Dissertation Type: Research

DEPARTMENT OF COMPUTER SCIENCE

An Exploration into Generalisable Deep Learning Trading Agents

Owen Coyne

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 21st May, 2021

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.

Owen Coyne, Friday 21st May, 2021

This project did not require ethical review, as determined by my supervisor, Dave Cliff

Contents

List of Figures

List of Tables

Executive Summary

The rise of algorithmic trading and deep learning within their respective fields is a well-documented matter. The success of algorithmic trading agents has not only brought about an arms race of algorithmic development but completely restructured the financial industry and the means by which financial exchanges operate. On the other hand, deep learning has shown a valuable tool in tackling a wide range of the modern world's most complex problems. Utilising large swaths of data and massive amounts of computing resources to spot trends and make predictions which can far exceed human ability. The fusion of these two concepts, then, draws increasing interest as a chance to utilise the vast amounts of data produced by financial exchanges to create robust and adaptive trading agents.

The application of Deep Learning Neural Networks (DLNNs) to the field of algorithmic trading has shown tentatively promising results in the works of "Deeptrader" and "Deeptrader 2". These results, however, exist within a series of constraints defined by the methods used to create and test both trading agents. The exploration of such constraints and their relation to the applied field of algorithmic trading, that exists within major financial exchanges, is the focus of this project. More specifically, this can be stated as:

> My research hypothesis is that existing algorithmic trading strategies that utilise DLNNs possess the capabilities to generalise to a wider range of more realistic market scenarios. This generalisation would strengthen the case for their utility in real-world financial exchanges.

This research hypothesis serves the basis for the work undertaken in this project. In an attempt to examine this hypothesis, this project serves to employ a range of skills gathered over the course of this Master's degree whilst introducing a range of relevant concepts from contemporary literature. In the pursuit of this goal, a number of notable contributions have been made (some of which are listed here).

- I independently implemented the work presented in the dissertations of Wray [55] and Meades [37]. This consisted of the Deeptrader and Deeptrader 2 networks. This required the creation and training of LSTM neural networks.
- I altered the base structure of Bristol Stock Exchange [8] to provide an exhaustive simulation of unique market scenarios and thus collect data to train the networks. This was supplemented by a further alteration of another instance of Bristol Stock Exchange to serve as an experimental environment that provided key statistics.
- I created a number of experiments to highlight the network's ability to generalise to new and unique market scenarios. This was furthered through the use of a range of data manipulation and statistical methods to assess the performance of the networks within that wide range of scenarios.
- I co-authored a paper (Wray/Coyne/Meades/Cliff) based on some of the findings made here in conjunction with previous work on the subject in the form of [56]. This paper has been submitted for review to the "Journal of Banking and Financial Technology" and is included within this works appendix.
- I am currently in the process of co-authoring a conference paper (Coyne/Cliff) based on a number of the later findings within this work. The intention of this paper is submission to the "International Conference on Agents and Artificial Intelligence (ICAART)".
- I intend, once examination of this work is complete, to publish the code and datasets I have created for this project as a GitHub repository. This will allow other researchers to replicate and extend the work presented here.

Supporting Technologies

The specific technologies utilised in the completion of this project are listed here. More generally across the project, the work was completed with the Python language and the listed libraries were managed with the Anaconda framework.

- The open-source market simulation Bristol Stock Exchange [9] was used to design and evaluate the experiments presented within.
- The Python library Captum [31] was used to apply a range of deep learning interpretability methods.
- An open-source simulation of Generalised Brownian Motion with Drift was utilised for data collection and testing, sourced from the GitHub repository [45].
- The Python libraries PyTorch [41], TensorBoard [2] and Dask [13] were utilised in the implementation of deep learning methods.
- General data manipulation and visualisation was achieved with the Python libraries: Numpy, Pandas, Seaborn and Matplotlib.
- Intensive computation was undertaken using the University of Bristol's high performance computing machine: BlueCrystal 4 (`https://www.acrc.bris.ac.uk/acrc/phase4.htm`).

Acknowledgements

I would first like to thank my supervisor, Dave Cliff, for his continued guidance and insight throughout the course of this project. In addition, I couldn't have done this work without the support and love of my family and friends. Finally, for anyone who has had to listen to me try and explain my dissertation, thanks for bearing with me.

Chapter 1

Contextual Background

Over the past 30 years it has become an undeniable fact that algorithmic trading is the dominant force across the world's financial markets. Example figures from the European Central Bank illustrate that 50% of all trades in the US equity market are the result of High Frequency Traders (HFTs) [4]. HFTs are a type of algorithmic trader used to automate the buying and selling of various stocks for private companies. Given this proportion we can safely say that algorithms are currently playing a central role determining what happens in an industry worth billions of dollars which underpins global finance.

This makes clear the case for a thorough and academic understanding of these algorithmic entities, however the reality is an industry composed of businesses using their own proprietary trading algorithms. The functionality and design of these traders is of the utmost secrecy in the interest of profits. Whilst this secrecy is understandable, it results in a gap in knowledge regarding these algorithms, where only individuals inside a company understand how they work.

This is the most pressing issue in the study of algorithmic trading, a one-way flow of information from literature to industry. These companies may well be utilising publicly studied methods but the resulting performance in real scenarios then becomes secret. This results in a body of work characterised by its disconnect from real utility, existing purely in academia.

The attempt to justify these academic methods as relevant to real world scenarios is the focus of a lot of work involving modelling and simulation. When dealing with this kind of justification, it can be considered that a new, more accurate model of the world's behaviour is just as important as an algorithm that exploits it. The literature proposing both models and trading algorithms can be broadly split into two distinct categories, the first of which covered here can be considered to have a direct lineage to the pioneering experimental economics work of Vernon Smith.

Experimental In the seminal 1962 publication in the Journal of Political Economy [49], Smith devised a series of experiments that aimed to simulate a real trading environment. At that time a real trading environment would be a trading floor with traders shouting orders between each other, the experiments aimed to simply replicate this. The structure of these experiments was a set of market sessions conducted under a continuous double auction (CDA) with real human traders as test subjects. The CDA is a method of trading used by all the world's major financial institutions and thus serves as a good basis for testing. Within an experimental CDA, the traders would be separated into two groups, of buyers and sellers, each aiming to trade a (fictional) stock whilst turning a profit.

Smith carefully designed these experiments by giving each trader different limit prices, so called because they define the limit which a trader cannot exceed when trading. This mimics the real world scenarios encountered by sales traders where customers engage them to buy or sell shares with a price cutoff point. Given the careful setting of these limits, Smith could manipulate the supply offered by sellers and the demand of the buyers.

Once the session was over and all the trades had been made - the resulting behaviour was recorded, with the profitability and individual actions of the traders examined. These experiments found that without any guidance traders will alter their prices over the course of the session. These alterations in

price by all the traders converged to a value known as the equilibrium price. This equilibrium was a price and quantity point, predetermined by Smith but unknown to the traders, where the supply and demand in the market were equal. This so called "Equilibriation" of the market occurred consistently and quickly, even when Smith altered the supply and demand. This was a key result in the study of trading and the publication birthed the field of experimental economics. It is however, more importantly, the concept of a contained market simulation that truly inspires the algorithmic trading work to come.

The next major step would come in 1993 with a study by Gode and Sunder into the "Allocative Efficiency of Markets with Zero-Intelligence Traders" [50]. The pair attempted the same setup as Vernon Smith but their goal was to see what would happen if simple trading algorithms replaced the humans in the experiments. The algorithms they used were two types of "zero-intelligence" trader, meaning they just randomly generated prices to quote to the market.

The two types consisted of the zero intelligence unconstrained (ZI-U) trader and the zero intelligence constrained (ZIC) trader. The difference being that the ZIC trader had a built-in bound so that it could not generate prices on the wrong side of its limit price, this ensured that it wouldn't quote prices at a loss (the ZI-U trader had no such guarantee). The study showed the obvious issue with the ZI-U trader, as it regularly made a loss. Its other, quite remarkable, finding was that a market composed of ZIC traders would act almost indistinguishably from the human traders in Smith's earlier experiment. These random traders, with no decision making capabilities, managed to repeatedly converge to the point of Equilibrium. The implication of this work in their eyes was that intelligent behaviour was a byproduct of the structure and rules of the financial market itself, not of the individual trader.

The bold nature of this claim led to an investigation by Cliff presented in a 1997 technical report [7]. Cliff discovered that the convergence of ZIC traders was an artifact of the experimental design and such results couldn't be replicated in scenarios with a different setup of supply and demand. This was another expansion towards realism in the market experiments, adding a range of supply and demand setups allowed the testing of scenarios that weren't present in the original two works.

This spurred Cliff to create the Zero Intelligence Plus (ZIP) trading algorithm. The ZIP trader was the first in this line of work to include a primitive machine learning technique. As you can imagine, the step-up from algorithmic traders which simply generated random prices, to those which made rudimentary decisions to optimise profit, was a watershed moment.

A litany of algorithmic traders were proposed and studied in the years following but the next major result in this space came in 2001 with "Agent-human interactions in the continuous double auction" [12]. The authors, Tesauro and Das, displayed a modified version of the Gjerstadt-Dickhaut trading algorithm [20] which they dubbed Modified Gjerstadt-Dickhaut (MGD). The real backbone of this paper however was a comparison between a series of algorithmic trading agents. This comparison was undertaken in a trading environment, inspired by the experiments of Smith, aiming to mimic a real market. The critical results were those pertaining to the ZIP and MGD traders, both such agents consistently outperformed human sales traders across the range of experiments. The advent of "super-human" algorithmic traders at that time rather presciently spelled out the future for human traders participation in financial markets.

In 2013, Cliff proceeded to create the Bristol Stock Exchange [9]. This stripped-back simulation was designed as an educational tool but served another purpose, the structure of this simulation mirrored that of Smith's seminal market experiments with a few modern additions. One such addition was the introduction of a limit order book (LOB), the trading agents within BSE interacted by posting their bids and offers, to buy and sell stocks, to a public LOB. This LOB allowed participants to view all the current trader activity and the potential for trades to be made. The BSE framework had implementations of all the major trading algorithms discussed above and a freedom to tweak market variables to run experiments. Therefore BSE presented a contained environment in which to test hypothesises about algorithmic trading, birthing a whole sub-genre of algorithmic trading literature, of which this work is a part.

Market Microstructure At the same time, in parallel, a body of work emerged whose immediate goal was an accurate mathematical modelling of financial systems. This field, born of quantitative finance, was chiefly interested in "the study of the process and outcomes of exchanging assets under explicit trading

rules" [19] and is commonly referred to as the study of market microstructure. An excellent in-depth review of this work is present in [5] but an overview of key concepts exists here.

Classic examples of these methods include the optimal trading strategies presented in works such as [39], an attempt to model supply and demand dynamics to find a mathematically optimal series of trades. Other branches include optimal liquidation [3], which considers the most efficient way to sell a block of assets. These examples may not be directly related to the aims of this work but the mathematical models which underpin them are.

The vast majority of these methods aim to represent financial markets as a series of interconnected stochastic events, the best approximation to series of activities that most would describe as seemingly completely random. They constructed these models of market behaviour from a range of mathematical processes which are linked through complex differential equations. These representations have an accuracy that far exceeds that of the hand designed experiments which so many traders are tested on. And yet, perhaps as a function of their difficulty, this style of modelling has yet to fully permeate the wider work of trading agents. We see some hints of it in the use of stochastic processes in certain trading works [9] but much of this work has yet to be adopted.

One of the major caveats of this body of work is their evaluation methods. Existing outside of a rigid simulation framework typically results in performance being tested according to historical market data. Whilst this historic data has an impressive resolution and obviously represents the stochastic nature of price movement in markets better than any simulation, it suffers from a few key issues.

The first is the obvious point that the activity of the algorithmic trader can't impact the price movement and thus any measure of performance in said market is an estimate at best. Another more subtle point of contention is the wide range of sources of historical market data. Whilst some are more popular than others, there is no standard and thus the comparison of methods becomes challenging under the likelihood that results are hand-picked to only show situations where the method under evaluation performs well. The lack of an agreed baseline test is at a detriment to the range of literature in which historical data is used for evaluation.

1.1 Deep Learning Trading

Given its great success in a number of tasks, deep learning as a methodology has been systematically applied to a number of real world problems which have historically been the domain of subject experts, or those tasks which remain mathematically complex to this day. Finance is one such field currently being scrutinised through the lens of deep learning. This seems like a logical step, as we have seen that its mathematical complexity is a matter of continued study. More importantly, financial markets satisfy a key quality of scenarios which benefit from deep learning: they produce vast amounts of data on which to condition models.

We must briefly consider a large array of work whose central pillar is stock price trend prediction. This at first seems like a different goal than the application of deep learning to create a trading agent, as noted by Le Calvez in his 2018 Dissertation [32]. It is however reasonable to argue that the prediction of a stock's price movement, if accurate, is only one step abstracted from a profitable algorithmic trader through some simple alterations. Thus, it seems apt to attempt to glean any potentially useful concepts from this area.

There have been a host of approaches within the supervised learning space in relation to stock price prediction. A number of works aim to utilise the unique nature of convolutional neural networks to extract high level features from complex market information: in the case of Gudelek et al. [22] this information takes the form of a series of market statistics. Some works, such as DeepLOB [58], extract these features directly from the current state of the limit order book. DeepLOB then proceeds to input these features into an LSTM network to capture temporal dependencies. This conceptual separation between the parsing of the market structure and the detection of trends over time appears across a number of works.

When examining trends, in the form of time series forecasting, the reputation of the recurrent neural network precedes itself. In a 2020 survey, on the use of deep learning methods in stock market prediction

[26], it becomes clear that RNN methods dominate the published literature - representing around 40%-50% of published work. Of these RNN methods, a comparison of performance against google stock price prediction placed the LSTM on top. In a head-to-head with basic RNN models and Gated Recurrent Units, the LSTM achieved the best performance with a 72% accuracy over a five-day window [15]. These achievements are further augmented by the introduction of attention mechanisms, famed for their high performance in a wide range of problem domains [33][38].

This application of general deep learning methods seems the norm however an under-explored avenue within the literature presents itself in the 2016 work of Sirignano [48]. Here we are presented with a neural network structure which has been hand designed to exploit the spatial structure of the limit order book. This concept, dubbed the "spatial neural network" provides a more interpretable and less computationally intensive alternative to the standard architectures. Thus, we can see the benefit that an expert understanding can impart on this specific sub-field of deep learning applications.

In a more directly applicable area to algorithmic trading agents, we see the rise of deep reinforcement learning techniques allow the modelling and traversal of complex real world situations in-situ. Techniques such as deep Q-networks have been adapted directly to the the actions of the trading agent, as opposed to the stock price prediction. In Xiong 2018 [57] this methodology "outperform[s] the two baselines in terms of both the Sharpe ratio and cumulative returns". The Sharpe ratio being a measure of trading performance when compared to the returns of a risk-free investment. This performance is further reinforced in the 2019 work of Yang Li [34] in which their trading agent outperforms the same baselines and "achieves stable risk-adjusted returns in both the stock and the futures market".

More specific to our area of interest is the series of papers which study the specific application of deep learning to an algorithmic trader within the context of Bristol Stock Exchange. This work began in 2017 with an attempt by Tibrewal [52] to replicate the behaviour of the ZIP trading algorithm with a neural network. This was furthered in 2018 with Le Calvez [32] presenting a neural network backed trader which had been trained using a single ZIP agent. This limited prototype proved able to out-perform the ZIP trader from which it had learned. This spurred Wray to expand upon this in his 2020 dissertation [55]. This improved trader included a wider range of trading data and an increased complexity as a result of the inclusion of LSTM units provided with a wider array of features. Impressive performance continued in this manner with the appropriately named "Deeptrader" outperforming all but one of the Bristol Stock Exchange's native traders. In the evolution of "Deeptrader" we must then consider the 2020 thesis by Meades [37]. This supplementary work investigated the performance of the "Deeptrader" networks in a number of contexts, highlighting the various avenues of inquiry worth further undertaking.

1.2 Deep Learning Interpretability

The successes of deep learning have come under increasing scrutiny over the last few years. Their state of the art performance comes at the cost of networks containing millions of parameters where the functionality of the model is, even for the developers, a black box system. Few models incorporate means to explain their predictions, with most citing testing performance as a justification of their success.

This attitude has led to a number of real world consequences. The most notable being COMPAS. The COMPAS system employed deep learning techniques to predict the probability of recidivism in an individual. This prediction was used as evidence in sentencing and parole decisions across the U.S. justice system. It was later uncovered by a series of journalists that the system had a clear bias against African-Americans in its predictions [27]. Further independent investigation was conducted showing that the system performed worse than a human on average [17]. The study also showed a similar performance could be replicated using a far simpler logistic regression model. The true extent of the COMPAS model's failings remains unclear as all public studies have only been given access to seven of the model's 137 features. This paints a clear picture of just one of a plethora of deep learning models whose internal workings are obscured under the mantle of proprietary information. The effects these black-box systems can have are clearly not just a matter of academic study but increasingly of serious importance to peoples' lives.

Closer to this work's area of interest are a number of incidents caused by errant behaviour in algorithmic traders on financial markets. These incidents do not directly relate to deep learning but indicate the real world damage caused by critical application of complex and poorly understood systems. The "flash

crash" of 2010 exemplifies this scenario, in the space of under an hour upwards of 850 million dollars were wiped from the U.S. stock market. A later investigation found that a completely automated sale of $4.1 Billion dollars worth of stocks caused a major market imbalance, inciting high frequency traders across the market into a trading frenzy [29]. A later report from U.S. financial regulators[1] stated that no single error was the cause of the flash crash. Rather it was the interconnected actions of a series of high frequency trading agents. This complex relationship between trading entities was further explored in a UK government technical report which argued that these disparate entities need to be examined from an ultra large scale systems perspective and that engineering methodologies need to be designed to cater to this new field [11]. This is further solidified by the quote "The software and hardware that control financial markets have become so complex that no individual or group of individuals is capable of conceptualizing all possible interactions that could occur among various components of the financial system" [30].

Given the demonstration of the dire need for methods to understand both deep learning (and other black-box) systems, we can also see how potential methods can benefit the use of these systems. Two separate studies have indicated that providing some form of explanation with a prediction can increase the rate at which human users accept the prediction [18][24]. Given this a number of frameworks have been created which attempt to approximate some explanation to the decision-making process of a deep neural network. One such example is the LIME method [42], this method takes any specific output of a network and learns a classifier using that prediction which aims to weight the importance of the different features in the prediction. This concept was further explored in [35] in an attempt to unify a number of interpretability methods. The key to this work was an explanation system that was trained across whole datasets, not specific instances. This generality allows the user to account for how different features are of varying in different examples.

1.3 Summary

The goal of this dissertation can thus be considered as the continued exploration of the role of deep learning within algorithmic trading. In doing this it is the hope that incremental influence by two strands of literature, representing both experimental economics and the study of market microstructure, can seek to alleviate the respective pitfalls of both. Furthermore, when considering any work with real world implications where deep learning is applied: we can see that it is imperative to strive to provide explainable and reasoned exposition regarding the decision making process. This will be achieved in the broad steps determined here:

- Re-implement the previous work by Wray and by Meades, in the form of the Deeptrader and Deeptrader 2 networks.
- Apply the conceptual spectrum of deep learning interpretability methods to provide a reasoned understanding to the performance and behaviour of the network. This analysis will include, but is not limited to, the furthering of the results achieved by Meades' investigation into the role of Deeptrader's input features.
- Explore the ability of the Deeptrader network to generalise to new, more relevant market scenarios. This will be achieved through the application of concepts from the broader literature surrounding algorithmic trading to the BSE market environments.
- Perform a confidence interval examination of the work presented to draw statistically relevant conclusions and serve to provide a cross-compatibility with the results presented in the previous literature.

Chapter 2

Technical Background

2.1 Deep Learning

A brief summary of the nature of deep learning and some of the more specific aspects of neural networks is included here as a prelude to the technical background surrounding the original Deeptrader network.

Deep learning is a field, within the broader work of machine learning, whose common unifying characteristics are the application of large quantities of data to achieve a wide array of tasks via the use of deep neural networks. More specifically the branch known as supervised learning, employed here, aims to find a mapping between a series of inputs to some relevant output. This is done by providing the network with "labelled" data that includes examples of inputs and their respective outputs.

A neural network, then, consists of a number of "neuron" nodes in various layers. The nodes in each layer carry weighted connections to one or more of the neurons in subsequent layers. Each of these neurons takes the inputs from the previous layer, applies an activation function and then outputs the resulting value: this pipeline can be seen in Figure 2.1. These values propagate throughout the network from the input layer, via one or more "hidden" layers, to a final output layer. Given the labelled data this method uses, we can thus see the difference between the output of the network and the provided target output.

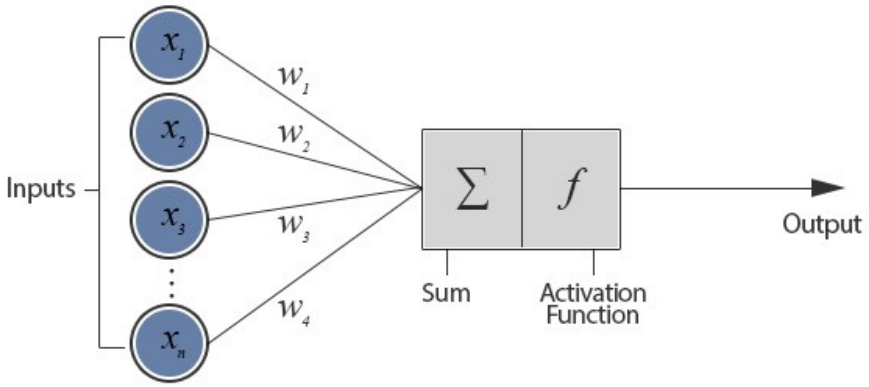

Figure 2.1: A diagram illustrating the input and output of a single artificial neuron, from [25].

The task of how this difference in outputs is quantified falls to the loss function, the output of which is known as the loss of the network. After each example is run through the network we can use the resulting loss in combination with the backpropagation algorithm to alter the connection weights.

