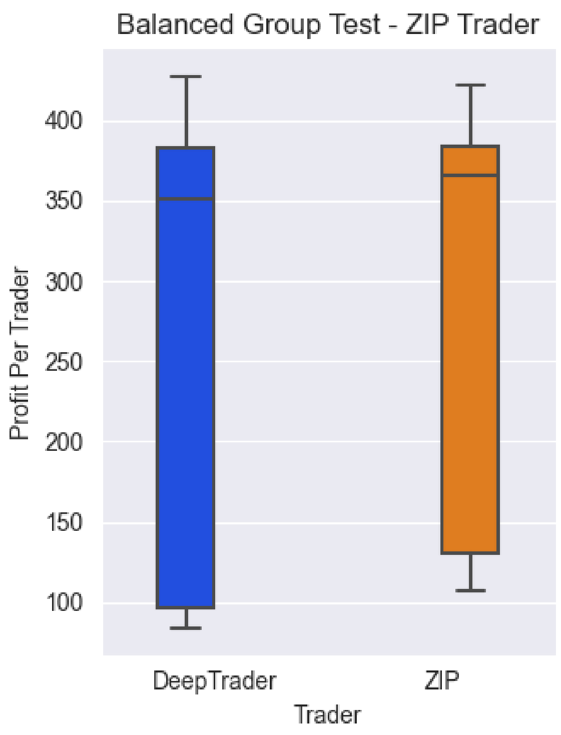
```
class DeepTrader(Trader):
    def getorder(self, time, countdown, lob):

        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

            # print(seld.tid, self.count)

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

Listing 3.1: The getorder Function of DeepTrader

For each box plot, the edge of the box displays the upper and lower quartiles, the line inside of the box shows the median of the dataset and the whiskers extend to show the rest of the distribution. Except, any data point that is a distance of 1.5 multiplied inter-quartile range from the upper or lower quartiles respectively is classified as an outlier. Outliers are represented on the box plots as diamonds.

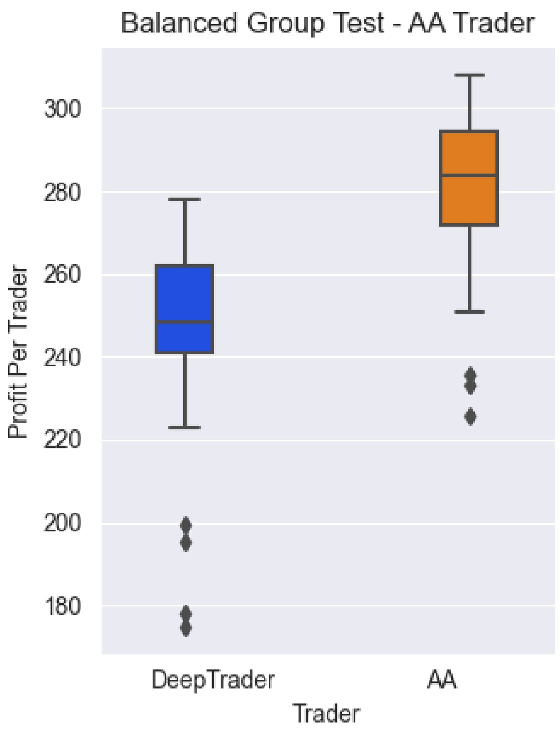
3.7.2 One In Many Tests

The One In Many test were also carried out using DeepTrader and the traders that exist within BSE. Again, these tests consisted of 100 iid market sessions. Each market session contained 1 DeepTrader buyer and 1 DeepTrader seller and 39 buyers and 39 sellers of another type of trading strategy. The ranges for limit prices are set in the same way as when the training data was collected (see Section 3.2).

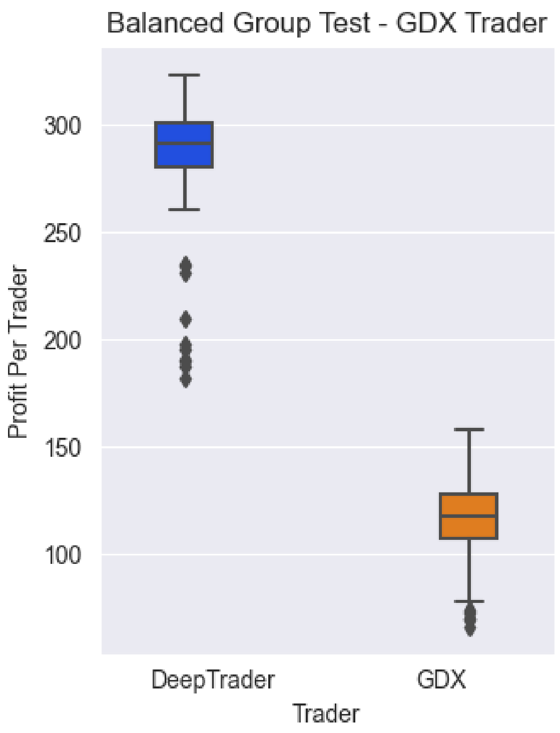


(a) Box plot to display profit per trader from the balanced group tests with DeepTrader and the ZIP trader.

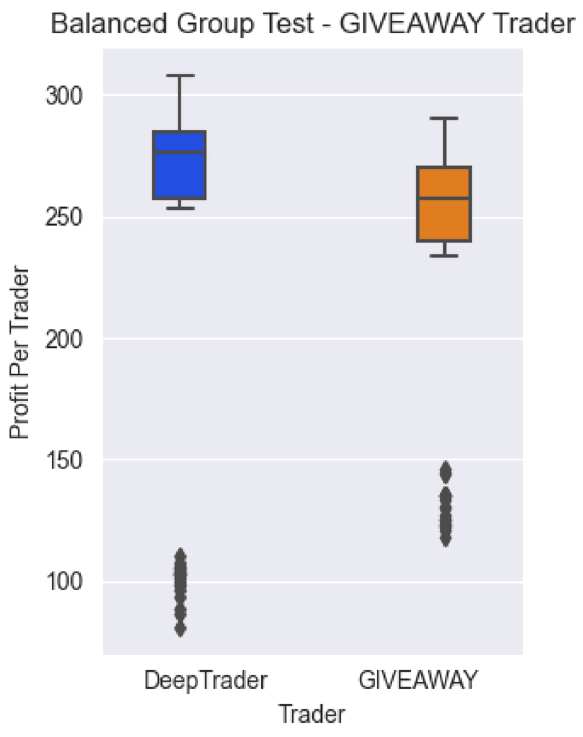
Figure 3.2: Balanced Group Test Results



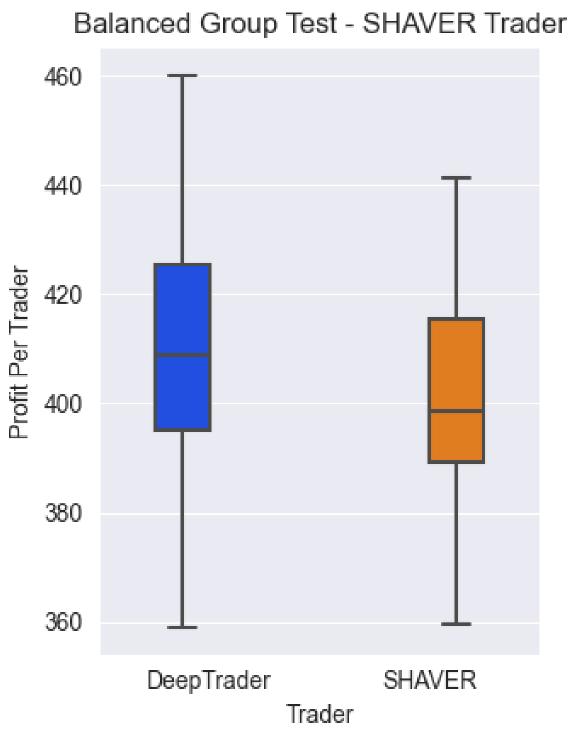
(b) Box plot to display profit per trader from the balanced group tests with DeepTrader and the AA trader.