The loss function being used in this work is something of a standard within deep learning tasks and is called Mean Squared Error (MSE) loss, the Equation for MSE loss is shown in Equation 2.1. Where y_i and $\hat{y}_i$ represent the network output and the true labelled value (target) respectively (with N equal

to the number of samples present). Thus, MSE loss quantifies the average squared difference between predictions and their true values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.1}$$

The backpropagation algorithm differentiates the loss function and each weighted connection in the network such that the loss can be attributed in portion to each connection, essentially answering the question "How much did each weight in the network contribute to the final loss?". This loss attribution is usually scaled up or down to result in actual weight changes by a parameter known as learning rate, which can be thought of as a measure of how rapidly changes are made to network weights.

The network is therefore iteratively shown examples from the data, followed by weight changes, in a process known as training the network. Having shown all samples from the dataset to the network is known as a single epoch and, in an attempt to minimise loss, most neural networks are trained for many epochs. It is worth noting however that training for an extended period when combined with certain network design decisions can lead to a phenomenon known as overfitting. Overfitting occurs when the network is too closely fitted to the provided data at the cost of its ability to generalise well to performance on real fresh data.

LSTM

Recurrent neural networks (RNN) are a heavily utilised range of deep learning techniques which maintain recurring connections within the network to propagate weights forward across multiple steps of training. This behaviour makes this style of network uniquely suited to processing sequences of data, such as time series, this is because the information from previous inputs is retained and thus the instances that precedes the current prediction can help influence a more accurate output for the network.

RNNs often struggle to process longer sequences of data. This is due to the fact that as weights propagate over longer sequences of time, the derivatives of these weights during backpropagation can create a compounding effect where the weights either "vanish" or "explode". These rapid changes over time massively upset the accuracy of training and bring us to the need for Long Short-Term Memory units.

The goal of a Long Short-Term memory unit is to provide a route throughout sequences where the derivative weights remain stable. This is primarily achieved by creating a self-loop within the unit which helps maintain information over long periods of time. The control of this self-loop via a gate also allows this repeat information to be "forgotten" as needed, allowing the ability as named to capture both short and long term dependencies. This has made it a highly popular option when attempting deep learning tasks on time series data.

2.2 Market Theory

The continuous double auction (CDA) is the *de facto* standard of operation across the world's major financial exchanges. The structure of a CDA can be thought of as a fusion of an open-ascending (English) and a descending auction (Dutch), where buyers and sellers can at any time post bids (to buy) and offers (to sell) at various prices for an asset. The crucial information regarding the state of the market is often provided in the form of a limit order book. The limit order book shows, on one side, the price and quantities at which traders bid to purchase the asset: in price descending order from the highest bid. This is mirrored on the other side of the LOB with ascending offers and quantities.

This structure allows a trader, from a cursory glance, to identify the current best bid and best offer-/ask. It also provides a rudimentary understanding of the distribution of prices within the market and the quantities available for purchase and sale. When a trader makes a bid higher than the current best ask, a sale is initiated at the price of the best ask (and vice versa) - this is known as a market order. A trader may also simply post a price at which they are willing to trade at (and which they will not

XYZ			
Bid		Ask	
10	152	155	20
60	150	162	50

Figure 2.2: Visualisation of an example limit order book, with Best Bid/Ask at the top of the book [9].

surpass) to the limit order book, this is known as a limit order.

The price of trades within a market can be used to construct supply and demand curves. These curves indicate the varying will to trade at different price points in a market for buyers and sellers, an example can be seen in Figure 2.3. The point at which these curves intersect is known as the competitive equilibrium, the theoretical price point where supply and demand are equal. Equilibriation within a market refers to the actions, of traders in a market, causing a convergence to this theoretical equilibrium state. In real markets this point is not static, the equilibrium is constantly shifting as a result of the transient interplay between supply and demand. The external factors which effect the willingness to trade are too numerous to simulate but some approximation can be achieved through the manipulation of customer limit orders.

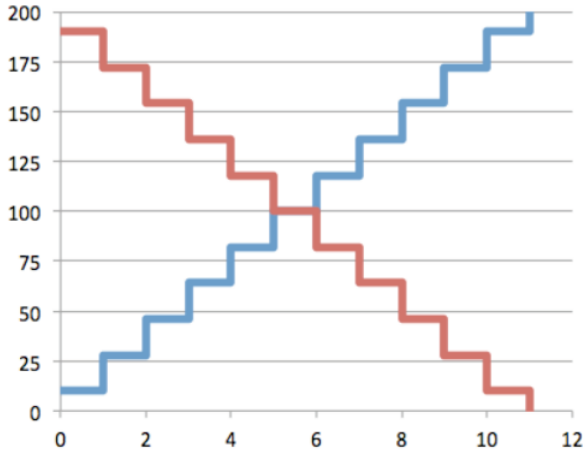

Figure 2.3: An example supply and demand curve with supply in blue and demand in red [9].

2.3 Bristol Stock Exchange

Bristol Stock Exchange (BSE) is an educational tool created by Cliff for the purpose of designing and conducting experimental economics studies [8][9]. It serves as an abstraction of a real financial exchange where the remaining functionality serves to provide robust and reproducible experiments.

BSE models a continuous double auction exchange where a single financial instrument is traded over a simulated time period. This time period is modelled sequentially such that at each time step all traders are given a chance to act. The order in which this action takes place is determined randomly every simulated second.

Within this exchange there exists a number of traders who are categorised as buyers or sellers. At varying intervals of time these traders are issued limit orders from their fictional customers. From there each trader will, according to their trading strategy, quote a price at which they wish to make a limit order. The difference between their customer limit order and their quoted price is where these traders aim to extract profit.

These orders are then posted to the central limit order book and BSE provides an opportunity for all traders to react to the change in the LOB. These traders can quote new orders based on this reaction which cancels any previous order they have made. To avoid the complexities of block orders, a limit order of quantity one is the only currently supported type of order within BSE.

The simplicity of BSE in comparison to a contemporary financial exchange is its elegance and provides a solid foundation on which to alter market variables and dynamics, providing valuable statistical data and thus serving as a basis for a number of academic publications.

2.3.1 Algorithmic Trading Agents

The traders discussed above exist within BSE as a series of automated algorithmic trading agents. Those agents implemented within BSE have not only been widely studied but also many can be classified as high performing trading agents. The performance and functionality of the work presented is compared to these trading agents throughout and thus seems pertinent to provide a brief summary of each agent below, the accompanying implementation code will be included in an appendix.

Giveaway

The Giveaway trader was first introduced in the original release of BSE [8], it is one of the simplest strategies and as a result is often less profitable than counterparts. Any particular giveaway trader will simply quote it's customer limit as the price to trade at. Thus, profit can only be extracted in situations where the limit price is better than the current best bid or ask price, this profit will be equal to the delta between these two values. This is because a bid that improves on the best bid or ask on the opposite side of the LOB will execute at that best posted price.

ZIC

The ZIC trader within BSE is an implementation of the zero intelligence constrained trader introduced by Gode and Sunder [21]. The aforementioned constraints refer to the customer limit price and the maximum trading price of BSE. These constraints are a bounds within which a ZIC trader quotes a randomly generated price. These bounds allow ZIC to never make individual trades at a loss as compared to it's predecessor the ZIU trader.

Shaver

Moving onto a class of traders which actively react to information from the limit order book we can now examine the simplest of these in the form of Shaver, another novel trader included in the original release of BSE [8]. Shaver aims to always be positioned as the best bid or ask on the limit order book. It achieves this by "shaving" the current best quotes on the LOB by adding or removing a single unit from their prices. The constraint of the customer limit price persists here (as it does with all subsequent traders), causing shaver's activity to cease when the current LOB values exceed its limit price.

Sniper

The sniper trading agent present in BSE is heavily influenced by a trading strategy known as Kaplan's Sniper. The strategy, famed for its winning performance in the 1990 Santa Fe Double Auction Tournament [44], aimed to have an initial period of low activity followed by increasing action leading up to market close. This is emulated in BSE by a trader which initially acts like the Shaver strategy (altering the best price by one unit) but over time within the market the amount which is shaved from the best prices is increased rapidly, leading to large price alterations near market close.

ISHV

The ISHV agent, as presented in the work of Church and Cliff [6], aims to extend the Shaver trading agent to incorporate a comprehension of the imbalance of the LOB. This LOB imbalance is calculated in a rudimentary fashion by finding the difference between the current midprice and microprice. This calculation functions as a representation of imbalance because the microprice takes into account the

quantity of orders as well as their price, unlike the midprice. This difference is then used to scale the amount shaved from the current best bid/ask for each order.

PRZI

The Parameterised-Response Zero-Intelligence trader [10] can be considered as a means to unify the distinct concepts present within the Shaver, Giveaway and ZIC traders. This trader generates quoted prices according to a strategy which is parameterised by the user between the values of [-1,+1]. At 0 this strategy is equivalent to ZIC, with a probability function that gives equal likelihood of all quoted prices within the trader's bounds. Skewing this probability function, by means of tending towards -1 or 1 allows the trader to generate prices within a lopsided range, generating prices skewed towards either the best bid/ask on one end or the limit price on the other. Thus, acting as the strategies Shaver or Giveaway at the extremes of this value range. This allows a trader which can be allocated a strategy that exists on the spectrum somewhere between each of the 3 strategies.

ZIP

Zero-Intelligence Plus is another trader created by Cliff and detailed in a 1997 technical report [7]. This strategy works by quoting a price that is the direct result of the customer limit price multiplied by some margin. The key to ZIP however is that it incorporates a simple machine learning rule to update this margin. In response to each action on the LOB, a decision tree is executed that alters the margin according to the current state of the LOB, incorporating information such as the best bid/ask at different points. Thus, ZIP is the first of our adaptive traders, able to alter its strategy over time, and also represents the infancy of a long tradition of overlap between automated trading agents and machine learning techniques.

GDX

The Gjerstad-Dickhaut extended (GDX) trader represents a modification within the work of Bredin and Tesauro [51] to the original 1998 Gjerstad-Dickhaut (GD) trader. This preceding trader had, at its core, a mechanism built around a belief function.

The belief function used the history of recent trades to quantify the probability that a trade would occur at any given price. This probability is calculated as a function of the number of orders that had been submitted or rejected from the lob within a history of orders H (as can be seen below). Given these probabilities, a strategy could be found that maximised returns by finding the largest product of the profit and the acceptance probability at each trading price.

$$f_b(p) = \frac{ABL(p) + AL(p)}{ABL(p) + AL(p) + UBG(p)} \tag{2.2}$$

$$f_s(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)} \tag{2.3}$$

The equation used to calculate the belief function, at each price point p, for a buying trading agent can be seen in Equation 2.2, this equation makes use of the following input variables across the last H orders:

- ABL(p) - Number of accepted bids with a price less than or equal to p
- AL(p) - Number of asks on the LOB priced below or at p
- UBG(p) - Number of unaccepted bids priced at p or above

The belief function for the seller's side is similarly structured, shown in Equation 2.3 it utilises the input variables:

- AAG(p) - Number of accepted asks with a price greater than or equal to p

- BG(p) - Number of bids on the LOB priced above or at p
- UAL(p) - Number of unaccepted asks priced at p or below

The modification of GD that is known as GDX is the transition to a dynamic programming framework. This framework adds a layer of complexity to GD by representing the traders' holdings as states within the dynamic programming and re-purposing the GD belief function to provide transition probabilities between these states. This approach leads to a strategy which can evaluate the long term impact of trading, allowing it to maximise profit across a whole session as opposed to each individual trade.

AA

The adaptive aggressive (AA) trader is the result of work undertaken by Vytelingum during his 2006 PhD thesis [54]. It is built upon the broad concept of aggression in trading. In this context, the aggression of a bid or ask refers to the likelihood of it being accepted. Where high aggression is employed a trader may, for example, drastically increase the price of their bids to ensure a trade takes place. This increase provides certainty but directly reduces the profit they stand to make from the trade, thus highlighting the need for the concept of aggression as a constant trade off between the likelihood of trades and their potential profit.

Before AA's trading strategy is fully outlined we must understand the role of the current equilibrium estimate. The true equilibrium of any market over time can only ever be known in experimental situations with rigid price movement rules and known supply and demand curves. Even in these cases, this information would never be present for an individual trader in the market. Thus, an estimate needs to be calculated, in the case of Vytelingum's work this estimate is calculated, at time t, as the weighted sum of the last N=5 transactions in the market. As seen in Equation 2.4, these transaction prices are summed with a series of weights w_i and then normalised to a price value - by dividing by the sum of these weights. These weights aim to highlight the price of the most recent transaction with each previous weight being reduced by a factor of $\lambda = 0.9$.

$$p* = \frac{\sum_{i=t-N}^{t} w_i p_i}{\sum_{i=t-N}^{t} w_i} \quad \text{where} \quad w_{i-1} = \lambda w_i \tag{2.4}$$

The trader is then split up into one of two scenarios: if its limit price is lower than the current equilibrium price when buying (or higher when selling) it is extra-marginal and thus will simply trade at its customer order limit price. This is because it is unlikely to be able to extract profit given the scenario and thus must settle for simply not making a loss. Alternatively if a buyer has a higher limit price than the current equilibrium then it is intra-marginal. This condition is where the aggression parameter comes into use. In intra-marginal scenarios the aggression defines the price point which the AA trader will quote. A less aggressive trader will quote higher prices as it is less intent on making a deal straight away, hoping to wait out current trades for potential future profit. The other side of this is the aggressive AA trader which will sacrifice profit margins in the name of rapidly securing a deal.

The alteration of the trader's aggression exists within two time granularities. In the short term the aggression is directly updated via a Widrow-Hoff learning rule, much like its predecessor ZIP. Over longer periods of time the aggression is adjusted in respect to the volatility of the market. In a volatile market the aggression is increased as the possibility of rapid price changes implies the need for an immediate deal. The assessment of market volatility used by the AA trader is Smith's α, a measure of how prices deviate from the estimated equilibrium, shown below in Equation 2.5.

$$\alpha = \sqrt{\sum_{i}^{t} \frac{p_i - p*}{|t|}} \tag{2.5}$$

Performance

The final three traders presented here: ZIP, GDX and AA, rapidly adapt to market conditions and have been shown to outperform human traders in certain scenarios [51]. The AA trader was shown to be the highest performing of the three, dominating the others in the tests carried out by De Luca [14]. Since then, the dominance of AA has been called into question in a range of scenarios and categorically disproved in the case of a multi-threaded version of BSE [43]. Despite this we can consider AA as one of

the strongest of the three strategies within the confines of our testing. Moreover, we can consider all three strategies' "super-human" performance as a reasonable benchmark with which to compare Deeptrader.

2.4 Deeptrader

The original Deeptrader is a neural network which is trained on the market data provided by trading data obtained from BSE. The network's first layer of neurons is comprised of 10 LSTM units, followed by layers containing 5, 3 and 1 fully connected neurons respectively. The structure of this network is illustrated in Figure 2.4. Deeptrader makes use of the ADAM optimiser [28] to guide the updates to its network weights.

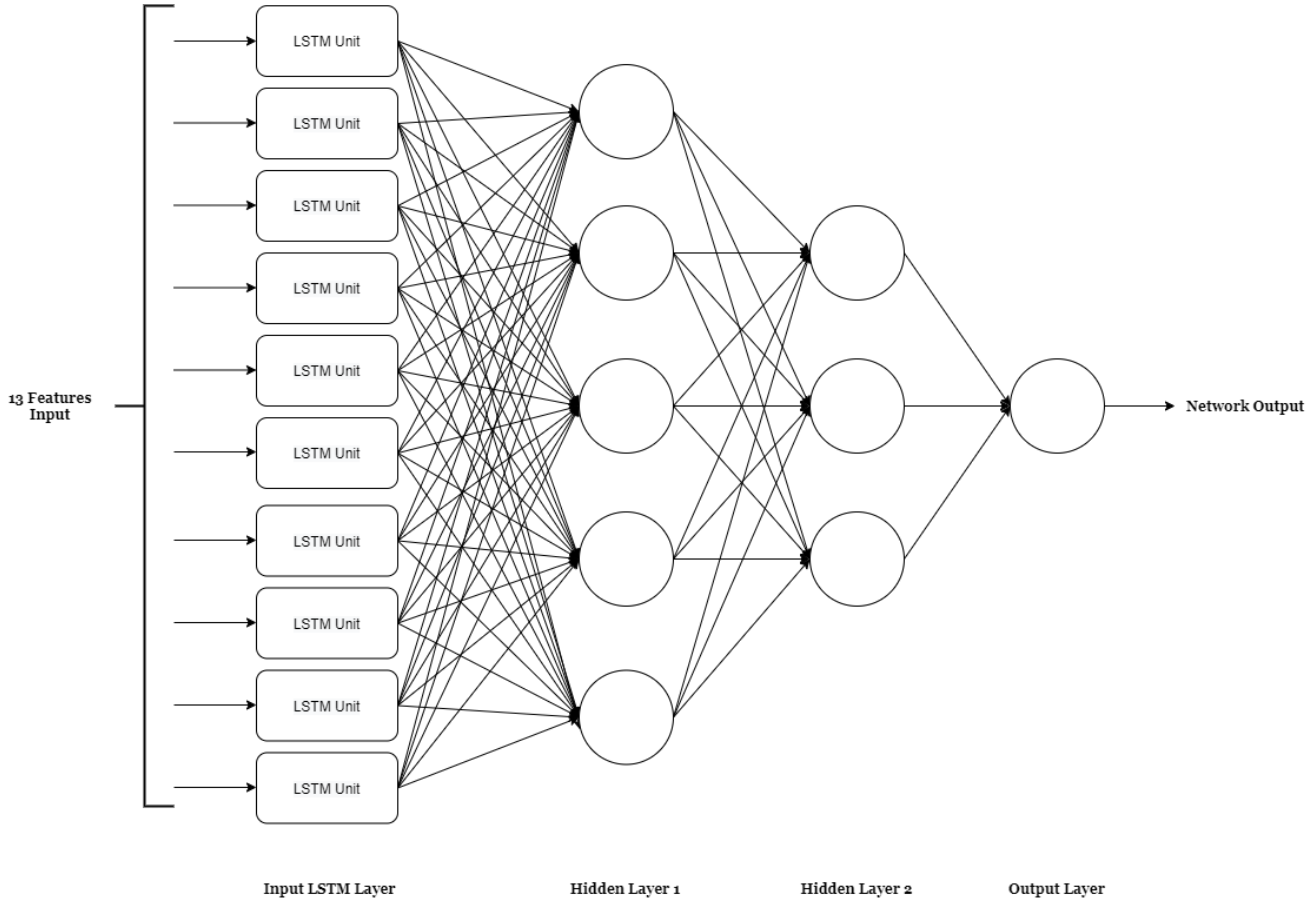

Figure 2.4: An illustration of the Deeptrader network layout

Each of the neurons within the network made use of a Rectifying Linear Unit (ReLU) activation function in the form of Equation 2.6. ReLU can be considered something of a standard within deep learning due to its ability to be rapidly evaluated and it's non-saturating nature. Saturation here refers to an activation function where many values passed through end up close to the boundaries of the function, degrading the relationships between inputs.

$$ReLU = \max(0, input) \tag{2.6}$$

The input to the network was a vast quantity of data collected from BSE, consisting of 13 features which were calculated and stored at every point a trade occurred within the data creation market environments. These 13 features are listed below and where any calculation was required, to extract the features from the information on the limit order book, the formulas are included. The output of the network was a price which the trader should quote to the LOB.

1. **Time** - The time within the BSE simulation at which the trade occurred.
2. **Bid/Ask** - A boolean variable (taking the value of 0 or 1) to represent if the trader that executed the order was a buyer or seller.
3. **Limit Price** - The limit price of the customer order which was provided to the agent which executed the trade.
4. **Spread** - The spread is the difference between the current best ask and best bid on the LOB, taking the form below, where the best bid is signified by p_b and the best ask by p_a (a convention continued throughout the work):
$$Spread = p_b - p_a \tag{2.7}$$
5. **Midprice** - The point equidistant between the Best Bid and Best ask, a rudimentary measure of equilibrium:
$$Midprice = \frac{p_b + p_a}{2} \tag{2.8}$$
6. **Microprice** - A measure of a possible market imbalance that aims to take into account the quantity of units involved in trades, as well as the respective prices. Here q_b being the quantity being demanded and q_a being the quantity offered:
$$Microprice = \frac{(p_b \times q_a) + (p_a \times q_a)}{q_a + q_b} \tag{2.9}$$
7. **Best Bid Price** - The quoted price of the current best bid on the LOB.
8. **Best Ask Price** - The quoted price of the current best bid on the LOB.
9. **Time Delta** - The time within the BSE simulation since the last successful trade occurred.
10. **LOB Imbalance** - A value which weights the quantity of bids and asks on the LOB to detect if there is an "imbalance" in either direction, where an excess of supply or demand would increase quotes on one side and signify a possible future movement of price:
$$Imbalance = \frac{q_b - q_a}{q_b + q_a} \tag{2.10}$$
11. **Quote Volume** - The quote volume is the total number of quotes, both bid and asks, that have so far been made to the LOB within the current session.
12. **Equilibrium Estimate** - The equilibrium estimation utilised by Vytelingum [54], calculated as in Equation 2.4
13. **Smith's** α - A measure of the current volatility, calculated as Equation 2.5 (using the equilibrium estimate defined as feature 12).

Both the data collection and testing of Deeptrader utilised a few common modes of operation within the BSE literature. Thus, each trading session lasted 600 simulated seconds and 100 independent sessions were used to assess performance in a way that provided reliability of results. To provide some realism, the original work utilised BSE's price offset function. This function was a means of dynamically altering the supply and demand, and therefore trading prices, throughout each session.

The base version of BSE however only contains a sinusoidal offset function, the effects of this function on market prices can be seen in the diagram in Figure 2.5. This sinusoidal offset unfortunately provides exactly the same shift in supply and demand across all sessions, providing a predictable environment for trader behaviour and severely limiting the range of trading prices.

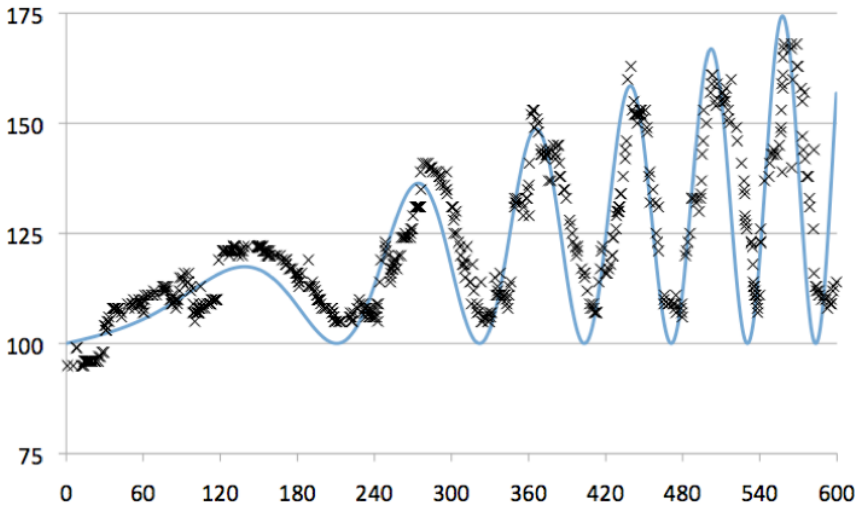

Figure 2.5: An illustration of the sinusoidal price offset function (sourced from [9])

Deeptrader was then trained, using the data collected at the previous stage, for 20 epochs before the state of the network was saved. This final state would then be used on demand within the algorithmic trading element of the network. This consisted of a trading agent which, when tasked with a customer order, would survey the state of the LOB and calculate the required input features. These features would then be normalised and applied to the pre-trained network and the resulting output price would be quoted as an order.

2.4.1 Deeptrader 2

Deeptrader 2 consisted of much the same structure as the original network. Its notable alterations can be summarised as:

- Implementation of a mechanism which linearly reduces the learning rate of the network after each epoch of training, known as learning rate decay.
- A reduced training time of only 10 epochs.
- A filtering of the original dataset to only include the 20 best performing traders.

One of the variables specifically mentioned in Meades' work is the method of customer order replenishment. BSE has a range of mathematical functions it can use to decide how often these fictional "customer" orders are made to the individual traders. The most realistic of these is the method which utilises the Poisson distribution. This ensures random intervals between each customer order, a method prevalently used within the mathematical market modelling present in [5]. Meades' notes that the method of customer order replenishment wasn't specified in the original Deeptrader work and thus we can only speculate as to Wray's choice. As a result of this ambiguity, and owing to its improved realism - the following work makes use of the Poisson order replenishment for all data collection and testing.

2.4.2 Testing

The previous works in this domain make use of two common methods of evaluating an algorithmic trader's performance. These methods are taken from the human-trader comparison work of Tesauro and Das [12]. The exhaustive comparison of all combinations of traders is generally unfeasible and these two methods aim to simulate two common scenarios in trading environments but also to act as a baseline across which to compare works. They are defined as such:

Balanced-Group Test - This test is composed of two separate trading strategies. Each strategy represents half of the overall market. In the case of my BSE experiments specifically this means that strategy 1 will be instantiated as 20 buying agents and 20 selling agents (40 agents total). The other half of the market will be composed of 20 buying and 20 selling agents of strategy 2. This allows a direct comparison between the strategies' performance against each other and their interactions within the market.

One-In-Many Test - This test aims to quantify the effect of a single trading agent, of a new strategy, on a homogeneous population of another trading strategy. Thus, the "dominant" strategy in the market, in this case, consists of 39 buyers and 39 sellers. This is compared to the invasive strategy consisting of only a single buyer and a single seller. This aims to replicate a scenario where the strategy being tested is introduced to a wider market.

These tests are then evaluated using a metric called "Average Profit Per Trader". This takes the final profit for all traders at the end of a session into account and then takes a mean of that for each strategy. Given the prevalence of these methods of evaluation in algorithmic trading work, they will serve as the starting point for the analysis of this project in an effort to provide meaningful comparison to previous and future work. The results of these experiments for Wray can be summarised in Figures 2.6 and 2.7 below to provide a point of comparison for later findings.

	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

Figure 2.6: Balanced-group results, taken from [37]

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

Figure 2.7: One-in-many results, taken from [37]

Chapter 3

Project Execution

3.1 Deeptrader Implementation

The initial replication of the Deeptrader network required the creation of a modified version of BSE which could be used to effectively collect the data required to train the network. This was achieved in the form of a copy of BSE which was altered, in its order processing function, to calculate and record the 13 required input features at the point of every successful trade. These modifications to what can be considered the "back-end" functionality of BSE highlighted the need for a separate testing platform which had not undergone structural alterations. From that point onward the above was designated the "Data Collection" BSE file and another separate "Test" BSE file was kept unaltered to maintain the integrity of testing.

The set up for data collection was well documented in both previous works and held a cohort of 40 buyers and 40 sellers. In each iteration these market participants consisted of 4 separate trading algorithms. Thus, each iteration existed as a different unique subset of all 7 possible trading agents, totalling 35 different permutations. Within each stage there exists a mechanism to alter the proportions of these 4 trading agents, in multiples of 5. This alteration was equal on the buyer and seller side and itself led to 35 permutations of proportions.

Each of the given (35×35=) 1225 market scenarios were run 36 times to increase the data gathered and account for any variability in trading behaviour when training the network. This repetition created a dataset containing 44,100 unique market sessions and around 18,428,463 individual trades.

Once the modifications were made to ensure the correct permutation of traders and repetition of runs, the data collection file was transferred to BlueCrystal4 (BC4): the University of Bristol's high performance computing machine. From there a series of scripts were written which automated the process of running the program in the correct configurations and collating the data, a task which required up to 32 computational nodes and averaging around 8 hours of run time. This resulted in the production of datasets commonly exceeding 2GB in size.

Upon the completion of this data collection in the original work, there existed a step where a relatively simple method of normalisation known as Min-Max Normalisation was employed on the data. This method (seen below in Equation 3.1) helps remove the effect of the data size and range in the network training, scaling the data only between 0 and 1 based on the minimum and maximum example of that feature in the network. Given the size of this dataset and the limitations of local computing, it could seem intuitive to perform this normalisation on the BC4 platform however due to popularity and resource limitations embedded within BC4 it seemed prudent to only perform completely necessary tasks within its confines. This work therefore utilised the Python Dask library [13], a specialised framework for the efficient execution of calculations and data manipulation on large deep learning data sets. This framework was used to normalise and combine the collection of csv files created by the data collection files.

$$x_{normalised} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

The neural network structure was implemented using the deep learning library PyTorch [41], in conjunction with the original design plans presented in Wray's work. The network consisted of the requisite 10 LSTM units followed by three fully connected layers containing: 5, 3 and 1 neurons respectively. Each of the neurons used the ReLU activation function.

A slight caveat in comparison to the original work of Deeptrader is that most recent version of BSE (at the time of writing BSE v1.4) executes within the Python 3 environment. Therefore the reference implementations of the GDX and AA traders, graciously provided within the complementary files thanks to the work of Ash Booth and Dan Snashall, were outdated to the previous version of BSE. This required the work to update these reference implementations by altering portions of their syntax and reconsidering their internal logic: due to the fact they utilised a number of features which had been deprecated in Python 3 [53].

3.1.1 Deeptrader 2

When considering the implementation of Deeptrader 2, the goal was to create an environment that provided uniform tests against the original Deeptrader. In this regard, when creating the Deeptrader 2 dataset I decided to use as my starting point the dataset I had created above for testing the original Deeptrader. Thus, in contrast to Meades' work, the two datasets would contain exactly the same set of trades occurring. The only difference would be the stripping back to the 6 features decided upon by Meades.

This methodology obviously ignores the filtering of trader data which occurred in Meades work, this is a conscious decision aimed at the trade-off we must consider when we remove training data. This reduction in the size of the training set by 75% may improve performance, although this improvement is negligible in Meade's analysis, but it must be considered that the loss of so much data can seriously hamper our efforts to generalise Deeptrader to a wider range of scenarios.

The linear learning rate decay presented in the work is not a native functionality in PyTorch due to its inflexibility in training. This inflexibility refers to the fact that altering the learning rate by a fixed amount each epoch must be hand-tuned and doesn't react to changes in the network. Nevertheless, I manually implemented linear learning rate decay through a learning rate schedule to match DT2. This brought up initial questions about the learning capability of the DT2 network: given that the ADAM optimiser, currently used for training, made minor learning rate changes in the course of its operation. The worry being that these larger linear learning rate changes were obscuring the minutia of ADAM's functionality.

3.1.2 Training

The training of both networks took place on the GPU nodes of BC4. A training program was composed that could be trivially altered to train both Deeptrader and DT2. This training program followed deep learning best practices by randomly splitting the dataset into a train and test set (comprising 90% and 10% of the data respectively). Both networks were then trained across 20 epochs, the maximum training time presented in Wray's work. The network was checkpointed and cross-validated every 5 epochs such that the networks could be compared at various points (including the recommended 10 epoch training range of DT2):

- **Checkpointing** - Saving of all network parameters such that the network can be instantiated in that exact state at a later point from the save data.
- **Cross-Validation** - The application of the current state of the network to the test data set. The loss across the test set can then inform the progress of training. This step also highlights potential overfitting, if the loss during training continues to reduce but the test set loss stagnates or increases: there is a potential that the network may be fitting too closely to the training set and thus not generalising to this test set.

This was combined with a continuous logging of data to provide a step-by-step understanding of the training of the network. This log can be interpreted through tools such as Tensorboard [2], the tensor-

board graphs that represent the training of Deeptrader and DT2 can be seen in Figures 3.1 and 3.2.

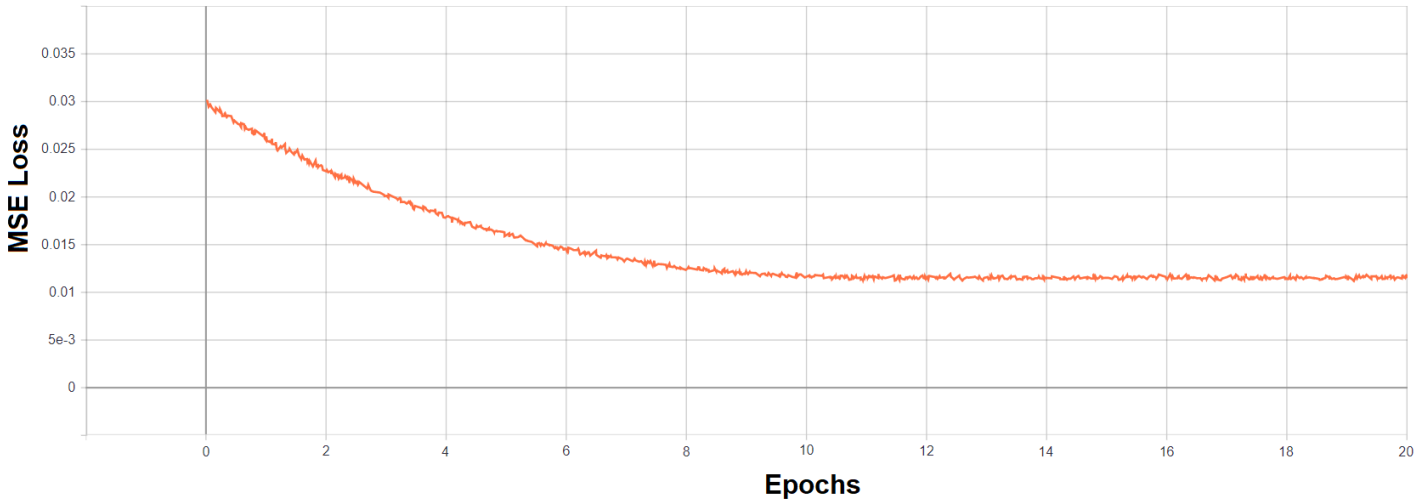

Figure 3.1: The training loss for the original Deeptrader network

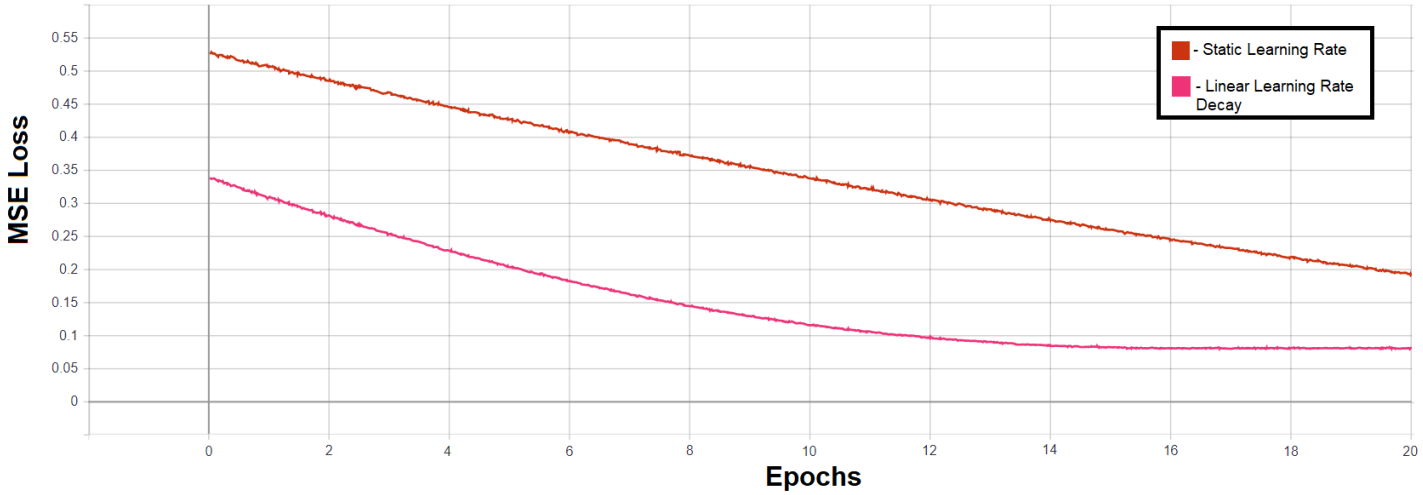

Figure 3.2: The training loss for two versions of Deeptrader 2, both with and without learning rate decay

3.1.3 Performance

In Figure 3.1 we can view the training of the original Deeptrader network. After only a single epoch of training has occurred, we can see that the network starts with a very low initial average training loss of 0.03. This, when denormalised, indicates an average price difference of approximately 43 between the predicted value and the labelled value in the training data. Whilst this difference is enough to render the untrained state of the Deeptrader network unprofitable, it does highlight that the underlying structure of the market is simple enough that the bare minimum of network training brings Deeptrader into the ballpark territory of successful price estimation.

This loss reduces by half within 6 epochs and begins stagnating with minor negative changes after that. When compared to Table 3.1 we see this stagnation reflected in results, with the cross-validation loss dropping by a whole order of magnitude within the first five epochs. After the network reaches ten epochs it reaches a cross-validation loss value where it essentially settles, with a further insignificant change of only 8.4 x 10^{-10} during the rest of training.

This reinforces the assertion made by Meades that 10 epochs was an optimal point to cease training within the network, as we can see that further training has little effect on loss. This is not an immediately surprising result as many deep learning networks converge in rapid time in simple situations. It does however speak to exactly the fact that the training situation may be limited in its variance and complexity, surely an indicator that the network could struggle to perform in more general scenarios.

Looking forward then to the training of Deeptrader 2, presented in Figure 3.2, the initial reservations about the learning rate decay were examined by comparing two mirror implementations of DT2. One with the learning rate decay implemented and one without it.

Learning Rate Decay The network with learning rate decay seems to have a lower loss after first training and a much more rapid descent in training. The decay creates an environment where the increasingly minor changes in weights allow for rapid traversal of the loss environment. This gradient soon levels off and itself begins to stagnate at around 14 epochs. This is reflected in the cross-validation in Table 3.1, where there is a minor improvement until 15 epochs where the network then remains stable.

Static Learning Rate In comparison the network without learning rate decay has a much higher initial loss and a much slower traversal downwards of the loss space. Notably however the reduction in training loss seems consistent across training. This edition of DT2 has a significantly higher cross-validation loss across the board during training, indicating a worse ability to approximate market structure than its counterpart. It is however worth noting that the cross-validation loss continues to decrease regularly throughout its training process, perhaps indicating that further training beyond the confines of this test could yield improved results.

	Average Testing Loss		
Epoch	**Deeptrader**	**Deeptrader 2 (without LR decay)**	**Deeptrader 2 (with LR Decay)**
0	1.5842×10^{-6}	3.0941×10^{-5}	1.8827×10^{-5}
5	8.6918×10^{-7}	2.4703×10^{-5}	1.0836×10^{-6}
10	7.0629×10^{-7}	1.9215×10^{-5}	6.1863×10^{-6}
15	7.0547×10^{-7}	1.4455×10^{-5}	4.9619×10^{-6}
20	7.0545×10^{-7}	1.0419×10^{-5}	4.9619×10^{-6}

Table 3.1: The average loss acquired when both networks were validated against the testing set. Note the suggested optimal training time for Deeptrader is 20 epochs whereas for Deeptrader 2, training for 10 epochs is suggested optimal (these optimal training points are highlighted in green).

The much higher overall cross-validation loss present in the DT2 networks, when compared to the initial Deeptrader, indicate the effect the feature removal has had on performance. Whilst the benefit of a more lightweight network (utilising only 6 features as opposed to 13) cannot be understated, when considering the training time and dataset sizes, it is clear to see that the removal of ∼50% of training information has a negative effect on the network's capabilities.

The previous observations aren't immediately damning in regard to these networks' utility however, the testing results initially seemed to replicate a key statement made in Meades' dissertation "it became apparent that loss was not a good predictor of trader performance" [37]. To explore the validity of this, the Deeptrader validation loss figures present in Table 3.1 were used alongside the APPT results from a benchmark comparison (a balanced group test against an AA trader) to calculate a Pearson R correlation coefficient. The result of this calculation was a score of R=-0.9936 indicating a strong negative correlation. This contradicts Meades' suggestion by highlighting that as network validation loss decreases, the Deeptrader performance increases with a strong likelihood.

When this is considered, it becomes very highly likely that the reason this trend wasn't initially identified was an unchanging performance between the later stages of training. When the performance (against the benchmark AA) of the final 3 checkpoints is examined, we see:

- 10 epochs - Mean APPT = 244.7
- 15 epochs - Mean APPT = 254.8

- 20 epochs - Mean APPT = 254.8

From this it can easily be understood how a trend across performance data could be missed when interpreting this data. When compared to the testing loss values in Table 3.1, the true cause of this halt in performance becomes clear. Between 10,15 and 20 epochs there is an extremely small alteration in average testing loss. This indicates that the network is not further improving despite extra training, an extremely common phenomenon that occurs as a result of overfitting.

This can likely be explained by the price offset function being used in both works. This sinusoidal increase in price throughout the session is slightly obscured by trader behaviour, providing little variations about the function, but essentially leads to the networks trivially learning the underlying sinusoid of price movement. This price movement is clearly learnt in the first few epochs of training and the remaining epochs contribute little to further improvement whilst increasing the chances of overfitting.

3.2 GradientSHAP/kernelSHAP

Before fully exploring the changes that could be made which would allow the previous networks to generalise to a wider range of scenarios, I chose to explore the impact of the features on the practicality of the networks. To some extent, this analysis was present in Meades' work: he produced an ablation study that measured network performance after repeatedly removing a feature from the network. This ablation study was a good point to start from, however lacked the nuance to understand the interplay between features. This is because the removal of one feature at a time couldn't capture the relationship between multiple features in the performance.

For this, I turned to Captum - a Python Library that provides a range of machine learning interpretation features. The approach that served to head up our analysis was Shapley Additive Explanations (SHAP) [36]. SHAP acts as a high-level framework that unifies a number of distinct machine learning interpretation methods such as LIME [42] and DeepLIFT [47] with methods based on the game-theoretic concept of Shapley values [46]. It stems from the observation that a number of these methods have similar structures but unique benefits.

A fully trained network can be used as input to the methods present within Captum such that analyses can be undertaken. The result of this SHAP analysis is a series of Shapley Values. The Shapley values, here, numerically indicate feature importance within a machine learning framework. They signify a measure of the distance between the predicted output of the network and the output if we didn't know each of the features.

In this case it refers to the difference in price output caused by having each of the features supplied to the network, compared to the "baseline" price. This baseline price was the price output when the network was applied to the mean of the validation set (∼$122.1632). Therefore, the Shapley value refers to the quantified increase or decrease in price from the baseline (caused by the according feature).

We applied gradientSHAP (Captum) to the full validation set of our network (1,896,262 trades) to get a distribution of Shapley values across the data. This distribution shows the magnitude of effects the respective features had on the final price across a number of different market scenarios.

To best display these distributions, this work utilised violin plots. Violin plots aim to show the range of distributions but also indicate, via the width of each plot, the density of these distributions. This density highlights where the majority of values lie when regarding the total population of results. The violin plot for the original Deeptrader network is shown in Figure 3.3.