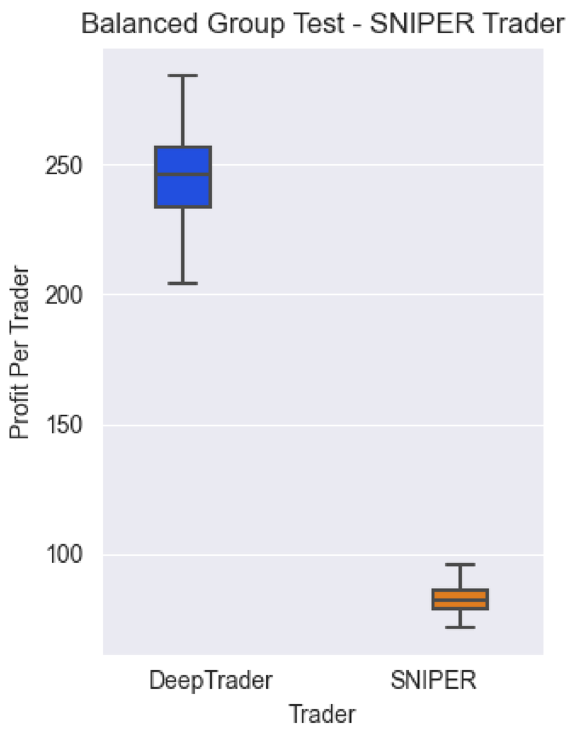
(c) Box plot to display profit per trader from the balanced group tests with DeepTrader and GDX trader.



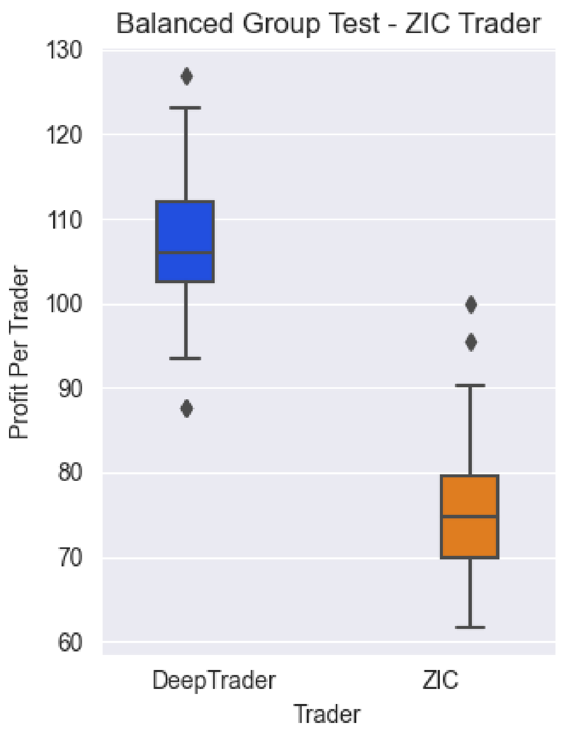
(d) Box plot to display profit per trader from the balanced group tests with DeepTrader and the Giveaway trader.



(e) Box plot to display profit per trader from the balanced group tests with DeepTrader and the Shaver trader.

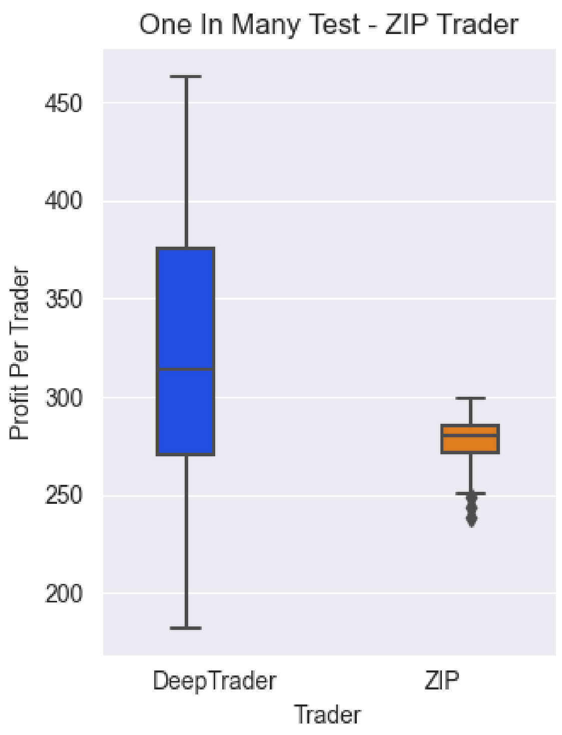


(f) Box plot to display profit per trader from the balanced group tests with DeepTrader and the Sniper trader.



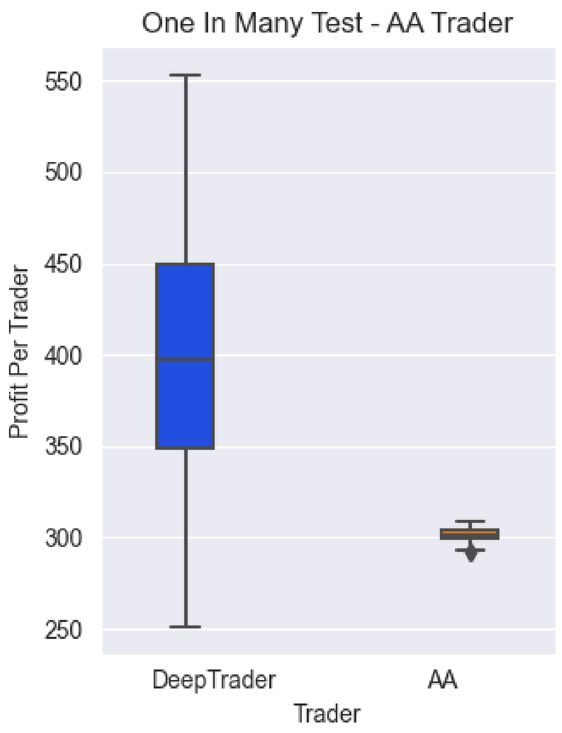
(g) Box plot to display profit per trader from the balanced group tests with DeepTrader and the ZIC trader.