This highlights a few key points. Firstly it supports the findings from Meades' work that the features:

- Type (Bid/Ask)
- Midprice
- Microprice
- Best Bid

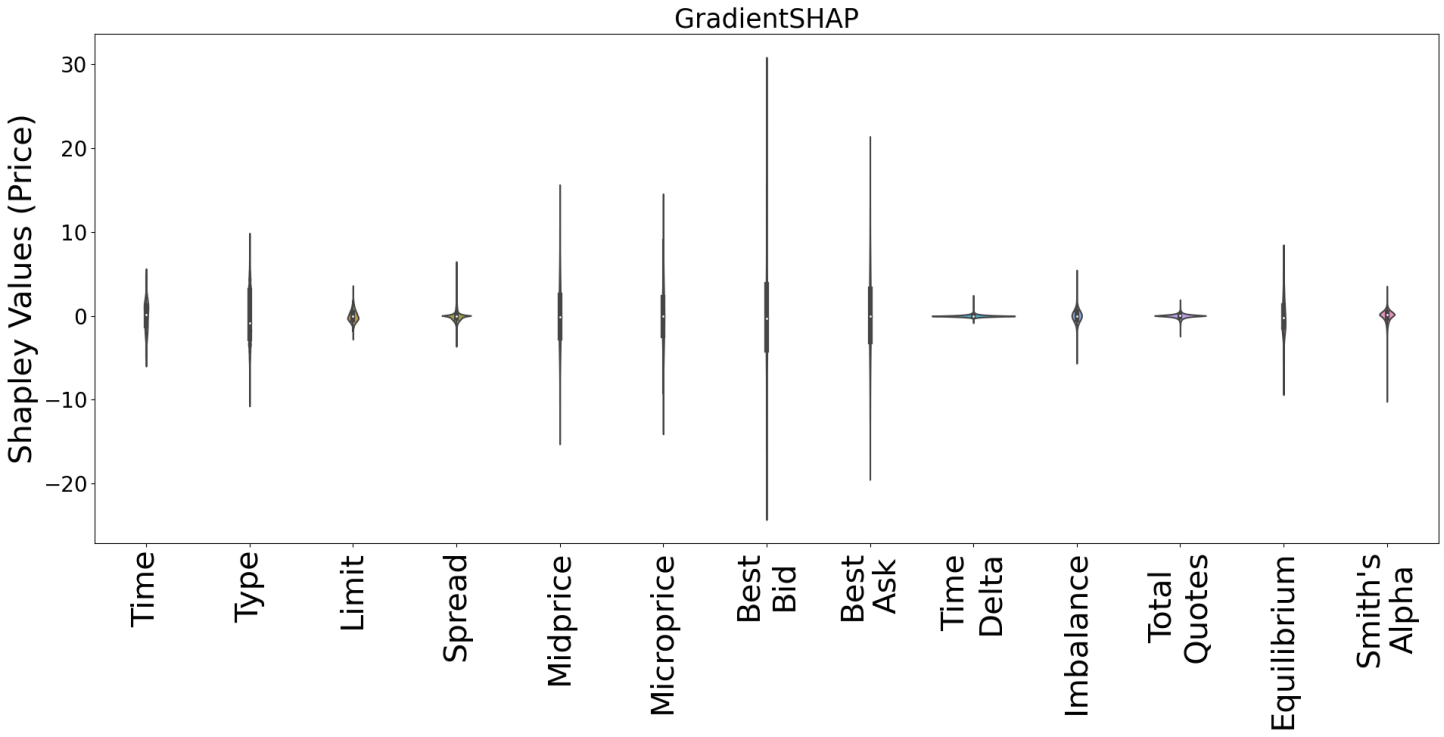

Figure 3.3: A violin plot of the distribution of Shapley values for the original Deeptrader.

play a significant role in the price discovery of the Deeptrader network. In addition it highlights the importance placed by the network on the Best Bid feature, with its value altering the final network output by up to 30.

Contrary to Meades' ablation study however are the results regarding the Best Ask and the Equilibrium estimation. These features can clearly be seen to have a relevant impact on network output. These relationships were likely not captured by the original ablation study as, when the features were individually removed, the network utilised other features more intensely to compensate. For example in the case of the Best Ask feature being removed, the network will have most likely have (in simple terms that do not directly apply to the function of deep learning) "extrapolated" this information from a combination of the Midprice and the Best Bid.

Another interesting point of divergence is the limited importance placed by the network on the features Time, Time Delta and Total Quotes. These features contributed prices changes up to +/- 5, however their density indicates that in the vast majority of cases they had no overall effect on the network output. These features share a common theme in that they serve to abstractly define the relationship between the previous and current states of the LOB across trade data. This implies that the network may struggle to capture the effect of temporal trends across trading data and instead simply uses its current snapshot of the LOB to decide an adequate price.

To extend this analysis we can employ the kernelShap method which allows us to take an individual sample from the data and investigate its Shapley values specifically (as opposed to the global estimation method of gradientSHAP).

To do this it applies the "Local Interpretable Model-agnostic Explanations" (LIME) framework. This works by taking the sample inputs, applying them to the model and observing the output. It then attempts to train a linear regression model that replicates this result. This linear model can be thought of as fitting a line-of-best-fit to the collection of input features. The coefficients of this model then represent how much each of these features are weighted in order to reach the output; kernelShap then converts these weights into Shapley values for ease of understanding.

This allows us to give a good explanation of any particular instance we would like to explore, however the feature importance given only applies to that specific instance. An example is given in tabular form below for illustration.

	Feature Input Value	**Shapley Value (Price)**
Time	257.8	-0.7656
Type	0	+2.2098
Limit	78.00	-0.5394
Spread	35.53	+1.4442
Midprice	95.76	-4.2804
Microprice	95.76	-3.6018
Best Bid	113.5	-3.1668
Best Ask	77.99	-7.2906
Time Delta	0.6377	-0.0696
Imbalance	-0.1304	+0.5742
Total Quotes	23.00	-0.3654
Equilibrium	118.2	-0.1914
Smith's α	0.0932	-0.6612
Output Price	105.8	

Table 3.2: The kernelShap values from a randomly sampled instance of Deeptrader. The set of Shapley values here sum to -16.7040

We can see the impact of each feature when calculating this single output price but also can consider that the sum of the Shapley values overall approximates the difference between the output and baseline (outputPrice – baselinePrice), i.e. \$105.85 – \$122.16 = –\$16.31.

Given the results of the SHAP work, we can clearly see that there are relationships not captured by the original ablation study. Whilst some elements of the analysis reinforce conclusions made by Meades, it provides enough counter evidence to rule out the removal of the 7 features suggested by DT2. This does not mean all the features must always remain, as we can continue our attempts to generalise the Deeptrader network with all 13 features and use the tools employed to periodically assess if some features in particular play more or less of a role than found here.

This analysis crucially helps us understand how Deeptrader parses the structure of the market environment through the LOB. Going further, when reintroduced back into the learning pipeline, it allows each trade the network makes to provide Shapley Values and explain to some degree the decision process behind its trading price. This brings back a level of understanding and trust that had been sacrificed for the trading performance provided by the nature of the black-box, deep learning methods employed.

3.3 Generalised Brownian Motion

Generalised Brownian motion with drift (GBM) is a mathematical model that has commonly been used to model stock price movement [16]. It consists of a function which randomly increases or decreases at each timestep, combined with another such function that makes minor changes in a certain direction over time. This leads to a wide variation in prices and somewhat emulates the erratic movements of stock prices in real markets. Across sessions, price movements can follow vastly different trajectories as can be seen in Figure 3.4. The aim in introducing this concept is an attempt to bridge the gap between the "walled garden", that BSE traders are studied in, and a wider literature around stock price prediction/algorithmic trading.

To assess the effect of GBM within BSE I created an alternative price offset function in both the data collection and testing versions of BSE. This was done using a public domain GBM algorithm [45] whose parameters where tweaked for application within the framework. From there a dataset was created containing all 13 features and exactly the same permutations of traders as the original Deeptrader data

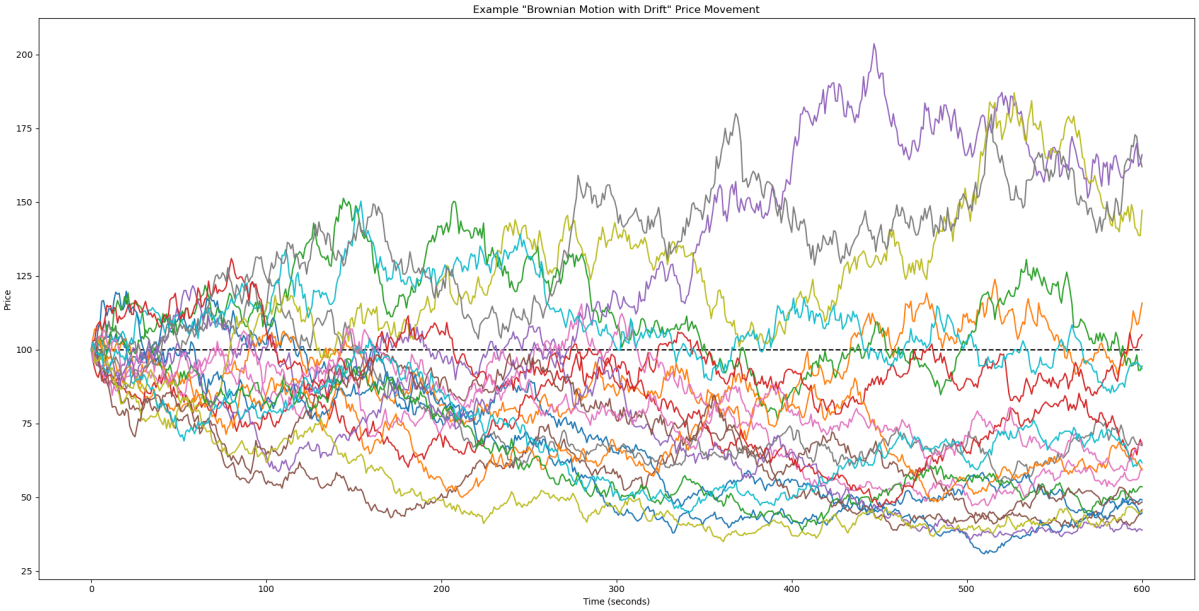

Figure 3.4: 20 Separate runs of the GBM price offset function to illustrate the price volatility.

collection. This was augmented with the price offset function which was generated uniquely for each of the 35 repetitions that each permutation underwent, providing a wide range of price movement information for each trading setup.

An intermediary step before training a new version of Deeptrader was a test for backwards compatibility. The previous edition of Deeptrader was examined in the testing environment both with and without the new Brownian price offset function. The balanced-group tests were carried out against the three adaptive traders within BSE (AA, GDX, ZIP) such that a comparison could be drawn to how an "intelligent" market participant would react to the alterations in price. The aim was to assess if the original work generalised to this new scenario without any further training.

It can be seen from Figures 3.5, 3.6 and 3.7 that this was not the case. On the left we can see the balanced-group performance within the original BSE market setup. As in Wray's work, Deeptrader outperforms GDX and has comparable performance with ZIP, being slightly outperformed by AA. On the right, there is stark difference, with Deeptrader's Average Profit Per Trader rarely exceeding 100, there are also significant instances where the network returns zero profit. In comparison the adaptive traders not only provide a steady profit but actually out-perform their counterparts in the traditional BSE setup.

This makes sense as we can consider that the range of values that Deeptrader is accustomed to dealing with is far smaller than the possibly range of inputs produced by a market modelled with GBM. The crux of this is a network that is unfamiliar with the price movement and trading behaviour that is occurring because it hasn't been exposed to it previously. This instance indicates that the variation present in the training data is key to the ongoing success of the network.

This contrast with the continued performance of the adaptive algorithmic traders and provides a key insight into their design. The ability to alter their strategies throughout trading in a changing market allows them to profitably traverse general scenarios.

3.3.1 Re-Training

The Deeptrader network was then trained using the GBM dataset. Immediately we can see from Figure 3.8 that the initial loss in training is much higher than the previous network, with the value of 0.14 being nearly five times larger than the base dataset. The loss curve of this training stage shows a sharp drop over the course of 10 epochs, with a continued steady reduction down to 20 epochs. This seems par for

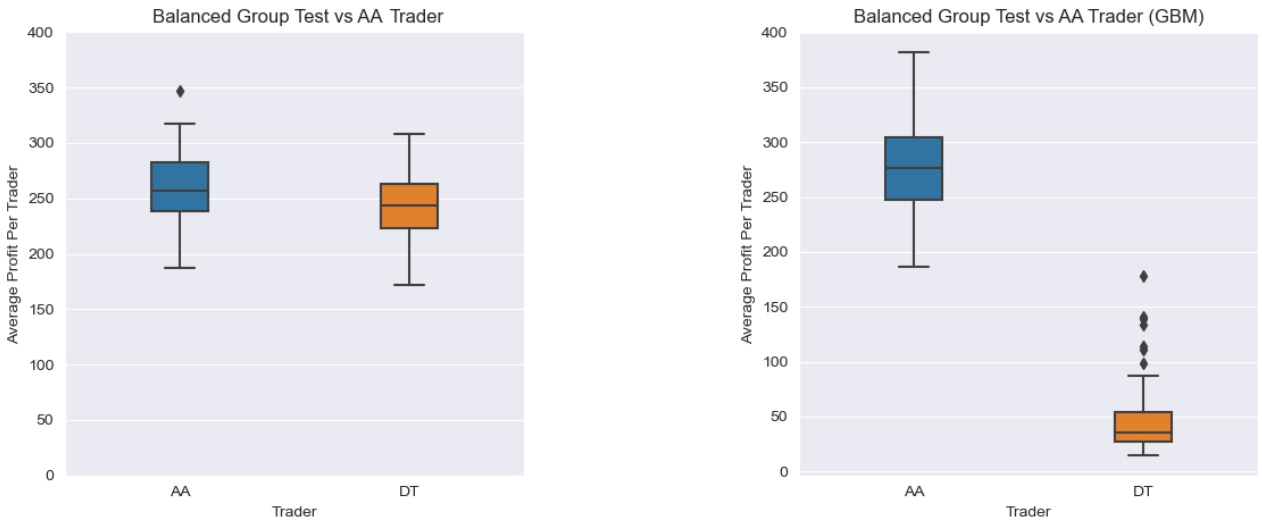

Figure 3.5: Old DT AA

Figure 3.6: Old DT GDX

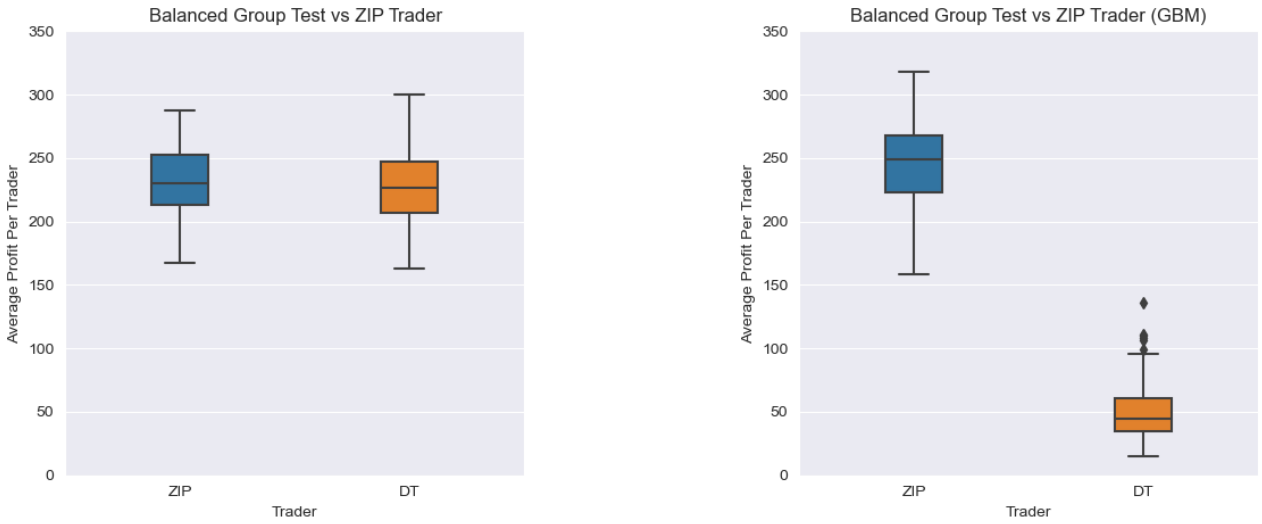

Figure 3.7: Old DT ZIP

the course with the original Deeptrader however when we inspect Table 3.3, the difference becomes clear.

During those first 20 epochs the cross-validation loss steadily decreases, not exhibiting the same behaviour as the two networks that hit a lower bound of loss during that period. This invites the possibility that extended training may benefit this updated Deeptrader before overfitting occurs. As such the network was trained for an extended 60 epochs. Immediately it could be considered when studying Figure 3.8 that after 30 epochs, the reduction in the loss is small enough to not warrant further training. Despite this when taking into consideration the cross-validation results, a trend of steadily improving performance presents itself across the entire training period.

This indicates that the added variability of the new data proves a more difficult function to learn and thus we can consider a network that is now actively learning a complex scenario and not immediately overfitting. This warrants a re-examination of the effectiveness of deep learning training methods, such as Dropout or hyperparameter tuning, given that their lack of impact on performance in Meades' work now seems trivially true.

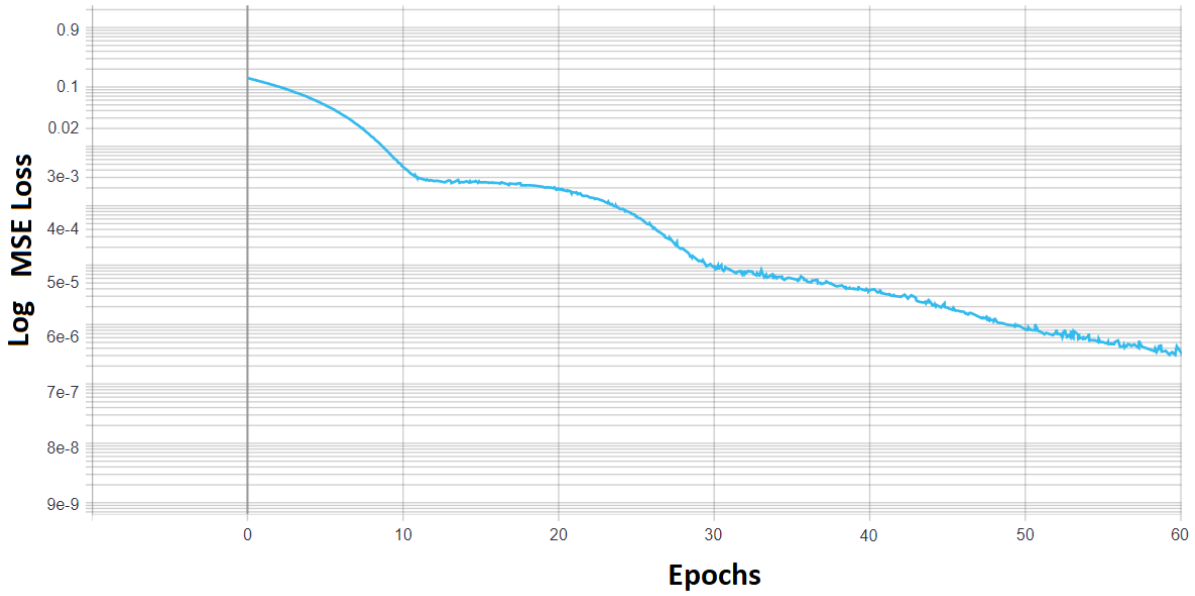

Figure 3.8: The training loss for the Deeptrader network trained on the Generalised Brownian Motion Dataset, shown here with log scaling

Epoch	Average Cross-Validation Loss
0	7.3993×10^{-6}
5	2.1192×10^{-6}
10	1.7721×10^{-7}
15	1.4870×10^{-7}
20	9.9176×10^{-8}
25	2.3927×10^{-8}
30	4.9262×10^{-9}
35	3.2187×10^{-9}
40	1.9148×10^{-9}
45	8.9823×10^{-10}
50	4.4224×10^{-10}
55	2.7420×10^{-10}
60	1.6730×10^{-10}

Table 3.3: Cross-validation loss for GBM Deeptrader

3.4 ISHV/PRZI

The PRZI trader presents an interesting concept in the form of a trader that can alter its strategy based on an input parameter. The issue with its current implementation however is the random designation of this parameter at the beginning of trading. This leads to a trader that performs admirably but doesn't utilise its ability to adapt. To change this, one can consider any number of measures of the state of the market upon which to condition the strategy variable. One such measure is the LOB imbalance statistic that plays a crucial role in the trader ISHV[6]. This statistic, shown in Equation 3.2 quite primitively measures imbalance in the market but here allows us to see how a PRZI trader that can continually adapt may perform. This measure is calculated every time another trader makes an alteration to the LOB and thus PRZI now has a continually updating strategy.

$$\Delta m = P_\mu - P_m \quad where \quad P_\mu = microprice,\ P_m = midprice \tag{3.2}$$

Δm was initially purposed to alter prices linearly via addition, thus the size of values it produces within BSE aren't suitable to be used as input to the [-1,0,1] range which PRZI takes. To remediate this, the result of this calculation was passed through a sigmoid function calculated as in Equation 3.3. Note the $\pm$ present as this mirrors the strategy calculation for buying and selling sides of the market. The output of this sigmoid function, when the PRZI trader is buying, in an example range of Δm values can be seen in Figure 3.9. This figure would be vertically mirrored on the sell side of the market.

This highlights how the Sigmoid converts these values depending on their size to a point within the input range of the PRZI trader. This function included a hand-tuned "Strength" parameter that determines how changes in Δm begin to converge to the -1 and +1 sides of the output. In this work that parameter took the value of Strength=0.25.

$$Strategy = \frac{2}{1 + \exp^{\pm Strength\ *\ \Delta m}} - 1 \tag{3.3}$$

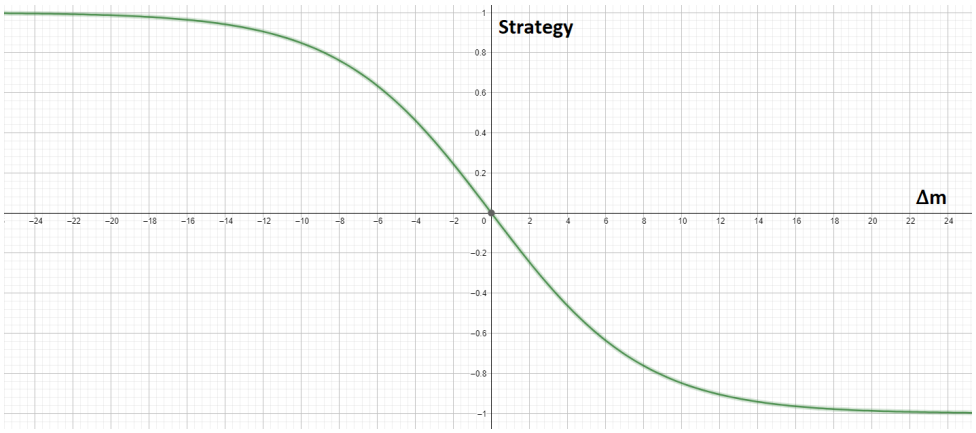

Figure 3.9: sigmoid

This new ISHV+PRZI trader, which can be called IPRZI for convenience, allows us to scrutinise two key aspects that speak to the generality of Deeptrader. The first being a series of experiments that serve to inform us in how Deeptrader reacts to a trading agent that wasn't present in its trading data? Can the wealth of experience generated by our training data allow the network to operate profitably against this "unseen" trader.

To test this, the IPRZI trader was pitted against a Deeptrader network which had not been trained on data that included the IPRZI trader. This occurred in the form of two balanced-group tests between the traders. The first of which was a standard GBM environment that would indicate general performance. The second was a market in which a trading "shock" occurred at random points during trading. This shock would imbalance the LOB by massively increasing the supply available in the market. It would be followed by a counter-shock at a later random point that would significantly increase the demand in the

market, imbalancing the LOB in the other direction. These moments of extreme imbalance provided an opportunity for the IPRZI trader to pre-emptively react, allowing it to extract profit by anticipating the future movement of prices. This second experiment was designed to be conducted in IPRZI's favour to measure the extent to which Deeptrader could compete.

To complete this experiment a "Shock" trader had to be created which made, at a random time, an ask of quantity 50 at the price of the current Best Ask. The trader would then wait another random amount of time and produce a bid of quantity 50 at the price of the current Best Bid. This shock trader was designed simply as a mechanism to alter market dynamics and thus didn't take part in any other trading activity or attempt to make profitable trades.

The first of these experiments, without the market shock, can be seen on the left of Figure 3.10. From this Figure we can see a Deeptrader agent that confidently outperforms the new IPRZI agent. We can view from the right of the Figure that this market shock improves the lower bound on IPRZI's performance, giving it a more consistent set of APPT returns. More surprisingly, we see a Deeptrader agent that continues to outperform the IPRZI trader. This result, from a market environment stacked against it, suggests that not only can Deeptrader reasonably generalise to new unseen trading situations. It can further generalise to exceed the profitability of unseen traders. This also suggests that whilst the imbalance parameterised methodology of the IPRZI trader is based on strong principles from the literature, it offers no extra utility within this specific environment.

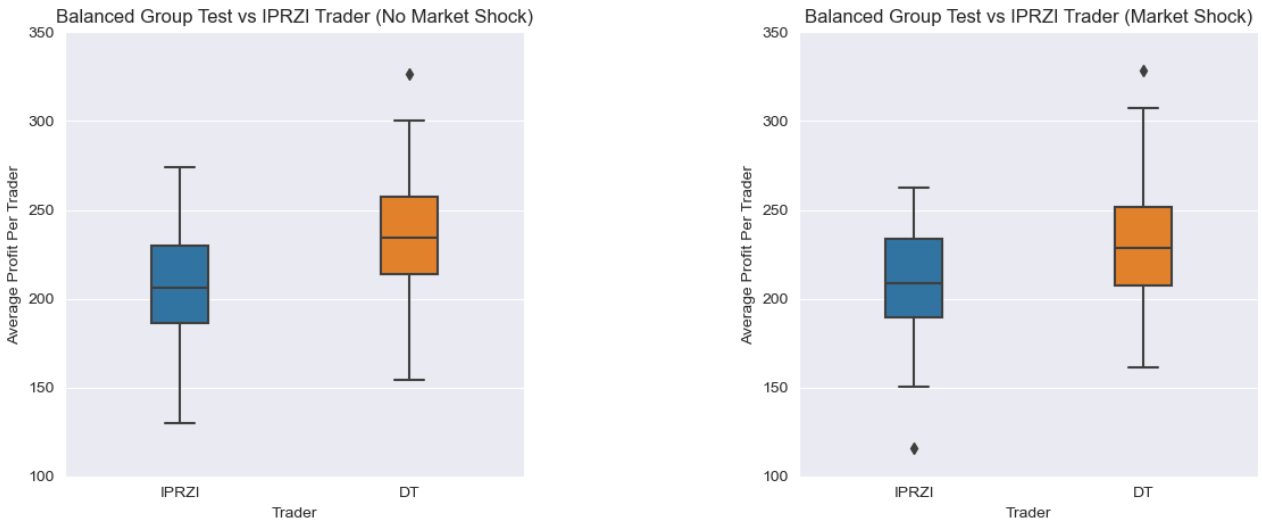

Figure 3.10: Balanced-group tests against the IPRZI agent

The outcomes regarding Deeptrader's ability to generalise to new traders here must of course be contextualised, whilst the IPRZI agent is indeed a completely new strategy. We must consider that the unique aspects of the strategy sit atop a unification of three traditional trading agents (ZIC, GVWY, SHVR). These three agents all exist within the training data used as input to the network. Therefore, it can be considered that Deeptrader's performance against the IPRZI trader may rely on some artifact of the learning conducted during initial training on the traders' constituent strategies. To strengthen this result, it is clear that a truly unique strategy independent from the current BSE traders must be implemented and tested.

The second point, in a way attempting to address the behaviourist mentality displayed in Wray, Meades and Cliff [56], is if the network can learn the specific behaviour of this new trader. Once it has learnt from a wealth of data provided by IPRZI, putting profitability to one side for a second, will the network be able to act and react to imbalance of the LOB in the same way that IPRZI does? This examination bears relevance for realism of the scenario played out in the stated work which supposes an eventual application to human traders.

To conduct this examination, a Deeptrader network was trained as we have established however the trader SNPR was removed from the dataset and replaced with IPRZI. Thus, the training data now contained up to one seventh of total data containing trades from the IPRZI trading agent. The limitations of the current iteration of the BSE data collection system resulted in an inability to include SHCK traders in the market environments in which IPRZI data was created. Thus the IPRZI trader could only react to the minor imbalances that occur naturally in the LOB, as opposed to the excessive imbalances caused by a SHCK trader. Thus it can be expected that the whilst the IPRZI will still exhibit the behaviour of reaction to LOB imbalance, this behaviour will be far less pronounced than had a SHCK trader been present.

A Deeptrader network that had been trained on this new dataset was then placed in a market with a SHCK trader aiming to imbalance the market.

The order prices of a singular Deeptrader agent were then examined across a number of trials for a time period of 100 seconds after the market shock. The SHCK agent in this case was imbalancing the market with an offer to sell 50 units of the asset for the price of the current Best Ask. Traditional market wisdom would imply that this action would, as a result of massively increasing supply, drive down the prices in the market. A Deeptrader agent which had adopted IPRZI behaviour would react to this imbalance by lowering its prices accordingly.

The monitoring of "post-shock" prices was supplemented by performing a linear regression on the data from across trials, hoping to indicate a trend. The results of this linear regression are shown in Figure 3.11. These results show a great variability in post-shock prices over the different sessions, despite using the same price offset function for each session (a GBM price trajectory was generated and then set to be re-used in further sessions). More importantly, the linear regression shows no clear reactive drop in prices after the market shock. This indicates that the Deeptrader agent hasn't developed the behaviour present in the IPRZI trader through training.

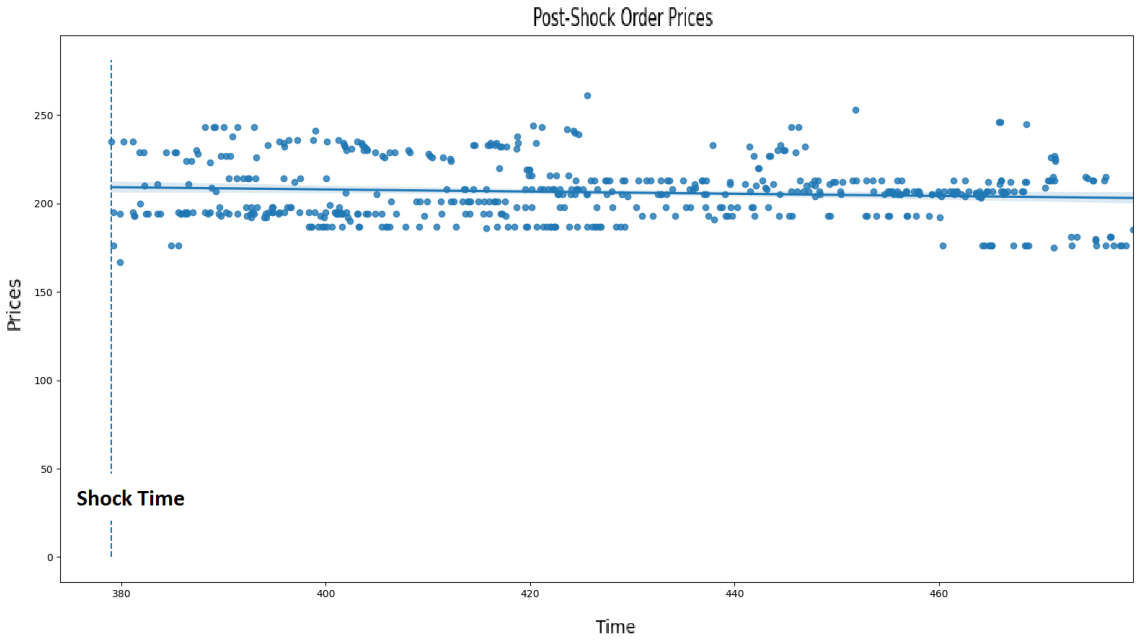

Figure 3.11: Linear Regression on Post-Shock Prices

These results highlight less so the tendencies of the network, but more the shortfalls of this experiment design. The results, as they are, offer little definitive insight into a positive or negative affirmation of Deeptrader's ability to learn specific behaviours. We can, however, say that the lack of SHCK trader action in the IPRZI trading data demonstrates the importance of our inputs when training the network. Without the exaggerated market imbalance caused by such an agent, there was little chance of the minor

deviations in behaviour caused by natural LOB imbalance being picked up as desirable behaviour by the Deeptrader network.

This is further compounded when we consider that the IPRZI trader only composed a subset of total training data, it is quite likely that other dominant traders have influenced the network's price formation strategy to a stronger degree. More importantly it speaks to a deeper understanding of the behaviourist approach adopted in Wray/Meades/Cliff [56]. To accurately recreate what could be considered the "behaviour" of a human trader, or otherwise, may require the training data to be almost completely composed of that trader's activity. This would entail a level of data collection that would require constant monitoring over very long periods of time, something that is not very commonly afforded to those wishing to study trading in-situ.

Even after this collection, it can be speculated that an agent trained on a single trader's data would not perform well in a real market. Thus far we have seen that the key to generalising to a wide range of scenarios is a breadth of training data for the network. This breadth of data is required for performance but can act to obscure the more specific of our behavioural goals, as has happened in the above experiment. Whether a single human trader could provide the range of data in a manner that would generalise to real market scenarios is up for debate, but quite unlikely.

Another angle from which to view this shortfall would be that of our training goals. The network has of yet only trained to accurately predict prices, but if we are able to encode some measure of the networks "behaviour" into our loss function, we may have a greater chance of incentivising the network to behave in certain ways. This incentivisation may have, to some extent, allowed the network to overcome the inevitable failure seen above when presented with poor training data. An implementation of this, for example, could consist of the inclusion of metrics which quantify an output's adherence to certain market actions. These metrics when included in our loss function would balance the network training to achieve accurate price predictions whilst exhibiting specific behaviour.

Chapter 4

Critical Evaluation

The new trading environment within which Deeptrader finds itself, with unique market dynamics and additional traders, makes clear that a re-evaluation of the networks performance is required. This re-evaluation within this chapter aims to highlight the strengths and weaknesses of Deeptrader whilst speaking to its ability to further generalise to as of yet unencountered scenarios.

4.1 Confidence Interval Test

To assess the performance of Deeptrader within this new Generalised Brownian Motion environment, the variability across sessions must first be accounted for. To do this a confidence interval test was undertaken using the APPT results across 100 independent market sessions. The chosen interval of 90% allows us to compare the bounds of each traders' performance and say with 90% certainty if one trader outperformed another.

This calculation uses the mean ($\bar{x}$) and the standard deviation (σ) of the APPT values, combined with the number of samples (n=100) and the 90th percentile of the normal distribution ($Z_{0.95} = 1.645$). The lower bound for this confidence interval (c_l) is calculated as in Equation 4.1 and the upper bound (c_u) is calculated as in Equation 4.2. The normality assumed in the application of this confidence interval test can here be achieved by the Central Limit Theorem. The APPT value serves as a mean of the distribution of profits. Given the independence of the APPT samples and a quantity of samples exceeding 30, we can infer from the Central Limit Theorem that the resulting distributions approximate the normal distribution.

$$c_l = \bar{x} - \frac{Z_{0.9} \times \sigma}{\sqrt{n}} \tag{4.1}$$

$$c_u = \bar{x} + \frac{Z_{0.9} \times \sigma}{\sqrt{n}} \tag{4.2}$$

4.2 Balanced-Group Tests

To reiterate, these balanced-group tests aim to directly compare the performance of two traders when they represent an equal proportion of the market. This is most level playing field in which two strategies direct utility can be compared.

4.2.1 ZIC

Figure 4.1 displays the results from the balanced-group comparison against the ZIC trader, the confidence interval analysis is displayed in Table 4.1. Deeptrader's lower confidence bound is significantly above the upper confidence bound of ZIC. Thus, we can say with 90% statistical confidence that Deeptrader outperformed ZIC in a balanced-group test. This is not immediately surprising as it can be considered that the random prices quoted by the ZIC trader provide reliable chances to profit to a trader that can successfully interpret the current state of the LOB.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	230.8	27.97	226.2	235.4
ZIC	199.4	23.90	195.4	203.3

Table 4.1: Confidence interval vs ZIC

4.2.2 GVWY

Figure 4.1 displays the results from the balanced-group comparison against the GVWY trader, the confidence interval analysis is displayed in Table 4.2. The mean APPT of both traders lies within the confidence bounds of the other trader. We can interpret this as there being no evidence to suggest a notable difference in performance between Deeptrader and GVWY, with 90% confidence. Despite this, it is interesting to note that the GVWY trader had a higher mean APPT than Deeptrader, but also a larger deviation in prices.
This surprising performance from a relatively simple trader becomes more understandable when referring back to the gradientSHAP analysis present in Figure 3.3. The under-reliance by the network on the limit price feature indicates that Deeptrader's pricing strategy aims for profit irrespective of its limit price (as the BSE system will simply clip that price to the limit price if it quotes a price above). This makes space for a trader closely tracking the current limit prices to profit off of Deeptrader's price quotes.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	242.7	29.73	240.4	250.2
GVWY	245.3	31.10	237.5	247.8

Table 4.2: Confidence interval vs GVWY

4.2.3 SHVR

Figure 4.2 displays the results from the balanced-group comparison against the SHVR trader, the confidence interval analysis is displayed in Table 4.3. Immediately the results display a dominance in this test by the SHVR agent and the confidence interval backs this up. SHVR's lower confidence interval is higher than Deeptrader's upper confidence interval, it can then be stated with 90% confidence that SHVR outperforms Deeptrader.
This highlights the first issue with Deeptrader's understanding of trends, as prices change rapidly in the GBM environment the quotes made by Deeptrader track these prices. Following this, for example, a SHVR quote just below the Best Ask exploits the following increased bid that a Deeptrader agent makes attempting to track upward price trajectory. This leads to a scenario where SHVR agents can regularly extract large profit from opposing Deeptrader agents.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	225.8	34.93	220.1	231.6
SHVR	260.1	38.73	253.8	266.5

Table 4.3: Confidence interval vs SHVR

4.2.4 SNPR

Figure 4.2 displays the results from the balanced-group comparison against the SNPR trader, the confidence interval analysis is displayed in Table 4.4. Deeptrader's lower confidence bound is above the upper confidence bound of SNPR. Thus, we can say with 90% statistical confidence that Deeptrader outperformed SNPR in a balanced-group test. More importantly, the APPT for both sets of traders is comparatively quite small, with a mean of 73 and 57 respectively. This is a result of the SNPR agents tendency to only complete trades near market close, thus there is less time for both agents to make profitable trades. This statistic doesn't reflect the number of high performance outliers that can be seen in Figure 4.2 indicating that the chance of a Deeptrader agent being highly profitable against SNPR is possible, if not probable.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	73.89	52.05	65.33	82.45
SNPR	57.55	7.502	56.31	58.78

Table 4.4: Confidence interval vs SNPR

4.2.5 ZIP

We now move on to the first of our adaptive traders, the agents which have historically rivalled Deeptrader in performance but also have shown in our previous testing (Figures 3.5, 3.6 and 3.7) to respond well to the new GBM environment. Their performance against Deeptrader will therefore make a good point of reference for the networks true utility.

Figure 4.3 displays the results from the balanced-group comparison against the ZIP trader, the confidence interval analysis is displayed in Table 4.5. ZIP's lower confidence bound is above the upper confidence bound of Deeptrader. Thus, we can say with 90% statistical confidence that ZIP outperformed Deeptrader in a balanced-group test. Explain

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	211.7	33.74	206.1	217.2
ZIP	243.3	34.60	237.6	249.0

Table 4.5: Confidence interval vs ZIP

4.2.6 GDX

Figure 4.3 displays the results from the balanced-group comparison against the GDX trader, the confidence interval analysis is displayed in Table 4.6. Deeptrader's lower confidence bound is above the upper confidence bound of GDX. Thus, we can say with 90% statistical confidence that Deeptrader outperformed GDX in a balanced-group test.

This is an important result because it confirms Deeptrader's superiority to a trading agent which had previously been shown to outperform human traders in [20]. This clarifies once more that even in this new, more realistic, model of a market environment that Deeptrader can be considered a trading agent with "super-human" performance.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	249.4	28.53	244.7	254.1
GDX	235.7	27.11	231.2	240.1

Table 4.6: Confidence interval vs GDX

4.2.7 AA

Figure 4.4 displays the results from the balanced-group comparison against the AA trader, the confidence interval analysis is displayed in Table 4.7. AA's lower confidence bound is significantly above the upper confidence bound of Deeptrader. Thus, we can say with 90% statistical confidence that AA outperforms Deeptrader in a balanced-group test.

This result was expected, as even within the more constrained original testing environment, the AA agent significantly outperformed Deeptrader. Given the AA trader's long held, if often challenged, supremacy within the BSE literature it seems that this result is an obvious outcome.

There is however a chance to glean some understanding about Deeptrader from this test. AA's performance suggests that there is some aspect of the agents behaviour which the network cannot replicate from its training data. The most likely culprit being the aggression mechanism which allows the AA trader to react to volatile markets.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	222.5	34.08	216.9	228.1
AA	262.1	37.47	256.0	268.3

Table 4.7: Confidence interval vs AA

When considering the goals of deep learning systems, it must be understood that favourable outcomes can only be achieved by framing the challenge correctly. We can infer then that despite the input of the volatility parameter used by AA (Smith's α), the Deeptrader network isn't suitably learning a facsimile of AA's aggression-based behaviour because we are not incentivising it to do so. The solution to this is reward structures which encode, as a goal, the network's resemblance to AA behaviour. One such structure that easily comes to mind would be the introduction of an aggression parameter in the training data. Quantifying the aggression value of an AA trader at the time they made an order.

Another such structure would result from an abstraction of the concept of AA's aggression and GDX's belief function. The commonality of these systems is a basis of the altering of price based on the likelihood of an order being accepted. It can then be considered that a restructuring of the Deeptrader network to maximise not the difference from a stated price, but the probability of an order being accepted. This alteration would also significantly increase the size of the training data as it would allow all trader quotes to be used, not just quotes which became trades. The quotes that became trades could then exist as positive reinforcement and those which didn't act to discourage the network from certain behaviour.