Figure 3.1 below show box plots of results for the various one-in-many configurations.

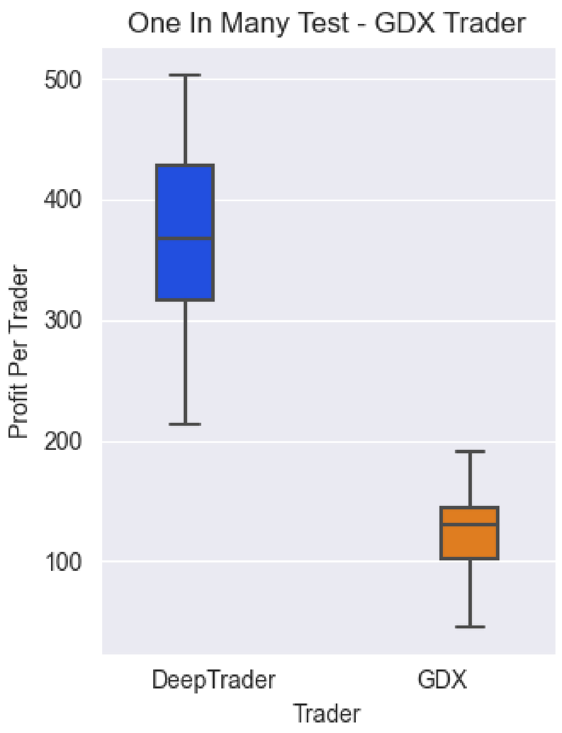


(a) Box plot to display profit per trader from the one in many tests with DeepTrader and the ZIP trader.

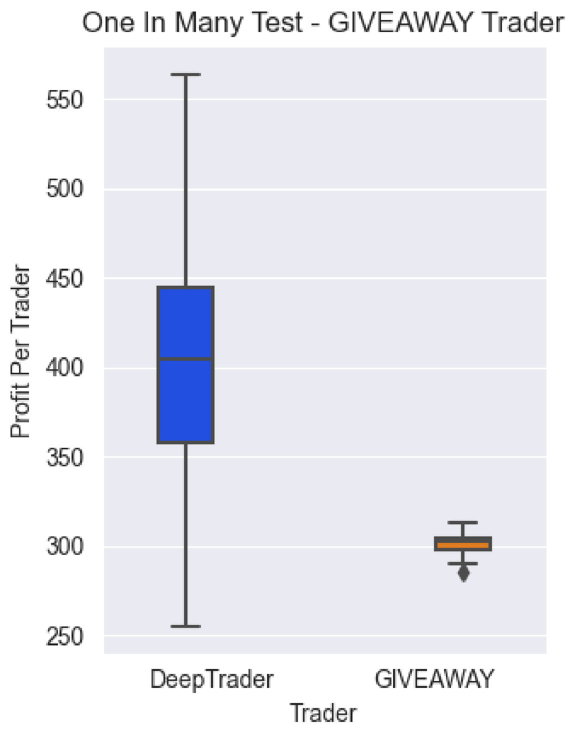
Figure 3.1: One In Many Test Results



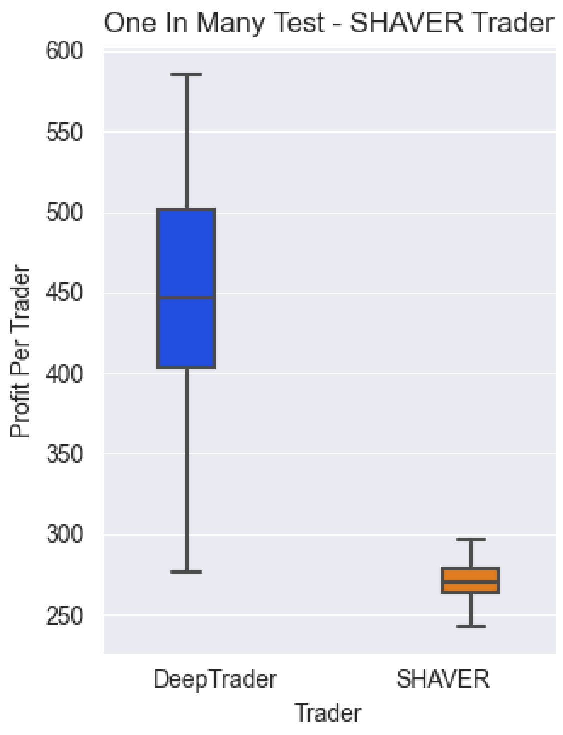
(b) Box plot to display profit per trader from the one in many tests with DeepTrader and the ZIP trader.



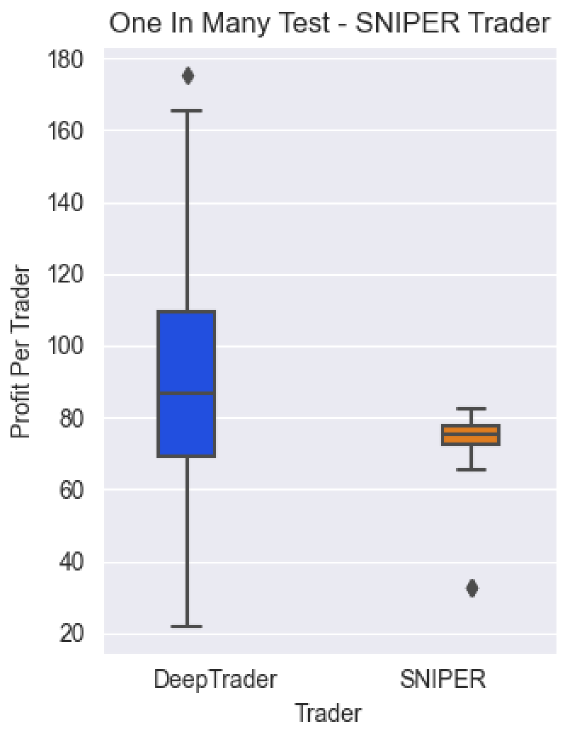
(c) Box plot to display profit per trader from the one in many tests with DeepTrader and the ZIP trader.



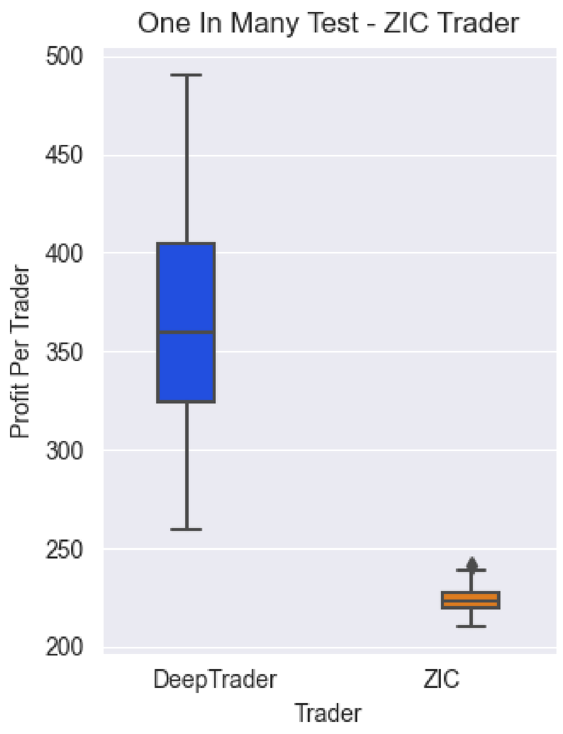
(d) Box plot to display profit per trader from the one in many tests with DeepTrader and the Giveaway trader.



(e) Box plot to display profit per trader from the one in many tests with DeepTrader and the Shaver trader.



(f) Box plot to display profit per trader from the one in many tests with DeepTrader and the Sniper trader.



(g) Box plot to display profit per trader from the one in many tests with DeepTrader and the ZIC trader.

3.8 Summary

This chapter establishes the actions completed in the project in order to successfully develop the DeepTrader strategy. In addition, a detail description of the experiments performed to compare DeepTrader with existing strategies was provided. Chapter 4 aims to provide statistical analysis from the experiments described.

Chapter 4

Critical Evaluation

The intention of this chapter is to evaluate the performance of the DeepTrader agent against existing trading strategies. This will be achieved by providing a detailed analysis of the results from the experiments that were presented in Section 3.7. For each test performed, this chapter will determine whether there is any statistically meaningful difference in the outcomes of the experiments for each trading strategy compared to DeepTrader under the various conditions. Where there is a significant difference, the evaluation will discuss which trading strategy is optimal.

4.1 Confidence Interval

The confidence interval $[c_1, c_2]$ for the mean $\bar{x}$ profit per trader produced by an experiment will be approximated. This will be done through the use of the central limit theorem see Equation (4.1) where s is the the standard deviation, n is the number of samples and Z_q is q^{th}-percentile of the standard normal distribution. The confidence intervals will be calculated using a significance level $\alpha = 0.1$. Therefore, $Z_{95} = 1.645$ and $n = 100$ in all cases. Analysis can be performed in this way as each sample drawn from an experiment is independent and there is a sufficient number of samples that have been drawn $n > 30$.

$$c_1 = \bar{x} - (Z_{1-\frac{\alpha}{2}} * \frac{s}{\sqrt{n}}), \;\; c_2 = \bar{x} + (Z_{1-\frac{\alpha}{2}} * \frac{s}{\sqrt{n}}) \tag{4.1}$$

4.2 Comparisons with AA

4.2.1 Balanced Group Comparison

Table 4.1 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between AA and DeepTrader. DeepTrader's upper bound estimate of the mean profit per trader c_2 is less than AA's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that AA outperforms DeepTrader in a market where both AA and DeepTrader each account for 50% of the market.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	249.1	17.86	246.2	252.0
AA	281.7	15.74	279.1	284.3

Table 4.1: Displays results from the Balanced Group Tests with DeepTrader and AA.

4.2.2 One In Many Comparison

Table 4.2 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between AA and DeepTrader. AA's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that DeepTrader outperforms AA when AA is the widely used strategy and DeepTrader is the minority. However, it is important to identify that DeepTrader has a large standard deviation for the samples in this test. Comparatively, AA has a very small standard deviation. The large standard deviation indicates that the profit per trader obtained from each experiment are spread far from the mean. Thus, DeepTrader outperforms AA in this setting but with a high variability.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	404.2	73.9	392.0	416.4
AA	301.2	3.930	300.5	301.8

Table 4.2: Displays results from the One In Many Tests with DeepTrader and AA.

4.3 Comparisons with GDX

4.3.1 Balanced Group Comparison

Table 4.3 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between GDX and DeepTrader. GDX's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the evidence demonstrates that DeepTrader is better than GDX in a market where both GDX and DeepTrader each account for 50% of the market.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	284.8	30.31	279.8	289.7
GDX	116.9	20.44	113.5	120.2

Table 4.3: Displays results from the Balanced Group Tests with DeepTrader and GDX.

4.3.2 One In Many Comparison

Table 4.4 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between GDX and DeepTrader. GDX's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the evidence indicates that DeepTrader is better than GDX in a market where GDX is the vast majority of trading agents and DeepTrader is the minority.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	367.3	72.06	355.4	379.1
GDX	123.5	31.52	118.3	128.7

Table 4.4: Displays results from the One In Many Tests with DeepTrader and GDX.

4.4 Comparisons with Giveaway

4.4.1 Balanced Group Comparison

Table 4.5 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between Giveaway and DeepTrader. The mean profit per trader for DeepTrader lies inside of the 90% confidence interval of Giveaway. Similarly, the mean profit per trader for Giveaway lies inside of the 90% confidence interval of DeepTrader. Therefore, at a confidence level of 90% there is no statistical evidence to suggest that there is a difference in performance between these two traders in a balanced group test.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	240.0	75.43	227.6	252.4
Giveaway	234.1	56.50	224.8	243.4

Table 4.5: Displays results from the Balanced Group Tests with DeepTrader and Giveaway.

4.4.2 One In Many Comparison

Table 4.6 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between Giveaway and DeepTrader. Giveaway's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the statistical evidence shows that DeepTrader is better than Giveaway in a market where Giveaway is the vast majority of trading agents and DeepTrader is the minority.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	400.9	68.19	389.7	412.2
Giveaway	301.9	5.313	301.0	302.7

Table 4.6: Displays results from the One In Many Tests with DeepTrader and Giveaway.

4.5 Comparisons with Shaver

4.5.1 Balanced Group Comparison

Table 4.7 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between Shaver and DeepTrader. Shaver's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the statistical evidence demonstrates that DeepTrader is better than Shaver in a market where both Shaver and DeepTrader each account for 50% of the market.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	410.1	21.26	406.6	413.6
Shaver	400.5	18.18	397.5	403.5

Table 4.7: Displays results from the Balanced Group Tests with DeepTrader and Shaver.

4.5.2 One In Many Comparison

Table 4.8 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between Shaver and DeepTrader. Shaver's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that DeepTrader outperforms Shaver when Shaver is the widely used strategy and DeepTrader is the minority. However, it is important to identify that DeepTrader has a large standard deviation for the samples in this test. Comparatively, Shaver has a very small standard deviation. The large standard deviation indicates that the profit per trader obtained from each experiment are spread far from the mean. Thus, DeepTrader outperforms Shaver in this setting but with a high variability.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	449.1	66.65	438.1	460.1
Shaver	270.5	10.29	268.8	272.2

Table 4.8: Displays results from the One In Many Tests with DeepTrader and Shaver.