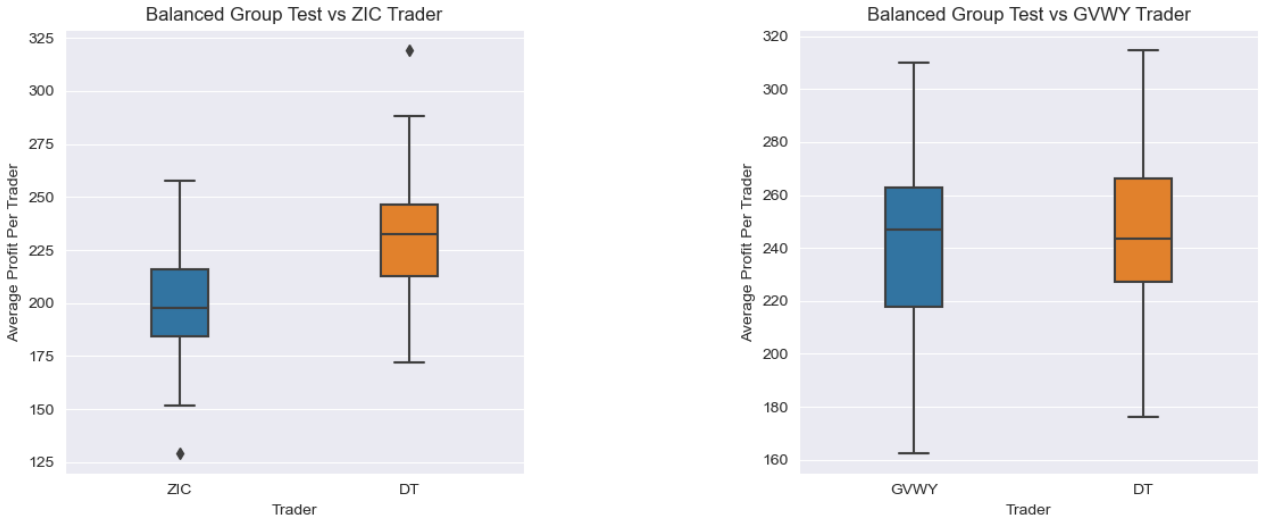

Figure 4.1: Balanced Group ZIC and GVWY

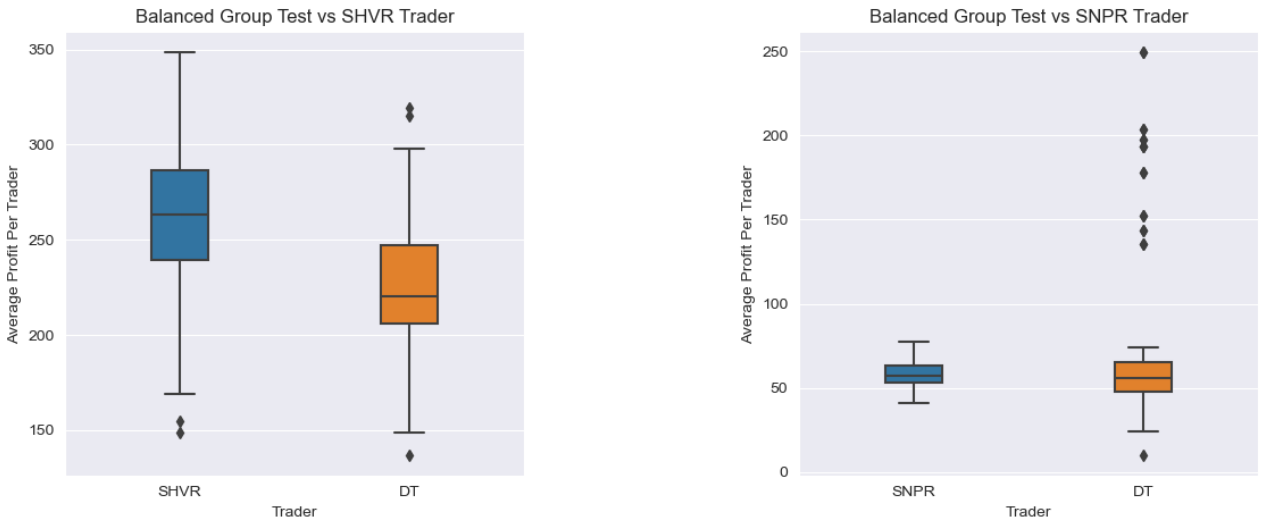

Figure 4.2: Balanced Group SHVR and SNPR

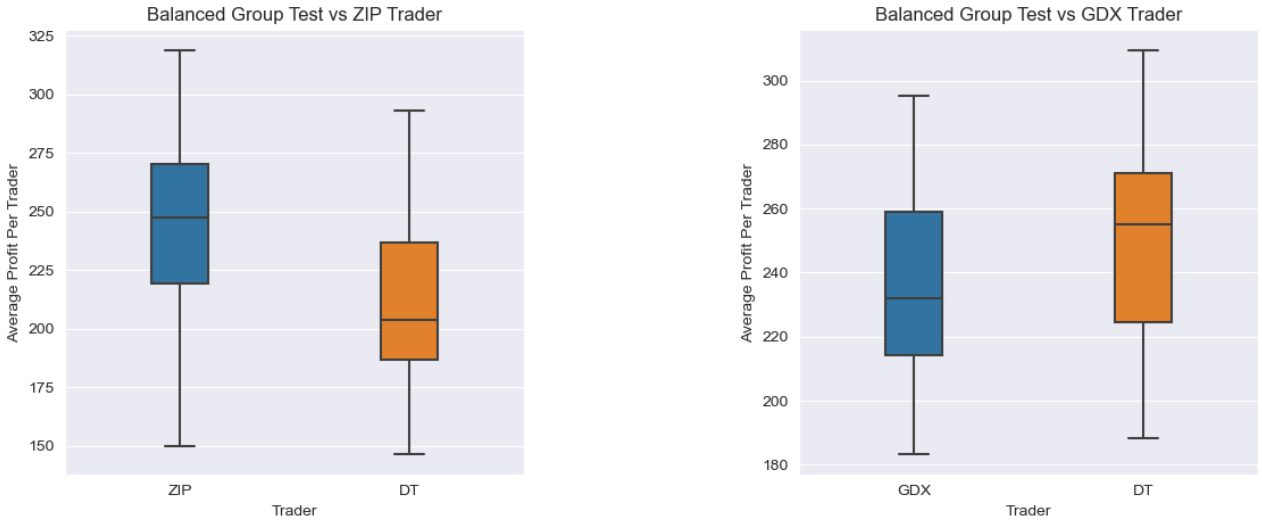

Figure 4.3: Balanced Group ZIP and GDX

Figure 4.4: Balanced Group AA

4.3 One-In-Many Tests

The One-In-Many tests conducted here allow us to measure the performance of a trading agent that exists as the minority of a market, akin to the introduction of a single strategy to an established market environment. The commonality amongst these results is the great variance that exists within the APPT results of Deeptrader, sessions were as likely to report extremely high profits as they were to result in negligible profit. This leads to a set of results with an impressive mean APPT but a standard deviation in these results regularly taking values of ∼200. In comparison all-but-one of the majority class traders have a lower mean APPT but extremely low variance in these results, highlighted by across-the-board standard deviations of under 10.

This trend was present in the original work of Deeptrader and whilst the conclusions reached below about statistical performance comparisons are valid, they must be interpreted within the context of highly irregular profitability. This lack of certainty quite clearly impedes the real-world value of a system such as Deeptrader. Especially in an industry such as that built around algorithmic trading - where the reliability of returns often eclipses, in importance, the possible maximum returns a strategy could exhibit.

4.3.1 ZIC

Figure 4.5 displays the results from the one-in-many comparison against the ZIC trader, the confidence interval analysis is displayed in Table 4.8. Deeptrader's lower confidence bound is above the upper confidence bound of ZIC. Thus, we can say with 90% statistical confidence that Deeptrader outperformed ZIC in a one-in-many test.

Trader	**Mean**	**Standard Deviation**	c_l	c_u
Deeptrader	259.8	216.0	224.2	295.3
ZIC	189.1	8.511	187.7	190.5

Table 4.8: OIM Confidence interval vs ZIC

4.3.2 GVWY

Figure 4.5 displays the results from the one-in-many comparison against the GVWY trader, the confidence interval analysis is displayed in Table 4.9. Deeptrader's lower confidence bound is above the upper confidence bound of GVWY. Thus, we can say with 90% statistical confidence that Deeptrader outperformed GVWY in a one-in-many test.

Trader	**Mean**	**Standard Deviation**	c_l	c_u
Deeptrader	288.5	225.8	251.4	325.7
GVWY	176.7	4.946	175.9	177.6

Table 4.9: OIM Confidence interval vs GVWY

4.3.3 SHVR

Figure 4.6 displays the results from the one-in-many comparison against the SHVR trader, the confidence interval analysis is displayed in Table 4.10. The mean APPT of both traders lies within the confidence bounds of the other trader. We can interpret this as there being no evidence to suggest a notable difference in performance between Deeptrader and SHVR in a one-in-many test, with 90% confidence.

Trader	**Mean**	**Standard Deviation**	c_l	c_u
Deeptrader	259.6	207.6	225.5	293.8
SHVR	246.8	6.187	245.7	247.8

Table 4.10: OIM Confidence interval vs SHVR

4.3.4 SNPR

Figure 4.6 displays the results from the one-in-many comparison against the SNPR trader, the confidence interval analysis is displayed in Table 4.11. Deeptrader's lower confidence bound is above the upper confidence bound of SNPR. Thus, we can say with 90% statistical confidence that Deeptrader outperformed SNPR in a one-in-many test.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	163.2	202.9	129.9	196.6
SNPR	55.5	3.489	54.97	56.12

Table 4.11: OIM Confidence interval vs SNPR

4.3.5 ZIP

Figure 4.7 displays the results from the one-in-many comparison against the ZIP trader, the confidence interval analysis is displayed in Table 4.12. The mean APPT of both traders lies within the confidence bounds of the other trader. We can interpret this as there being no evidence to suggest a notable difference in performance between Deeptrader and ZIP in a one-in-many test, with 90% confidence.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	250.0	210.8	215.3	284.7
ZIP	223.6	8.872	222.1	225.1

Table 4.12: OIM Confidence interval vs ZIP

4.3.6 GDX

Figure 4.7 displays the results from the one-in-many comparison against the GDX trader, the confidence interval analysis is displayed in Table 4.13. The mean APPT of both traders lies within the confidence bounds of the other trader. We can interpret this as there being no evidence to suggest a notable difference in performance between Deeptrader and GDX in a one-in-many test, with 90% confidence.

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	192.2	180.1	162.6	221.8
GDX	168.2	5.160	167.3	169.0

Table 4.13: OIM Confidence interval vs GDX

4.3.7 AA

Here we reach the notable exception to the broad statement of results highlighted at the start of this section. As such it warrants a more exacting examination.

Figure 4.8 displays the results from the one-in-many comparison against the AA trader, the confidence interval analysis is displayed in Table 4.14. Deeptrader's lower confidence bound is above the upper confidence bound of AA. Thus, we can say with 90% statistical confidence that Deeptrader outperformed AA in a one-in-many test.

Viewing these results, the first thing visible of immediate importance is the standard deviation of the AA trader. Compared to the other results in this section it is many times larger, at around 112. This suggests that this trading environment has created great variability for both the Deeptrader and AA agent. When compared to Figure 4.8 we see that a large majority of the testing data received skews towards the lower range of performance, with a significant portion of that APPT existing around the 0 profit mark.

When the data was inspected again, it could be seen that the cause of this skew was the subset of sessions where 0 APPT was recorded for both traders. Of the 100 sessions used for this comparison, 42

Trader	Mean	Standard Deviation	c_l	c_u
Deeptrader	180.5	225.0	143.5	217.5
AA	120.7	112.4	102.2	139.2

Table 4.14: OIM Confidence interval vs AA

of those sessions recorded 0 APPT for their outcome. These 0 profit sessions came as a result of intra-marginal AA sellers with low aggression. As prices were altered throughout the session, they continuously cancelled previous quotes and posted quotes far above their competitive equilibrium estimate. Thus, very few to no trades were completed. Whilst this unexpected behaviour is itself interesting, it makes evident that the two Deeptrader agents were not able to extract profit from these unusual scenarios. The behaviour being exhibited was suitably outside of the bounds of their training data in a way that resulted in the network not taking advantage of the imbalance between supply and demand. Thus, even in a GBM environment we can that the generality of Deeptrader is bounded by the breadth of behaviour it has previously observed.

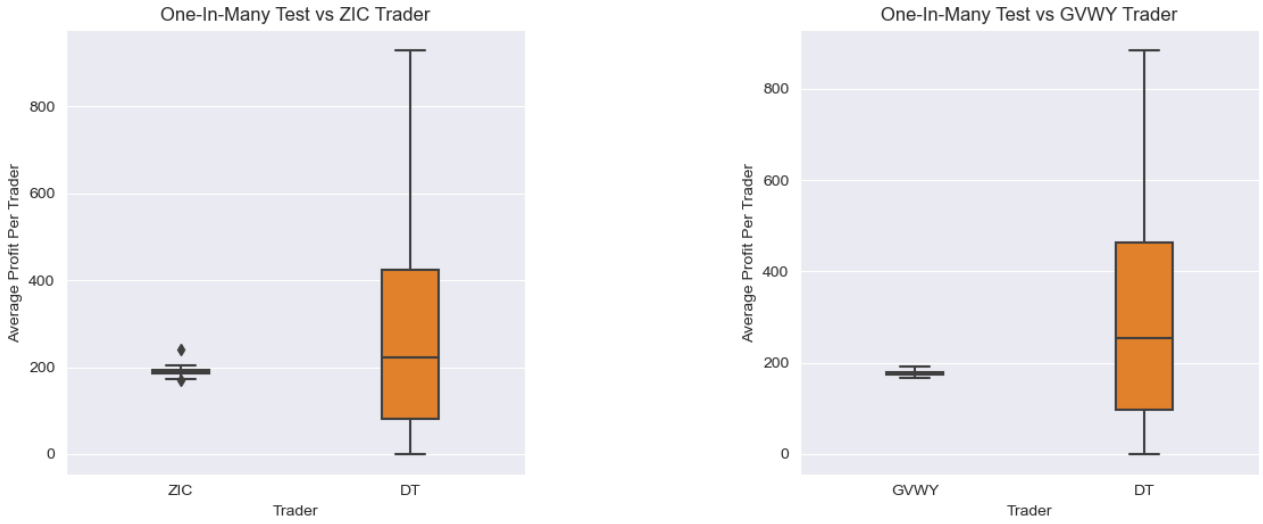

Figure 4.5: One-In-Many ZIC and GVWY

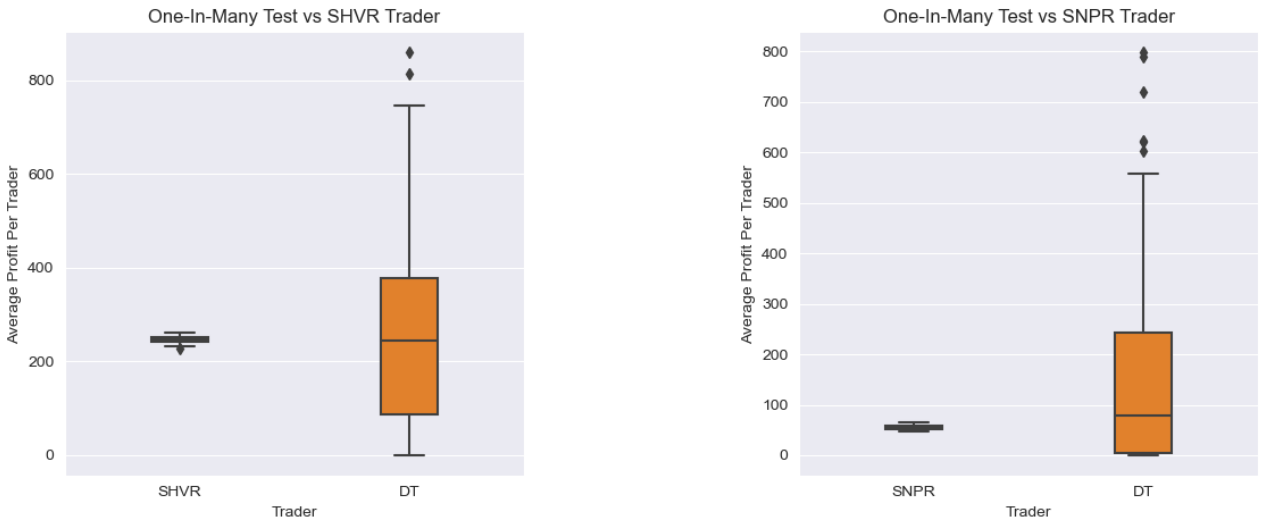

Figure 4.6: One-In-Many SHVR and SNPR

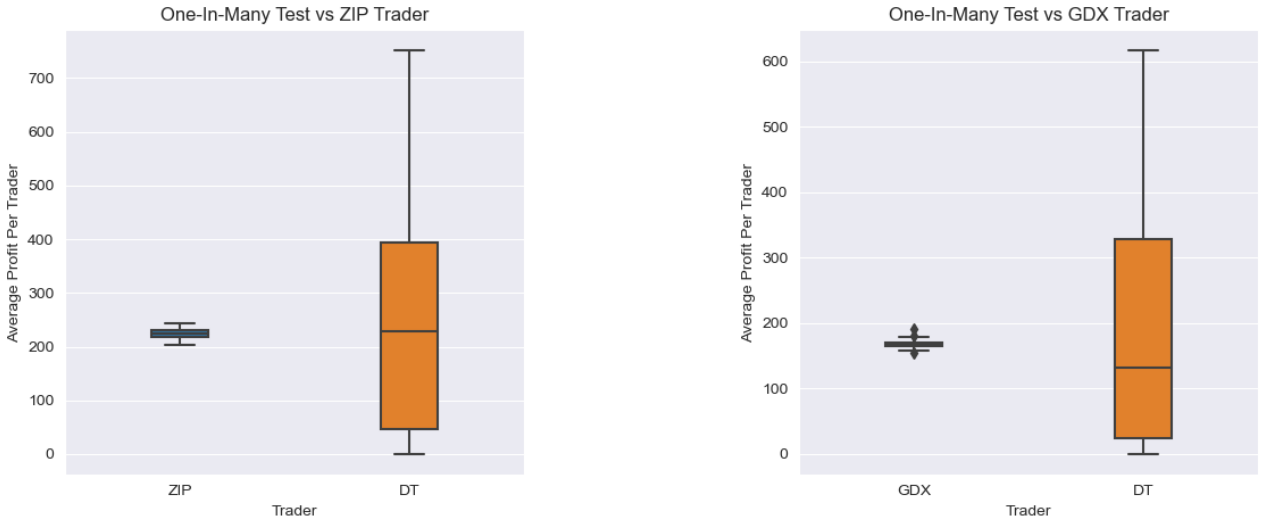

Figure 4.7: One-In-Many ZIP and GDX

Figure 4.8: One-In-Many AA

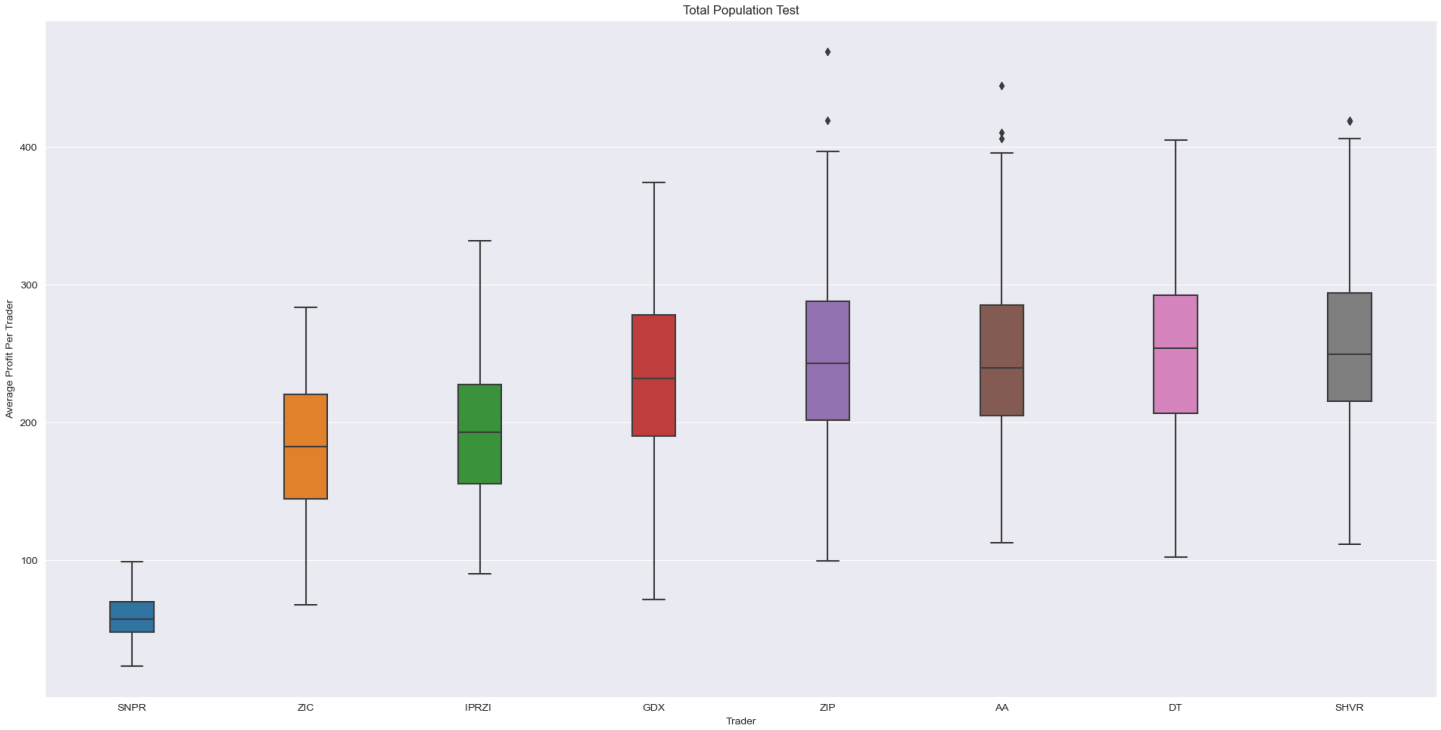

Figure 4.9: Total Population Test

4.4 Total Population Test

Diverging from the established testing methods of Tesauro and Das [51], we can further explore Deeptrader in a test involving the total population of traders. This method forgoes direct comparisons between individual traders to give a more general idea of performance in a wider market. This approximates the more realistic trading scenarios where there exist many traders each employing their own strategy for maximum gain. The structure of this experiment then is composed of 10 of each of the trading agents we have thus far explored: ZIC, GVWY, SHVR, SNPR, IPRZI, ZIP, GDX, AA and Deeptrader. The comparison is shown in Figure 4.9.

From this we can immediately gather that Deeptrader significantly outperforms SNPR(mean APPT = 57.9), ZIC(mean APPT =183.8) and IPRZI(mean APPT=192.1). With a mean APPT of 247.9, the network results in comparable performance to the ZIP, AA and GDX traders - the three adaptive agents this work sought to pit Deeptrader against. The notable surprise within this comparison is the performance of SHVR, on par with a suite of far more complex strategies. In a market populated by different strategies vying for profit the SHVR agent utilised the differences between strategies to extract a high mean APPT at 255.0. This exemplifies the wisdom that a simple strategy based on mathematical and market principles can still be highly profitable in complex scenarios.