4.6 Comparisons with Sniper

4.6.1 Balanced Group Comparison

Table 4.9 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between Sniper and DeepTrader. Sniper's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the statistical evidence demonstrates that DeepTrader outperforms Sniper in a market where both Sniper and DeepTrader each account for 50% of the market.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	246.0	17.03	243.2	248.8
Sniper	82.59	5.236	81.72	83.45

Table 4.9: Displays results from the Balanced Group Tests with DeepTrader and Sniper.

4.6.2 One In Many Comparison

Table 4.10 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between Sniper and DeepTrader. Sniper's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that DeepTrader outperforms Sniper when Sniper is the widely used strategy and DeepTrader is the minority. However, it is important to identify that DeepTrader has a large standard deviation for the samples in this test. Comparatively, Sniper has a very small standard deviation. The large standard deviation indicates that the profit per trader obtained from each experiment are spread far from the mean. Thus, DeepTrader outperforms Sniper in this setting but with a high variability.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	89.71	32.21	84.41	95.01
Sniper	74.28	6.854	73.16	75.41

Table 4.10: Displays results from the One In Many Tests with DeepTrader and Sniper.

4.7 Comparisons with ZIC

4.7.1 Balanced Group Comparison

Table 4.11 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between ZIC and DeepTrader. ZIC's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level the statistical evidence demonstrates that DeepTrader outperforms ZIC in a market where both ZIC and DeepTrader each account for 50% of the market.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	106.7	7.459	105.5	107.9
ZIC	75.37	7.458	74.14	76.60

Table 4.11: Displays results from the Balanced Group Tests with DeepTrader and ZIC.

4.7.2 One In Many Comparison

Table 4.12 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between ZIC and DeepTrader. ZIC's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that DeepTrader outperforms ZIC when ZIC is the widely used strategy and DeepTrader is the minority. However, it is important to highlight that DeepTrader has a large standard deviation for the samples in this test. Comparatively, ZIC has a very small standard deviation. The large standard deviation indicates that the profit per trader obtained from each experiment are spread far from the mean. Thus, DeepTrader outperforms ZIC in this setting but with a high variability.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	366.1	54.69	357.1	375.1
ZIC	224.1	6.113	223.1	225.1

Table 4.12: Displays results from the One In Many Tests with DeepTrader and ZIC.

4.8 Comparisons with ZIP

4.8.1 Balanced Group Comparison

Table 4.13 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the balanced group tests between ZIP and DeepTrader. The mean profit per trader for DeepTrader lies inside of the 90% confidence interval of ZIP. Similarly, the mean profit per trader for ZIP lies inside of the 90% confidence interval of DeepTrader. Therefore, at a confidence level of 90% there is no statistical evidence to suggest that there is a difference in performance between these two traders in a balanced group test.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	245.0	144.7	221.2	268.8
ZIP	263.1	128.9	241.9	284.3

Table 4.13: Displays results from the Balanced Group Tests with DeepTrader and ZIP.

4.8.2 One In Many Comparison

Table 4.14 displays the mean, standard deviation and 90% confidence interval for the profit per trader of 100 samples from the one in many tests between ZIP and DeepTrader. ZIP's upper bound estimate of the mean profit per trader c_2 is less than DeepTrader's lower bound estimate of the mean profit per trader c_1. Therefore, at a 90% confidence level there is enough statistical evidence to suggest that DeepTrader outperforms ZIP when ZIP is the widely used strategy and DeepTrader is the minority. However, it is important to identify that DeepTrader has a large standard deviation for the samples in this test. Comparatively, ZIP has a very small standard deviation. The large standard deviation indicates that the profit per trader obtained from each experiment are spread far from the mean. Thus, DeepTrader outperforms ZIP in this setting but with a high variability.

Trader	$\bar{x}$	s	c_1	c_2
DeepTrader	318.5	63.40	308.1	328.9
ZIP	277.4	12.10	275.4	279.4

Table 4.14: Displays results from the One In Many Tests with DeepTrader and ZIP.

4.9 Summary of Results

Table 4.15 and 4.16 summarises the results by test approach for each trader strategy when compared against DeepTrader. As the table shows, DeepTrader was only outperformed by the AA trading strategy in the balanced group tests. More impressively, in the one in many tests, DeepTrader is shown to have outperformed all traders within the BSE.

It should be noted however, that particularly in the one in many tests the variability is high, suggesting that the DeepTrader outperforms another strategy but typically with a large difference in the amount it outperforms another strategy by.

	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

Table 4.15: Summary of results for the Balanced Group tests.

	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

Table 4.16: Summary of results for the One In Many tests.

4.10 Real World Application

4.10.1 Results

It should be noted that the tests carried out in Section 3.7 do not completely imitate real financial exchanges. Firstly, financial exchanges in the real world consist of a large number of different traders all acting within their own self interest. The trading strategies they use are often kept secret to ensure that they maximise any potential profit they earn and for these reasons, the assumption that an entire real world market can be modelled on only two types of trading strategy is implausible. Secondly, the buyer/seller distributions used for each trading strategy may be deemed to be unrealistic. In light of these points, the results obtained from the tests are promising but should be considered as preliminary. Therefore, whilst it can be concluded that these results present a direct comparison of the performance of DeepTrader against existing trading strategies within the public domain, they do not provide any indication of how DeepTrader will perform in the real world.

4.10.2 Availability of Limit Prices

Section 3.1.2 outlined all the features that were extracted from BSE and used to train DeepTrader's LSTM network. All of this data is publicly available on a LOB based financial exchange, except for a trader's limit price. Naturally, a trader's limit price is always kept secret to maintain a competitive advantage over the other traders to maximise their profits and ensure that their position is not compromised by other traders. Hence, the capturing of limit price information when collecting and compiling the training data for this project can be considered impractical.

Nevertheless, DeepTrader may still be implemented in a real world setting. A real world implementation of the DeepTrader strategy would require a history of (preferably profitable) trades where each of the 14 features listed in Section 3.1.2 are all captured. This could be done by the organisation or individual trader automating the collection of this information as they trade. After a sufficient amount of data (typically millions of data points) has been collected, the DeepTrader strategy can then be used to trade a real security. Whilst trading, the DeepTrader strategy like any other trading strategy is completely oblivious to another trader's limit price.

Chapter 5

Conclusion

5.1 Outcomes

The main achievement of this project has been to devise a deep learning trading strategy that can trade competitively on a LOB based financial exchange against other existing well known trading strategies within the BSE.

With respect to the aims and high level objectives that were introduced in Section 1.5, the potential successes and shortcomings will be evaluated in turn below. To recap, those objectives were:

1. Research and survey literature on existing automated trading agents.
2. Build on work completed as part of the Applied Deep Learning Unit at Bristol, for further research into the use of RNNs.
3. Implement a data pipeline for the collection and pre-processing of training data from BSE modelling a range of trading conditions.
4. Create a profit making RNN which performs well under differing trading conditions. This will include defining its architecture and methodology, and how it should be initialised.
5. Compare the performance of the new strategy with existing automated trading agents under the various trading conditions.

In regards to the first objective, detailed research and survey of the literature on existing automated trading agents was conducted and evidenced in Chapter 1 where an overview into the origins of experimental economics and publicly available automated trading agents was presented.

Chapter 2 went on to provide a detailed summary on the inner workings of all trading strategies that exist within BSE. It gave a more detailed treatment and investigation into the use of DLNNs in Section 2.5 as well as a thorough description of how DLNNs work and how they could be applied for the purposes of this project.

Chapter 3 began by outlining the approach taken to collect data and process it in a meaningful way for a DLNN in the BSE so that it could be used to model a range of trading conditions. It also described the network architecture of the DLNN, defined its input and described how this input will be used to trade. The tests conducted using the DeepTrader DLNN were described in Section 3.7. This included specifying the types of tests that would be used to compare DeepTrader against the other trading strategies within BSE.

Finally, the results from these tests were analysed in Chapter 4. The analysis demonstrated that DeepTrader was a competitive and profitable strategy against most traders in the BSE under most conditions. However, it also revealed that further work is necessary to understand why there was such high variability in profit taking under certain conditions and why there was no appreciable difference for some of the simpler training strategies (e.g. Giveaway).

Recalling the research hypothesis stated in the Executive Summary:

> Does the use of DLNNs - specifically recurrent neural networks (RNNs) - and the additional parameters given by a LOB such as the quantities supplied or demanded at each price, and extended time series information on past bids and offers, let us train and create a trading agent that is comparably better than existing trading agents in the public literature?

and in light of the accomplished objectives, DeepTrader has shown to be comparably better than most of the existing trading strategies under a prescribed set of testing conditions. However, the results presented in this paper do not provide a definitive answer to this question. In order to state this with a high level of confidence, further work is necessary. This will include subjecting DeepTrader to an exhaustive set of comparison tests across a sufficiently wide range of market scenarios and providing a detailed analysis and critique of those findings. [20].

5.2 Future Work

5.2.1 Further Testing

As highlighted in the previous section, a larger, comprehensive number of tests under a variety of different market conditions are need to properly answer the research hypothesis. Market conditions, where the types as well as proportions of different trading strategy, will need to be varied exhaustively to determine how often DeepTrader outperforms existing strategies. Fortunately, with the emergence of cloud computing such as Amazon's AWS, the resources and infrastructure required to conduct a large number of different market sessions are readily available, but extensive work will need to be carried out to configure the environment and data to model the conditions. Given the limited time available, it has been beyond the scope of this project to conduct such large conduct experiments.

5.2.2 Training Data

A well known phrase used within the field of machine learning is "garbage in, garbage out". Essentially, this means that the performance of learning algorithms are heavily dependent on their input data. Put differently, the results gained from the algorithm are only as good as the accuracy and volume of data supplied to it. To this end, an attempt has been made in this project, to provide information from a large number of trades collected from thousands of market sessions consisting of all trading strategies that were available within BSE.

Nevertheless, this may not be the most effective approach for creating a training dataset for DeepTrader. For example, this project used trading strategies that were known to be underperform such as ZIC. In practice, this may skew the data towards realistic scenarios and as as a consequence DeepTrader may only be performing well under this conditions. Therefore, further investigation and analysis needs to be completed as to what strategy or combination of strategies should be used for training DeepTrader.

Moreover, the BSE simulation may have particular biases compared to real world exchanges which inadvertently provide a favourable outcome for DeepTrader's performance. Hence it would be ideal to use training data from other stock exchanges simulations and even the real world.

5.2.3 Feature Perturbation and Sensitivity Analysis

Section 3.1.2 outlines the 13 different features that were used to train the network. One other area of investigation could be to perform a "sensitivity analysis" on these different features. This would involve varying each feature with respect to each other and then analysing the performance of DeepTrader. This for example, could be done in several ways when training the network, such as clamping one of the inputs to its normalised limits (0 or 1) or masking an input with random noise and then re-testing it against other traders. If there is no appreciable change in DeepTrader's performance then that would indicate that a specific feature is redundant.

5.3 Final Words

It is hoped that this project has demonstrated a novel and effective approach of using deep learning to create a profitable trading strategy. The intention is that the ideas from this project can be the catalyst for new research into using deep learning to trade on financial markets which could eventually become the most effective method of addressing the sales trader problem.

Bibliography

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

[2] Charles Cao, Oliver Hansch, and Xiaoxin Wang. The information content of an open limit-order book. *Journal of Futures Markets*, 29(1):16–41, 2009.

[3] Dave Cliff. Minimal-intelligence agents for bargaining behaviors in market-based environments. Technical report, 1997.

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

[5] Dave Cliff. BSE: An open-source limit-order-book exchange for teaching & research. https://vimeo.com/307827599, 2019.

[6] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart, and Gerald Tesauro. Agent-human interactions in the continuous double auction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, page 1169–1176, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] Marco De Luca and Dave Cliff. Human-agent auction interactions: Adaptive-aggressive agents dominate. In *IJCAI International Joint Conference on Artificial Intelligence - Volume 1*, pages 178–185, January 2011.

[8] John Cristian Borges Gamboa. Deep learning for time-series analysis. *CoRR*, abs/1701.01887, 2017.

[9] Wouter Gevaert, Georgi Tsenov, and Valeri Mladenov. Neural networks used for speech recognition. *Journal of Automatic Control*, 20, 01 2010.

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

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8), 1997.

[12] Arthur le Calvez and Dave Cliff. Deep learning can replicate adaptive traders in a limit-order-book financial market. In Suresh Sundaram, editor, *2018 IEEE Symposium Series on Computational Intelligence (SSCI 2018)*, pages 1876–1883, United States, January 2019. Institute of Electrical and Electronics Engineers (IEEE).

[13] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis, Univ. Helsinki*, pages 6–7, 1970.

[14] James Moor. The Dartmouth college artificial intelligence conference: The next fifty years. *AI Magazine*, 27(4):87, December 2006.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] John Rust. Behavior of trading automata in a computerized double auction market. *The Double Auction market: Institutions, Theories, and Evidence*, pages 155–198, 1993.

[17] Hochreiter Sepp. *Untersuchungen zu dynamischen neuronalen Netzen.* PhD thesis, Technische Universität München, 1991.

[18] JPCL & Cardonha CH & Fernandes AA & de Sousa RT Serrano, AM Rubio & Da Costa. Neural network predictor for fraud detection: A study case for the Federal Patrimony Department. In *The International Conference on Forensic Computer Science (ICoFCS)*, pages 61–66, September 2012.