We can investigate this more closely by providing a confidence interval test on the high-performing traders present. The results of this test, shown in Table 4.15, actually indicate only 1 statistically significant outcome. This outcome indicates that the SHVR agent outperforms GDX with a 90% statistical confidence level. Outside of that findings we can see that all other confidence bounds overlap, thus we can say with 90% statistical confidence that there was no discernible difference in performance between these 5 high-performing trading agents. The performance of Deeptrader in a total population test can therefore be deemed comparable to GDX, ZIP, AA and SHVR.

We can extrapolate from this experiment that the current iteration of the Deeptrader network has a reasonable performance in a market consisting of many trading strategies. It follows from that then that the Deeptrader network has shown signs that it may be able to profitably trade in a real market

Trader	Mean	Standard Deviation	c_l	c_u
GDX	231.7	64.57	221.1	242.3
ZIP	250.0	66.16	239.1	260.9
AA	248.0	65.48	237.2	258.7
DT	247.9	67.74	236.8	259.1
SHVR	255.0	61.59	244.8	265.1

Table 4.15: Confidence interval test for the 5 high-performing traders

environment. This of course is based on the assumption that Generalised Brownian Motion with drift is a realistic model of stock price movement. This argument could be strengthened by analysing the performance of Deeptrader against historic stock price data, such that its value could be compared to the wider work of quantitative finance algorithmic trading.

4.5 Shapley Analysis

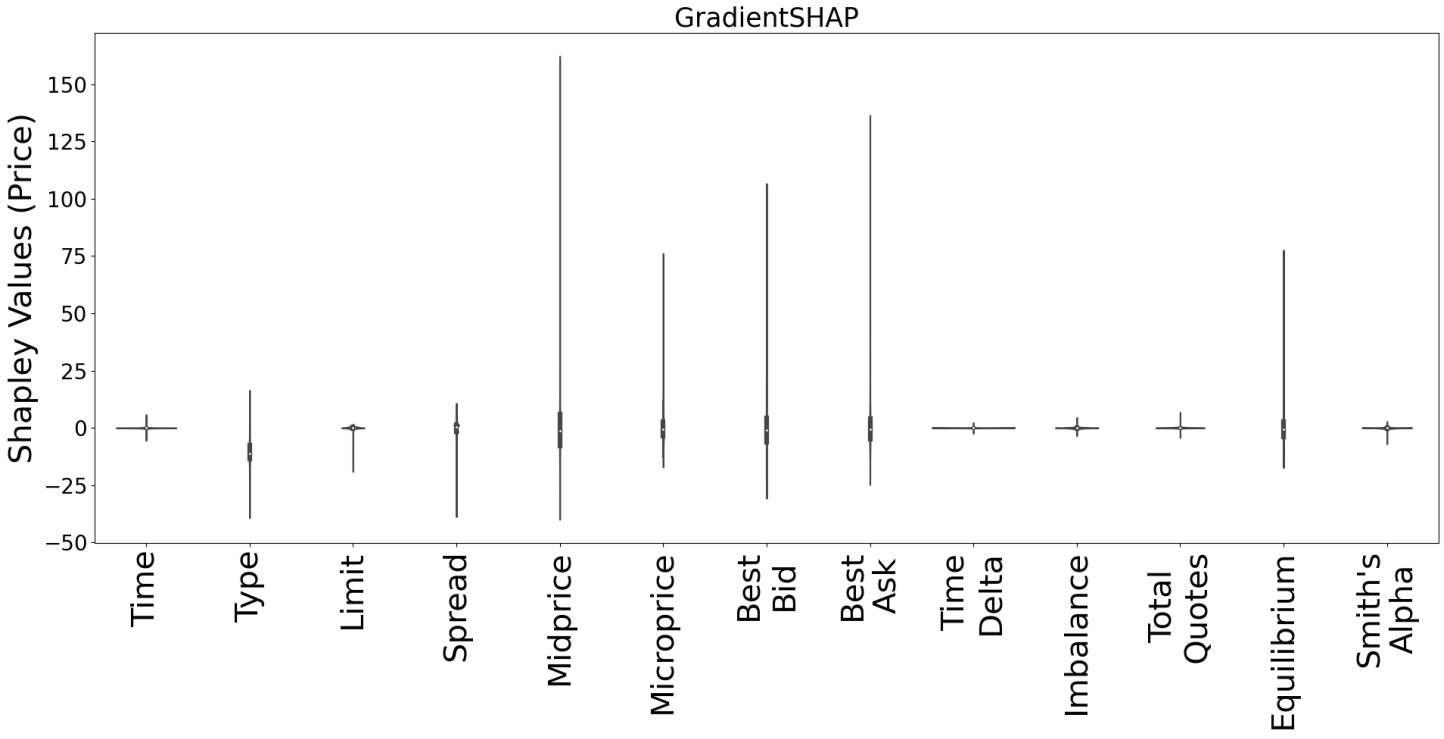

Figure 4.10: A violin plot of the distribution of Shapley values for the Deeptrader network within a Generalised Brownian Motion environment.

The application of a second gradientSHAP analysis allows us to contrast the previous results to this new market environment. The violin plots are shown in Figure 4.10. The first thing of note to discuss is the scale of this new data. The changes in price caused by each feature can now reach in size up to 150 above the base price (base price). This is an artifact of the much greater range of prices that can be taken in a GBM market environment, this range can be highlighted by the minimum and maximum trading prices displayed in the training data of 67 and 786 respectively.

Of continued importance is the lack of reliance on temporal features, discussed in conjunction with the previous iteration of our gradientSHAP results. The evidence continues to suggest that the Deeptrader network isn't parsing its inputs in a manner that utilises relationships across the course of a market session. Whilst this result was previously interesting, it is even more notable here due to the volatility present in markets where prices can alter rapidly over the course of a session. It would follow that if

the Deeptrader network were able to extract trends from its input data, not only would its performance be higher in the balanced group tests, but the features of: Time, Time Delta and Total Quotes would be more significantly utilised by the network. This finding suggests that either the network structure has not been created in a manner suitable to exploit trends or that the current temporal features are not fit for purpose, perhaps the information they convey is encoded in some other input to the network. This implies that further work could effectively remove or replace these features if dataset size became a priority.

In comparison to the previous Shapley analysis, we can now see the heightened significance of the Equilibrium estimate feature. Given the great variation in prices, it seems logical that having an accurate approximation of a market's equilibrium price would bear a greater importance than in scenarios with less extreme price movement. One interesting observation that can be tied to the increased utilisation of the equilibrium estimate is the relative lack of reliance on Smith's Alpha. In the original Deeptrader network, Smith's Alpha had a minor but noticeable impact on the Shapley values. In this new work the corresponding Shapley values are strongly clustered around zero, implying little impact. This can be considered a symptom of the number of previous trades used to calculate Smith's Alpha, in line with Vytelingum's work we utilised the last 5 trades to create this measure of volatility. The resulting feature in the dataset had a mean of 0.04 and a standard deviation of 0.03, implying extremely low volatility. We know this is not the case and thus can conclude that the sliding window within which this measure was computed was too small to capture an accurate representation of market volatility.

The increased importance of the midprice feature here plays, to some degree, the same role as the equilibrium estimate in that it helps the Deeptrader network more accurately navigate a market with rapidly changing prices. The midprice is a rather simplistic estimate of the current market's equilibrium and thus when we consider the combined importance of both the midprice and equilibrium estimate here, it seems a logical next step to further consider if more complex measures of market equilibrium could benefit the network's price discovery strategy in future.

A final point of interest is the balance of importance between Best Bid and Best Ask in this new market. In our previous Shapley analysis, Best Bid played a greater role in the network's decision making than the Best Ask. In contrast, here the Best Ask has a greater impact on output prices than its counterpart. This counter's Meades' assertion that the Best Bid was the most important of the two features. It instead indicates that given the supplemental input of other features, the two are essentially interchangeable. Any given instance of a trained network may choose to favour one or the other but this is simply a result of the random weight initialisation that occurs before a neural network is trained.

4.6 Timings

The speed of an algorithmic trader is of the utmost importance in the high frequency market ecosystem we find ourselves in, one such quote from the chief economist of the Bank of England best exemplifies this mentality "In a high-speed, co-located world, being informed means seeing and acting on market prices sooner than competitors. Today, it pays to be faster [than the average bear], not smarter. To be uninformed is to be slow" [23].

What it means to be fast varies greatly based on the intended use cases but in the case of modern financial exchanges we can greatly summarise with the quote "The time scales involved are astonishingly small. Order latencies are measured in milliseconds (one thousandth of a second), microseconds (one millionth of a second), and in some settings even nano-seconds (one billionth of a second)" [40]

The execution speed of the various trading agents within the BSE ecosystem has as of yet gone un-examined during our testing. Whilst these execution speeds bear no relevance to the sequentially processed simulation of BSE, they do allow us to draw conclusions about the viability of a deep learning strategy in a real high frequency market. As such timings were recorded over a number of market sessions and the key results are presented here.
The respond function of a BSE trader is used to update the internal state of a trader in reaction to any changes to the state of the LOB, this function usually encompasses the calculation of the learning updates required for any of the adaptive traders. The getOrder function of a BSE trader is used to place

an order to the LOB, this order will be a product of the customer limit order and the current internal state of the trader.

In viewing the results, presented in Table 4.16, the timing of Deeptrader's respond function is of immediate attention. Having the shortest response time compared to the other adaptive traders initially seems impressive. It is however overshadowed by a getOrder function which dwarfs the other adaptive traders tested in length. The combination of these two values leads to the conclusion that whilst Deeptrader rapidly responds to market activity, it would then struggle to produce an order to take advantage of any insights within a reasonable time.

The result however speaks to a deeper truth about the Deeptrader network, the respond function of Deeptrader is only being used to keep a list of recent transactions for the equilibrium estimate feature. It serves no purpose to maintain an internal representation of the market behaviour. Rather instead of tracking the market, all insights are gathered from the snapshot of the LOB used as input to the network. The calculation of the relevant input features and their application to the network then explains the significant time increase present in the getOrder function.

This singular application of summary statistics at the point of order seems the root cause of Deeptrader's previously noted issue with navigating trends in market behaviour.

This timing also brings to mind one further question: Is the inherent complexity of an LSTM deep learning network going to create a lower bound on the execution time of any such trader? Whilst this work hadn't explicitly focused on efficiency in the source code produced, it does seem an avenue worth exploring.

	Execution Time (Microseconds (x10^{-6}))	
	getOrder function	respond function
Deeptrader	352.67	0.11
ZIP	8.19	13.59
GDX	212.66	1.06
AA	6.49	3.02

Table 4.16: Function Timings for the adaptive traders

Chapter 5

Conclusion

This work now reaches a point that warrants an examination of the original goals set out in Chapter 1 and the extent to which those goals have been met.

The first success comes in an independent replication of Deeptrader and Deeptrader 2. This replication validates a number of the findings present in the work of Wray and Meades. This hopefully providing a, now undeniable, testament to the functionality and performance of the original works. Furthermore, the implementation provided here uses a different deep learning framework, Pytorch, as opposed to the original Keras/Tensorflow, indicates that the core concepts of Deeptrader's success are universal and not tied to the characteristics of any particular framework or style.

We furthered this validation through the introduction and careful examination of the network through deep learning interpretability methods. This came in the form of the Python Captum library and its constituent methods of gradientSHAP and kernelSHAP. Where gradientSHAP uncovered flaws in Meades' original ablation study it also allowed us to uncover some underlying truths in relation to the structure of the Deeptrader network and its decision-making process. Continuing from this, the kernelSHAP methods introduced allowed a more specific view of network output but also brought the Deeptrader network one step closer to initial goal stated in Wray/Meades/Cliff [56], a network trained on human data and successfully utilised within a real market environment. This came in the form of the functionality provided by kernelSHAP in explaining each model output. This feedback when included in the full network pipeline would allow potential future users to crucially understand a model which had been, till this point, completely opaque. Given the growing importance of human understanding in the field of deep learning, this feels like a step in the right direction when compared to the high-performing but misunderstood initial iterations of the network.

In striving to bring a wider range of literature surrounding algorithmic traders into the fold of the Deeptrader ecosystem, the first point of consideration was a more realistic model of the markets in which our experiments were contained. This led to the adoption of Generalised Brownian Motion with Drift, a model of stock price movement whose variability underpinned many of the challenges of this work. The initial experiments showed that Deeptrader in its original state was not up to the task of navigating an environment with such volatility. This led to the creation of the dataset comprised of data from markets simulated with Generalised Brownian Motion with Drift. The subsequent re-training of the network resulted in a vastly different scenario, gone were training times and inevitable overfitting of the original training sessions. The network could now benefit from extended periods of training and its performance within market experiments mirrored these sentiments. The question this experience brings to mind, however, is the true utility of network that must be trained and re-trained for each new alteration in market structure. Perhaps something could be learnt from the adaptive trading agents of ZIP, GDX and AA, who maintained their respectable performances even in market conditions which were never conceived as part of their original design.

Continued consideration of the literature took the opportunity, in the work of ISHV and PRZI, to examine Deeptrader's performance in the presence of a trading agent about which it had no prior knowledge. This required the creation of a wholly new, if slightly derivative, trading agent in the form of IPRZI - a zero intelligence trader parameterised by a measure of limit order book imbalance. The surprising

results here indicating that Deeptrader generalised well to this new trader and successfully outperformed it across multiple scenarios. These results however do not exist in a vacuum and the consideration must be made that this "unseen" trader exists as a synthesis of three strategies (GVWY, ZIC, SHVR) which exist within Deeptrader's training data. This caveat dampens the importance of such results and implores a further investigation of the matter with another, truly unique, "unseen" trading agent.

Looking further into the work carried out around the IPRZI agent, there was also an attempt to address the behaviourist mentality adopted in Wray/Meades/Cliff. Could Deeptrader really learn, not just a strategy for profitability but specific behaviours present in trading agent data. This work required the creation of a shock trader that would provide suitable LOB imbalance for the agents to respond to. The lack of agent response to this new imbalance was not surprising given the flawed experimental design but leaves room for a re-analysis with a more accurate methodology. Despite the failings of this contribution, it highlighted some important limitations when considering the data collection required to implement Deeptrader in realistic scenarios. The results of this experimentation also gave the first indications that the deep learning "goal" as such was something to remain very aware of. Unless the model was incentivised to behave in specific ways, it would simply act to make accurate price predictions.

The evaluation of this work initially consisted of a confidence interval analysis. The use of confidence intervals here allowed a more accurate comparison with the previous works' outcomes and provided statistical evidence for claims made about the networks performance. The balanced-group tests provided evidence for Deeptrader's superiority against a number of trading agents, most notably a trading agent (GDX) which had previously been shown to have "super-human" trading performance. From this we can infer that the Deeptrader network within a Generalised Brownian Motion market environment can also be classified to have "super-human" trading performance, although the verification of this would require a human study with professional sales traders which exists outside the scope of this work. The balanced-group tests also introduced some notable exceptions in results to the original network. These exceptions can be attributed largely to Deeptrader's inability to track market trends, instead relying on the individual LOB snapshot at the time of order.

The confidence interval work conducted on the one-in-many experiments paints an overall consistent picture. The results indicate a Deeptrader agent which can regularly extract significant profits, this is however hampered by the extreme variability of outcomes. This lack of reliability places a serious thorn in the side of any future utility Deeptrader may have in real world scenarios. These outcomes do however suggest that a widening of scope in data collection could remedy the issue. Perhaps the inclusion of one-in-many market scenarios in the training data could accustom the network to that style of market behaviour. This train of thought does however circle back to the key issue of utility - is the consistent widening of our input data scope combined with re-training a sustainable practice? This dilemma seems even more pressing when one considers the seemingly infinite combinations of market behaviour that occur on real financial exchanges in a given day.

Our second Shapley analysis helps reinforce our beliefs surrounding the Deeptrader network's "understanding" of market trends. In addition, it highlights the importance of the inputs that attempt to pinpoint the market's current equilibrium point, giving room for a more in-depth further exploration of network input features.

Undertaking a total population test was a novel introduction to this work in comparison to the established testing methods of Tesauro and Das [51]. Whilst the limitations of this method obviously obscure any direct comparisons of performance, it did allow the work to convey an idealised representation of a real market environment. The facets of a real market represented here were the wide range of traders using different strategies but also a more active and rapidly changing LOB driven by the interplay between numerous methods, something less directly experienced in the previous set of tests. This methodology allowed us to suggest Deeptrader's ability to hold its own against a number of the more complex adaptive traders present whilst reasonably outperforming the majority of the simpler strategies present within BSE.

From there our final analysis came in the form of a timed comparison of the current adaptive traders in relation to the Deeptrader network. These comparisons provide an interesting insight into each strategies mechanism cannot be taken at face value. The educational nature of BSE has led to each implementation being made with comprehension at the forefront and thus none can be said to be optimised for efficiency.

Despite this, a clear conclusion about the significant execution time of the Deeptrader network can be made. There may of course be methods that exists to optimise the efficiency of the neural network structure, as of yet unexplored, but it can be reasonably assumed that the sheer complexity of a modern LSTM neural network places a limit on the resulting trader execution time that exceeds the time frames relevant to the world of High Frequency Trading.

This work then can be deemed to have achieved the original goals that were initially outlined. The Deeptrader agent can now be classified as high performing in a much wider range of realistic market scenarios. The failings of Deeptrader uncovered here can now be further inspected via a wide avenue of different methods and concepts. A brief overview of these avenues of inspection are listed below, of which some will comprise the basis for the continued work undertaken in relation to this dissertation to produce a conference paper that explores the application of Deeptrader to more general scenarios.

5.1 Future Work

The most pressing of Deeptrader's current limitations is the inability to comprehend and track market trends over time. This can be considered a symptom of the networks lack of continued internal state tracking the markets activity, instead opting for individual instances of market behaviour. This specific issue rather stems from a lack of adaptability that presents a wider set of issues for the network. Once Deeptrader is trained and deployed it can only perform within the framework of the training data it has been supplied. These issues can be further explored by modifying the network in a few key ways.

The inability of the network to process temporal relationships, as demonstrated in the Shapley value analysis, indicates that the introduction of input data across time to a single order instance may be beneficial. The method that first comes to mind is the utilisation of the sequence processing capabilities of the LSTM network. Therefore, each input to the network would be accompanied by the preceding trades which had been executed by all traders on the LOB. This would require the re-factoring of the BSE setup used for data collection. Consisting of the creation of blocks of orders which the network could intake at each training step. This would require a significant increase in the amount of training data collected to ensure the network doesn't overfit to specific market trends exhibited within common trading behaviour.

Deeptrader's adaptability could be further supplemented with the introduction of online learning. Online learning is a branch of deep learning study which employs methods to continue to train the network during its operational deployment. These methods would allow the "training" of the network to continue throughout its time trading, with new orders applied back to train at each step of operation.

This brings us to the concept of the wider machine learning goals we wish to undertake. This is key because unless we can mathematically encode our expectations of the network's behaviour, we cannot ensure any specific learning outcomes. A network incentivised to correctly predict a single price may lack the ability to learn specific behaviours or struggle to adapt to new scenarios. To achieve these goals, we can consider a paradigm shift in our training regime, one such shift as previously suggested is the move to a probabilistic outcome where the Deeptrader network aims to maximise the probability of order acceptance. This would not only broadly replicate the mechanisms that underpin the success of the AA and GDX traders but would significantly increase the scope of our training data. Allowing accepted orders to provide positive reinforcement for the network whilst unaccepted orders discourage in-advantageous actions.

Going further afield, the question bears asking, could techniques employed by state-of-the-art models for stock price prediction provide a suitable benefit within the BSE environment. One such concept that provides a suitable intrigue is the use of hybrid deep learning models, where the parsing of the market structure and the parsing of temporal trends are separated. This could be implemented in this instance using, for example, a convolutional neural network to interpret the state of the LOB. The high-level features output by this convolutional network could then be used as input for an LSTM network designed to track the change of those features over time. This methodology would hopefully provide a final network greater than the sum of its parts. It does however present the challenge of a combined network in which the complexity of training and testing may hinder accurate study.

Another such inspiration from the field of modern algorithmic trading is the recent revelation of deep reinforcement learning systems. These systems train an agent to directly navigate and interact with a state-based environment in an attempt to gain a reward. The question of BSE's suitability as a simple enough environment to adequately model as a state space is an important one, but the thought remains. Could the next iteration of Deeptrader exist as an agent whose learning process applies directly to its trading behaviour, not simply the correct prediction of a profitable price?

5.2 Parting Words

This work can be thought of as having, to the extent of my ability, explored the intricacies and limitations of not only the Deeptrader agent, but more generally, the wider suitability of deep learning methods within the world of Bristol Stock Exchange. In an attempt to unite a range of disparate interests, it is hoped that a breadth of thought and methodology was encountered in this evaluation of the notable work of Wray and Meades. In the spirit of all research work, this project can be considered to have created more questions than it set out to answer. It does however aim to serve as a comprehensive starting point for the future work undertaken in this unique sub-field of algorithmic trading and deep learning techniques.

Bibliography

[1] Findings regarding the market events of may 6, 2010. Technical report, U.S. Commodity Futures Trading Commission, U.S. Securities Exchange Commission, 2010.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Robert Almgren and Neil A Chriss. Optimal liquidation. *Available at SSRN 53501*, 1997.

[4] Johannes Breckenfelder. Competition among high-frequency traders, and market quality. 2019.

[5] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[6] G Church and D Cliff. A simulator for studying automated block trading on a coupled darks/lit financial exchange with reputation tracking. In *Proceedings of the European Modelling and Simulation Symposium*, 2019.

[7] Dave Cliff. Minimal-intelligence agents for bargaining behaviors in market-based environments. *Hewlett-Packard Labs Technical Reports*, 1997.

[8] Dave Cliff. Bristol stock exchange. https://github.com/davecliff/BristolStockExchange, 2012.

[9] Dave Cliff. BSE: A minimal simulation of a limit-order-book stock exchange. *arXiv preprint arXiv:1809.06027*, 2018.