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

[20] Daniel Snashall and Dave Cliff. Adaptive-aggressive traders don't dominate. In Jaap van den Herik, Ana Paula Rocha, and Luc Steels, editors, *Agents and Artificial Intelligence*, pages 246–269, Cham, 2019. Springer International Publishing.

[21] Shyam Sunder and Dan Gode. 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] Christian Szegedy, Alexander Toshev, and Dumitru 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*, pages 2553–2561. Curran Associates, Inc., 2013.

[23] Gerald Tesauro and Jonathan L. Bredin. Strategic sequential bidding in auctions using dynamic programming. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, New York, NY, USA, 2002. Association for Computing Machinery.

[24] Daniel Vach. Comparison of double auction bidding strategies for automated trading agents. *MSc Thesis, Univerzita Karlova, Fakulta sociálních věd*, 2015.

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

[26] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[27] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, Jun 2019.

Appendix A

Appendix

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

AnonymousAuthor1
Department1
Company1
Location1
anony1@email.com

AnonymousAuthor2
Department2
Company2
Location2
anony2@email.com

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 outperform or equal such existing algorithmic trading systems. Thus, we propose that this demonstrates a methodology that can in principle create a copy of an arbitrary trader *T* via "black-box" deep learning methods. In our discussion, we argue that this method could easily be used to cheaply and efficiently replicate the trading activity of a successful human trader.

CCS CONCEPTS

• Artificial Intelligence • Intelligent Agents • Agent Models.

KEYWORDS

Financial Markets; Algorithmic Trading; Automated Traders.

ACM Reference format:

FirstName Surname and FirstName Surname. 2020. Automated Training of a High-Performing Algorithmic Trader. In *Proceedings of ACM Conference on AI in Finance. ACM, New York, NY, USA, 8 pages.* https://doi.org/10.1145/1234567890

1 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 box colored matt-black, 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 <u>*its trading performance matches or exceeds that of the human trader*</u>*. 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][20][26]), 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 [15]) 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 or behavior 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*: [23][24]) 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 is structured as follows. In Section 2 we summarize the background to this work. Section 3 then gives more details of our methods. In Section 4 we present results which demonstrate that DeepTrader learns to outperform many of the pre-existing algo traders in BSE, and matches or exceeds the trading ability of the two "super-human" algorithms AA and ZIP. We discuss our plans for further work in Section 5, and we draw our conclusions in Section 6. Much of the rest of this paper is adapted from AnonymousAuthor1's recent master's thesis [25], which gives full details of all the work summarized here.

2 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 [17]. 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 talk use '*utility*' 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*.

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 'α' 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 been used to test and compare algorithms throughout research on automated trading agents.

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 [16]. 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 negotiations. Once the bid/ask price was within an adequate range, the Kaplan's agent would then enter and "steal the deal". Aptly, Kaplan's program was named *Sniper*. If time was running out in a market session, Kaplan's Sniper was programmed to rush to make a deal rather than not make one at all.

Subsequent to this, in 1993 academics Gode & Sunder published a JPE paper investigating the intelligence of automated traders and their efficacy within markets [19]. 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 (from *eXtended GD*) were published in 2002 by Tesauro & Bredin [21]: 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 [23] and subsequent article in the prestigious *Artificial Intelligence* journal [24]. The key element of AA is aggressiveness: a more aggressive trader places a bid/ask that is more likely to be accepted, where as a less aggressive trader will aim to seek a larger gain. This trading strategy estimates the market equilibrium by using a weighted moving average and calculates 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 *et al.* 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 *at al.* [18] performed a brute force exhaustive search of all possible permutations of different trading strategies, 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 the best choice would be GDX, others in which it is 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* [26] 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 the work of Le Calvez *et al.* [14] which demonstrates preliminary results from the use of a DLNN to successfully replicate the behavior of a ZIP trader.

3 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. Tesauro and Das [REF] present three separate experiment designs for comparing trading agents, two of which will be used here: in *one-in-many tests* 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 *balanced-group tests* involve buyers and sellers being split evenly across two types of strategy. In addition, every agent of one type of strategy has an identical limit price. This test, introduced by the IBM team, is 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:

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 by Vytelingum [23][24].
13) Smith's α metric, calculated from P^* at time t.
14) The price of the trade.

The first 13 items in the list are the multivariate input to the network: if any of them is undefined at time t then zero is used instead of *NaN* or *None*. 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 4 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 (instances). The Python library Boto3 v.1.13.3 [10] was used to create, manage and terminate the EC2 instances. The work was automatically split amongst the instances by making every instance run each market configuration once and via a separate custom utility, created for this project.

Typically, neural networks have train, validation and test datasets. The train dataset is used to train the model and 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, there is no distinct test dataset.

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 [REF].

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 involve typically a rough average of 20 LOB snapshots, so the total number of snapshots used in training was on the order of 15 million.

4 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 (BGT), while Figures 3 and 4 show results from the one-in-many tests (OMT).

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.

Figure 1a shows BGT comparison of APPT scores between DeepTrader and ZIP, and Figure 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 Figure 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.

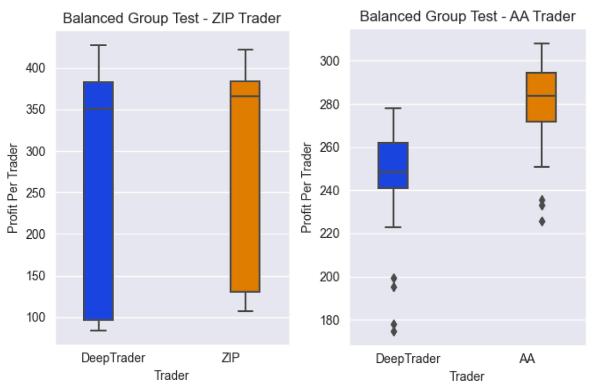


Figure 1: box-plots showing average profit per trader (APPT) from balanced-group tests (BGTs) for DeepTrader vs ZIP algorithmic traders (left: Fig1a) and AA algorithmic traders (right: Fig1b).

Figure 2a shows BGT comparison of APPT scores between DeepTrader and GDX, and Figure 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.

Figure 3a shows OMT comparison of APPT scores between DeepTrader and ZIP, and Figure 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.

Figure 4a shows OMT comparison of APPT scores between DeepTrader and GDX, and Figure 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.

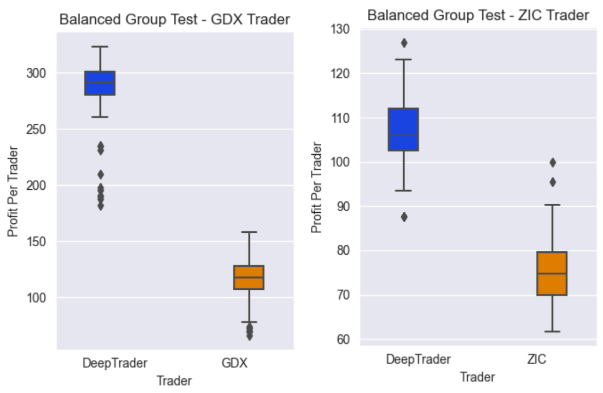


Figure 2: box-plots showing APPT from BGTs for DeepTrader vs GDX algorithmic traders (left: Fig2a) and ZIC algorithmic traders (right: Fig2b).

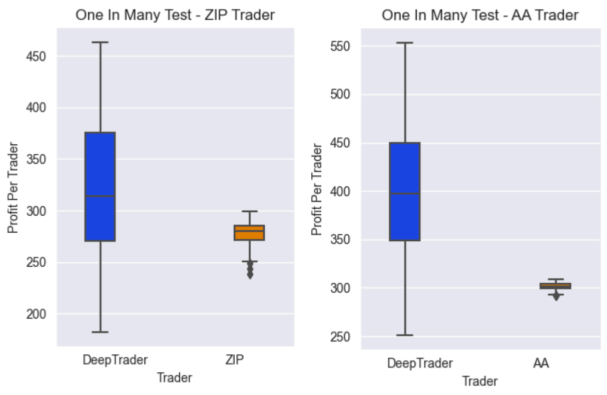


Figure 3: box-plots showing average profit per trader (APPT) from one-in-many tests (OMTs) for DeepTrader vs ZIP algorithmic traders (left: Fig3a) and AA algorithmic traders (right: Fig3b).

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 [25], 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.

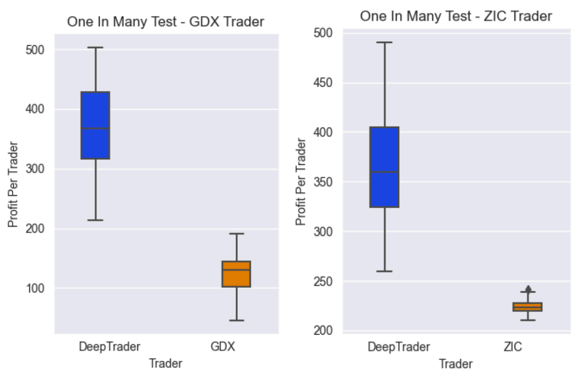


Figure 4: box-plots showing APPT from OMTs for DeepTrader vs GDX algorithmic traders (left: Fig4a) and ZIC algorithmic traders (right: Fig4b).

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 and in [25] 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: we turn to discussion of our plans for future work in the next section.

5 Future 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 3 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.

6 Discussion and Conclusions

In this paper we have addressed 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 continuous double auction CDA process mediated by a continuously updated limit order book (LOB) showing Level2 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.

ACKNOWLEDGMENTS

The work reported here was made an awful lot easier by the use of public-domain open-source software that is free to use, specifically *Boto* and *BSE*. We are grateful to M. Garnaat for creating Boto, a Python package for automating use of Amazon Web Services, the cloud provider on which we ran our compute intensive experiments; and we are grateful to D. Cliff for initially creating the BSE market simulator we used, and also to the many other people that have contributed to the BSE project since it was first released in 2012, in particular D. Snashall who uploaded his code for AA and GDX to the BSE GitHub repository.

REFERENCES

[1] Marco 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] Charles Cao, Oliver Hansch, and Xiaoxin Wang. The Information Content of an Open Limit-Order Book. *Journal of Futures Markets*, 29(1):16–41, 2009.

[4] Dave Cliff. *Minimal-Intelligence Agents for Bargaining Behaviors in Market-Based Environments*. Technical Report, Hewlett-Packard Laboratories, 1997.

[5] Dave Cliff. BSE: An Open-Source Limit-Order-Book exchange for teaching and research. In *Proc. Computational Intelligence in Financial Engineering, Symposium Series on Computational Intelligence* (SSCI), pp1853–1860, 2018.

[6] Nancy Cooke. Varieties of Knowledge Elicitation Techniques. *International Journal of Human-Computer Studies,* 41(6), 1994.

[7] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart, and Gerald 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] Marco De Luca *et al.* Human-agent auction interactions: Adaptive-aggressive agents dominate. In *Proceedings International Joint Conference on Artificial Intelligence (IJCAI2011), Volume 1*, pages 178–185, January 2011.

[9] John Cristian Borges Gamboa. Deep Learning for Time-Series Analysis. CoRR, abs/1701.01887, 2017.

[10] Mitch Garnaat. *Python and AWS Cookbook: Managing Your Cloud with Python and Boto.* O'Reilly, 201..

[11] Steven Gjerstad and John Dickhaut. Price Formation in Double Auctions. *Games and Economic Behavior*, 22(1):1–29, 1998.

[12] Sepp Hochreiter and Jurgen Schmidhuber. Long Short-Term Memory. *Neural Computing*, 9(8), 1997.

[13] Diederik Kingma and Jimmy Ba. Adam: A Method For Stochastic Optimization. *Proceedings ICLR*, 2015.

[14] Arthur Le Calvez *et al.* Deep Learning can Replicate Adaptive Traders in a Limit-Order-Book Financial Market. In Suresh Sundaram, editor, *2018 IEEE Computational Intelligence in Financial Engineering: Symposium Series on Computational Intelligence (SSCI 2018)*, pages 1876–1883, United States, January 2019. Institute of Electrical and Electronics Engineers (IEEE).

[15] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323:533-536, 1986.

[16] John Rust. Behavior of Trading Automata in a Computerized Double Auction Market. In *The Double Auction market: Institutions, Theories, and Evidence*, pages 155–198, Addison-Wesley, 1993.

[17] Vernon L. Smith. An Experimental Study of Competitive Market Behavior. *Journal of Political Economy,* 70(2):111–137, 1962.

[18] Daniel Snashall *et al.* Adaptive-Aggressive Traders Don't Dominate. In Jaap van den Herik, Ana Paula Rocha, and Luc Steels, editors, *Agents and Artificial Intelligence,* pages 246–269, Springer, 2019.

[19] Shyam Sunder and Dan Gode. 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.

[20] Christian Szegedy, Alexander Toshev, and Dumitru 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.

[21] Gerald Tesauro and Jonathan L. Bredin. Strategic Sequential Bidding in Auctions Using Dynamic Programming. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, New York, NY, USA, 2002. ACM Press.

[22] Daniel Vach. Comparison of Double Auction Bidding Strategies for Automated Trading Agents. MSc Thesis, Charles University at Prague, 2015.

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

[24] Perukrishnen Vytelingum *et al.* Strategic Bidding in Continuous Double Auctions. *Artificial Intelligence*, 172(14):1700-1729, 2008.

[25] AnonymousAuthor1. Master's Thesis, Department of Computer Science, Anonymous University, Completed May 2020.

[26] Zihao Zhang, Stefan Zohren, and Stephen Roberts. DeepLOB: Deep Convolutional Neural Networks for Limit Order Books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, Jun 2019.