[10] Dave Cliff. Parameterised-response zero-intelligence (PRZI) traders. *Available at SSRN 3823317*, 2021.

[11] Dave Cliff and Linda Northrop. The global financial markets: An ultra-large-scale systems perspective. In *Monterey workshop*, pages 29–70. Springer, 2012.

[12] Rajarshi Das, James E Hanson, Jeffrey O Kephart, and Gerald Tesauro. Agent-human interactions in the continuous double auction. In *International joint conference on artificial intelligence*, volume 17, pages 1169–1178. Lawrence Erlbaum Associates Ltd, 2001.

[13] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016.

[14] Marco De Luca and Dave Cliff. Human-agent auction interactions: Adaptive-aggressive agents dominate. In *Twenty-second international joint conference on artificial intelligence*. Citeseer, 2011.

[15] Luca Di Persio and Oleksandr Honchar. Recurrent neural networks approach to the financial forecast of google assets. *International journal of Mathematics and Computers in simulation*, 11:7–13, 2017.

[16] Abdelmoula Dmouj. Stock price modelling: Theory and practice. *Masters Degree Thesis, Vrije Universiteit*, 2006.

[17] Julia Dressel and Hany Farid. The accuracy, fairness, and limits of predicting recidivism. *Science advances*, 4(1):eaao5580, 2018.

[18] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. The role of trust in automation reliance. *International journal of human-computer studies*, 58(6):697–718, 2003.

[19] David Easley and Maureen O'Hara. Market microstructure. *Handbooks in operations research and management science*, 9:357–383, 1995.

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

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

[22] M Ugur Gudelek, S Arda Boluk, and A Murat Ozbayoglu. A deep learning based stock trading model with 2-d cnn trend detection. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[23] Andrew G Haldane. The race to zero. In *The Global Macro Economy and Finance*, pages 245–270. Springer, 2012.

[24] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250, 2000.

[25] Lee Jacobson. Introduction to artificial neural networks - part 1. https://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7.

[26] Weiwei Jiang. Applications of deep learning in stock market prediction: recent progress. *arXiv preprint arXiv:2003.01859*, 2020.

[27] Surya Mattu Julia Angwin, Jeff Larson and Lauren Kirchner. Machine bias: There's software used across the country to predict future criminals. and it's biased against blacks. *Propublica*, 2016.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Andrei Kirilenko, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, 72(3):967–998, 2017.

[30] Andrei A Kirilenko and Andrew W Lo. Moore's law versus murphy's law: Algorithmic trading and its discontents. *Journal of Economic Perspectives*, 27(2):51–72, 2013.

[31] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[32] Arthur le Calvez and Dave Cliff. Deep learning can replicate adaptive traders in a limit-order-book financial market. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1876–1883. IEEE, 2018.

[33] Hao Li, Yanyan Shen, and Yanmin Zhu. Stock price prediction using attention-based multi-input lstm. In *Asian Conference on Machine Learning*, pages 454–469. PMLR, 2018.

[34] Yang Li, Wanshan Zheng, and Zibin Zheng. Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access*, 7:108014–108022, 2019.

[35] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*, 2017.

[36] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] Matthew Meades. An investigation in to the success of deep learning trading agents. Master's thesis, 2020.

[38] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 2021.

[39] Anna A Obizhaeva and Jiang Wang. Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1–32, 2013.

[40] Maureen O'Hara. High frequency market microstructure. *Journal of Financial Economics*, 116(2):257–270, 2015.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[42] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[43] Michael Rollins and Dave Cliff. Which trading agent is best? using a threaded parallel simulation of a financial market changes the pecking-order. *arXiv preprint arXiv:2009.06905*, 2020.

[44] John Rust, Richard Palmer, and John H Miller. Behaviour of trading automata in a computerized double auction market. Santa Fe Institute, 1992.

[45] Tirthajyoti Sarkar. Brownian motion with python. https://github.com/tirthajyoti/Stats-Maths-with-Python/blob/9e5d12f6e6bf683082ff377e55915ba141212d02/Brownian-motion-with-Python.ipynb, 2020.

[46] Lloyd S Shapley. *Notes on the N-person Game–II: The Value of an N-person Game*. Rand Corporation, 1951.

[47] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2017.

[48] Justin Sirignano. Deep learning for limit order books, 2016.

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

[50] 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.

[51] 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*, pages 591–598, 2002.

[52] Kanupriya Tibrewal. Can neural networks be traders? explorations of machine learning using the bristol stock exchange. Master's thesis, 2017.

[53] Guido van Rossum. What's new in python 3.0. https://docs.python.org/3.5/whatsnew/3.0.html.

[54] Perukrishnen Vytelingum. *The structure and behaviour of the Continuous Double Auction*. PhD thesis, University of Southampton, 2006.

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

[56] Aaron Wray, Matthew Meades, and Dave Cliff. Automated creation of a high-performing algorithmic trader via deep learning on level-2 limit order book data. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1067–1074. IEEE, 2020.

[57] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

[58] 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, 2019.

Appendix A

Journal Paper

Toward Explainable AI Traders: Analysis of Algorithmic Traders Created via Deep Learning on Level-2 Limit Order Book Data

Aaron Wray, Owen Coyne, Matthew Meades, & Dave Cliff*
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.

***Abstract*— We show that a Deep Learning Neural Network (DLNN) learns to be a high-performing algorithmic trading system, operating purely from training-data inputs generated by passive observation of an existing successful trader $\mathcal{T}$. That is, we point our DLNN system at trader $\mathcal{T}$ and successfully have it learn from $\mathcal{T}$'s trading activity, such that it then trades at least as well as $\mathcal{T}$. Our system, *DeepTrader*, takes inputs derived from Level-2 market data, i.e. the market's *Limit Order Book* (LOB). 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 $\mathcal{S}$ of Level-2 LOB data taken when $\mathcal{T}$ issued a quote $\mathcal{Q}$ (i.e. a bid or an ask) to the market; and DeepTrader's desired output is to produce $\mathcal{Q}$ when it is shown $\mathcal{S}$. That is, we train our DLNN by showing it the LOB data $\mathcal{S}$ that $\mathcal{T}$ saw at the time when $\mathcal{T}$ issued quote $\mathcal{Q}$; and in doing so our system comes to behave like $\mathcal{T}$, acting as a profitable algorithmic trader. We evaluate DeepTrader against other algorithmic trading systems, including two that have repeatedly been shown to outperform human traders. DeepTrader matches or outperforms such pre-existing algorithmic trading systems. We analyze successful DeepTrader networks to identify what features are relied on and which features can be ignored. Our methods can in principle create an explainable copy of an arbitrary trader $\mathcal{T}$ via imitative deep learning methods.**

Keywords—automated trading, financial markets, deep learning, model analysis, continuous double auctions.

I. Introduction

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

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

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

In the language of research psychologists, our approach sketched in this story is a *behaviorist* one: we are concerned only with the "sensory inputs" and "motor outputs" of the 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. [17]) 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 (e.g. [61]), or even a MACD signal (e.g. [11]): 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. All we seek to do is *imitate* the original trader, the source of the training data.

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), a form of machine learning (ML) that has in recent years been demonstrated to be very powerful in a wide range of application domains (for further details, see e.g. [33]; [60]; [26]; and [71]). 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,

* Please direct all correspondence to Dave Cliff: csdtc@bristol.ac.uk; +44 79 77 55 22 50.

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 [52]) 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 for DeepTrader 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. a summary of 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 each such price: this is known in finance as *Level 2* LOB data: we discuss this in more detail in Section II. The DeepTrader output vector, the action to be associated with each input, could be an order (a fresh quote, or a cancellation of a previous order) issued from the trader to the exchange, and/or it could be a trade executing on the exchange.

More generally, as a source of input-data for DeepTrader, we need a market environment, which we'll denote by $\mathcal{M}$; and to generate the target outputs used in the training-set we need a training trader, which we'll denote by $\mathcal{T}$. 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 $\mathcal{M}$ – 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 $\mathcal{T}$, 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 (see [21], [23], [24], and [25]). These two "super-human" algos are known by the acronyms AA (for *Adaptive Aggressive*) and ZIP (for *Zero Intelligence Plus)*: full details of AA are given in [66] and [67], and of ZIP in [12]. Given that AA and ZIP 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 $\mathcal{T}$, then the likelihood is that it will also do very well when we deploy the same methods albeit using data from a human $\mathcal{T}$ – 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 $\mathcal{T}$ 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 $\mathcal{T}$ to produce target outputs, we also need an $\mathcal{M}$, 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 $\mathcal{M}$ in this study we use the long-established public-domain market-simulator BSE ([4], [13]) 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 $\mathcal{T}$ traders can be found alongside the source-code we used for our $\mathcal{M}$ market, in the BSE GitHub repository [4].

The rest of this paper, which is revised and expanded from [69], is structured as follows. In Section II we summarize the background to this work, and in Section III we explains the basics of the LOB, sufficient to understand the LOB-derived input features that DeepTrader relies upon. Section IV then describes our methods. In Section V we present results from [68] which demonstrate that DeepTrader learns to outperform pre-existing algo traders in BSE, and matches or exceeds the trading ability of the two "super-human" algorithms AA and ZIP. Section VI, drawn from [43] and [20], presents analyses of

trained DeepTrader networks created as independent replications of the results first published in [68], in which we demonstrate that the operation of successfully trained DeepTrader networks can be explained to a reasonable extent; we discuss our plans for further work in Section VII, and offer conclusions in Section VIII.

II. Background

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

In 1962, Vernon Smith published an article in the prestigious *Journal of Political Economy* (JPE) on the experimental study of competitive market behaviour [56]. The article outlined a number of laboratory-style market simulation experiments where human subjects were given the job of trading in a simple open-outcry CDA where an arbitrary asset was traded, while the experimenters looked on and took down their observations. The supply and demand curves used in these experiments were realistic, but were predetermined by Smith, who allocated each trader a private limit price: the price that a buyer cannot pay more than, or the price that a seller cannot sell below. Different buyers might be given different limit prices, and the array or schedule of limit prices would determine the shape of the demand curve in the experimental market; ditto for the schedule of sellers' limit prices and the resultant market supply curve. In this sense, Smith's experimental subjects were like sales traders in a brokerage or bank, running customer orders: some external factor sets a limit price, and the trader's job is to do their best to buy or sell within that limit. If a buyer can get a deal for less than her limit price, the difference is a saving; if a seller can get a deal for more than her limit price, that's profit. Economists use 'utility' or 'surplus' to refer to both the buyer's difference and the seller's difference, but as we're focused on applications in finance we'll use 'profit' for both.

The experiments run by Smith demonstrated a rapid convergence of a market to its theoretical equilibrium price (the price, denoted as P^*, the price at which 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 become *de facto* standards for test and comparison of trading algorithms. Further details of research in experimental economics are given in [35], [57], and [48].

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, as described in [53]. Thirty contestants competed for cash prize incentives totaling $10,000. The prize money won by each contestant was in proportion to the profit that their agent received in a series of different market environments. The highest ranked algorithm, designed by Todd Kaplan, was a simple agent that would hide in the background and hold off from posting a bid/ask price whilst letting other traders engage in bidding negotiations. Once the bid/ask spread was within an adequate range, Kaplan's agent would enter and "steal the deal". Aptly, Kaplan's program was named *Sniper*. If the market session was about to end, Sniper was programmed to rush to make a deal rather than not make one at all.

Subsequent to this, Gode & Sunder [30] published a JPE paper investigating the intelligence of automated traders and their efficacy within markets. 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 have been widely used as a *de facto* standard benchmark for a lower-bound on intelligence in automated traders, and models of markets populated by ZI traders have proven to be surprisingly informative (for reviews, see [27] and [37]).

Extending the work of Gode & Sunder, Cliff [12] 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 details of ZIP, the economists Gjerstadt & Dickhaut co-authored a paper [29] that approached the sales trader problem from a new perspective. 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 [21] at the prestigious *International Joint Conference on AI* (IJCAI) that generated worldwide media coverage: 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-management companies, and in following years the world of finance started to see ever increasing levels of automation, with more and more human traders replaced by machines.

Academic and industrial R&D continued after the landmark IBM study, and two significant subsequent developments were the extension of MGD into GDX, and a new ZIP-related trading algorithm called AA. Details of GDX were published by Tesauro & Bredin [62]: 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 [66] and subsequent article in the prestigious journal *Artificial Intelligence* [67]. The key element of AA is *aggressiveness*: a more aggressive trader places a bid/ask that is more likely to be accepted, while a less aggressive trader will aim to seek a larger gain. This trading strategy estimates the market's equilibrium price by using a weighted moving average and estimates the volatility of the market by using Smith's α metric.

Inspired by the IBM experiments pitting human traders against robot traders, a decade later in 2011 De Luca & Cliff ran a series of experiments, reported at IJCAI [23], 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 [58], inspired by [65], performed a brute-force exhaustive search of all possible ratios or permutations of different trading strategies for markets populated by a specific number of traders, consisting of over 1,000,000 market sessions, in order to show that AA doesn't always outperform GDX or ZIP: there are some circumstances in which AA can be dominated by GDX, or by ZIP: a topic further explored in [51] and [15].

This small set of zero-intelligence and minimal-intelligence trader-agent algorithms (in chronological order: ZIC, ZIP, GD, MGD, GDX, and AA) have been used in a large number of studies and the papers in which they were first described are highly cited.[1] And, over the past 30 years, the study of artificial markets populated by trader-agents, an approach known as *agent-based computational economics* has steadily grown as a third tool in the armoury of empirical economists: for further details see [64], [9], [10], and [34]. This complements the laboratory-based experimental economics approach as pioneered by Vernon Smith and his colleagues, and the approach known as *market microstructure*, in which fine-grained high-resolution data from real financial markets is analysed in forensic detail (see e.g. [46], [32], and [39])

While AA, GD, GDX, MGD, and ZIP were all early instances of AI trading for financial markets, 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 [33]) 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 [71] which uses a form of DLNN traditionally used in image processing, to capture the spatial structure of a LOB, coupled with an additional recurrent DLNN that incorporates information gathered over long periods of time. The second is by Le Calvez & Cliff [38], which demonstrated preliminary results from the use of a DLNN to successfully replicate the behavior of a ZIP trader, but which used only the best bid and ask prices (i.e., *not* Level 2 LOB data). As Sirignano & Cont [55] have recently and elegantly demonstrated, deeper (Level-2) LOB data can be highly informative about short-term market trends, so a natural question to explore given Sirignano and Cont's result is: can we extend the methods reported by Le Calvez & Cliff to instead use Level-2 LOB data? And, if that does result in a DLNN-trader that performs well, can we analyse it and explain its operation? These two questions are explored and answered in this paper.

Our methods are explained in Section IV, but before that Section III briefly explains the Limit Order Book in more detail, introducing various concepts that are important in understanding the input features that DeepTrader operates with.

[1] As of late March 2021, according to *Google Scholar* data, the six primary papers describing these six trader-algorithms had a total of 3452 citations: [30] on ZIC has 1995; [12] on ZIP has 431; [29] on GD has 534; [63] on MGD has 174; [62] on GDX has 150; and [67] on AA has 168.

III. The Limit Order Book (LOB)

Entire books have been written about LOBs (see, e.g. [47], [45], and [1]) but for the purposes of this paper we need only to introduce a few basic concepts and establish the necessary terminology for describing how DeepTrader works.

In financial markets that operate via a central electronic exchange, it is entirely commonplace for traders to interact with the exchange by sending orders of various types to the exchange, and (depending on the order-type) the exchange will then typically either execute the order immediately, or hold the order on its internal records for some period of time – either until a specified condition is met and the order is executed, or until some other condition is met (such as a time-limit being reached, or a cancellation-request coming from the trader that originated the order) and the order is deleted.

Any order sent by a trader to the exchange would usually specify a direction (i.e. *buy* or *sell*), a quantity (also known as the *size* or *volume*), and a price-per-unit. Buy orders are known as *bids*, and sell orders as *asks* or *offers*. Most contemporary exchanges allow traders to issue a fair number of different order-types (for an illustrative list, see Cliff, 2018b) but the two fundamental order types are *market orders* and *limit orders*.

Almost always, a market order (MO) executes immediately: an ask MO will be matched with the exchange's current best (highest) bid order, and the trader that issued the ask is then committed to buy from the trader who issued that best bid: the two traders have each found a counterparty, and the transaction goes through at the price of the best bid that was hit by the incoming ask MO – this is known as *hitting the bid*. Similarly, a bid MO will be matched against the exchange's current best (lowest) ask, a process known as *lifting the ask;* the two traders concerned are then matched as counterparties, and the transaction goes through at the price of the best ask that was lifted by the MO. The only times when a MO does not execute are when there are simply no potential counterparties to the trade: i.e., no best bid to hit and/or no best ask to lift. In most liquid real-world markets such situations occur extremely rarely, if ever.

In contrast, the usual expectation is that a limit order (LO) will not execute immediately. Instead, it will sit "on the books" of the exchange in the hope that it will eventually be matched (and, usually, the hope is that it will be matched *soon*), but at the point a LO is received at the exchange, it will usually be stored and held on the exchange's internal record, its "book", which has an obvious name: the limit order book (LOB). This is because a LO specifies a price that the trader wants to transact at, but that price is the wrong side of the best price currently being quoted by potential counterparties: that is, a bid LO price would usually be *less* than the current best ask price, and an ask LO price would usually be *more* than the current best bid price. Different traders will typically quote different prices and for their LOs, and so at any one time the exchange will be holding some number of bid LOs and some number of ask LOs: these are summarised in the LOB that is published by the exchange (usually updated in real-time, as each new order comes in). The LOB aggregates and anonymises the orders sat at the exchange, showing only the total quantity available for all LOs held at any specific price. Typically, the LOB is displayed in tabular format, with the summary of all current bid LOs in one vertical half of the table, and the summary of all current LOs in the other: both sides of the LOB are arranged from top-to-bottom in best-to-worst price order, so the lowest-priced ask and the best-priced bid are at the top of the book. All of this is illustrated in Figure 1.

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

Fig. 1. Typical tabular arrangement of a limit order book (LOB). All limit orders (LOs) currently within the exchange's matching engine are aggregated and summarised, with the identities of the traders contributing the orders not disclosed. Bid LOs are summarised on the left-hand side of this LOB, asks on the right. Prices are shown in the inner two columns, and the total aggregated quantity available at each price is show on the outer two columns. Both bids and asks are ordered from top to bottom as best-to-worst, so bid prices decrease as you move down the table, but ask-prices increase. The prices at the top of the table (highlighted by the red box) are the current best bid and ask; the difference between them is the *spread;* and the arithmetic mean of the two is the *mid-price*.

There is special terminology for various aspects of the LOB. The prices at the top of the LOB are the current *best bid* and *best ask*, and the difference between them is known as the *bid-ask spread*, often simply abbreviated to the *spread*. The arithmetic mean of the best bid and ask prices is the *midprice,* which is often used as a single-number summary of the market's current price situation. The mid-price pays no attention to the quantities available at the best bid and ask prices, but if there is a large difference between those two quantities (i.e., an *imbalance* between supply and demand) then the midprice can be a poor indicator of where actual transaction prices are likely to be in the immediate future, because if supply is a lot greater than demand then prices are likely to go down, while if demand is a greater than supply then prices are likely to rise. A different statistic, known as the *microprice*, uses the prices and the quantities of the best bid and ask to take account of any imbalance: any imbalance in quantities pushes the micro-price away from the mid-price and toward either the best bid or the best offer. We use the microprice as defined in [8].

Finally, for completeness, note that if a trader issues a LO to the exchange with a price that crosses the spread (i.e., a bid LO priced higher than the current best ask, or an ask LO priced lower than the current best bid) then that is interpreted and executed as if it was a market order: it is matched with the best counterparty order from the top of the other side of the LOB, and the transaction goes through.

The computational finance research literature exploring issues in LOBs and market trading dynamics is extensive, and it is beyond the scope of this paper to provide a literature survey here. Relevant notable papers include [2], [5], [31], [18], [19] and [70].

IV. Methods

Comparing the performance of trading strategies is not a straightforward task. As previously mentioned, the performance

of a strategy is reliant upon the other traders within the market and in real-world financial markets, it is implausible to know what algorithms other traders are using, as this information is confidential. Traders tend not to disclose their strategies in order to remain profitable, for obvious reasons. Nevertheless, there are well-established experiment-methods which can be used to compare trading agents. IBM's Tesauro and Das [63] present three separate experiment designs for comparing trading agents, two of which will be used here: in one-in-many tests (OMTs), one trader is using a different strategy to the all rest -- this test is used to explore a trading strategy's vulnerability to invasion and defection at the population level; and in balanced-group tests (BGTs) buyers and sellers are split evenly across two types of strategy, and for every trader using strategy A is matched with a trader using strategy B, with the matched-pair ech being given the same limit price. BGTs are generally considered to be the fairest way to directly compare two strategies.

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

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

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

In training, the DeepTrader DLNN takes as input 14 features, each 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 features are as follows (the +/– prefixes on input values are markers, explained in Section VI):

F1. – The time t the trade took place.

F2. + Flag: did this trade hit the LOB's best bid or lift the best ask?

F3. + The price of the customer order that triggered the quote that initiated this trade.

F4. + The LOB's bid-ask spread at time t.

F5. + The LOB's midprice at time t.

F6. + The LOB's microprice at time t.

F7. + The best (highest) bid-price on the LOB at time t.

F8. – The best (lowest) ask-price on the LOB at time t.

F9. – The time elapsed since the previous trade.

F10. – The LOB imbalance at time t.

F11. – The total quantity of all quotes on the LOB at time t.

F12. – An estimate P^* of the competitive equilibrium price at time t, using the method reported by Vytelingum *et al.* (2006, 2008).

F13. – Smith's (1962) α metric, calculated from P^* at time t.

F14. The price of the quote.

Features F1 – F13 are the inputs to the network: if any of them is undefined at time t then zero is used. F14 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 F1 – F13 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 [68].

The BSE GitHub repository [4] 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 "giveaway" trader that simply quotes at its own limit price, giving away all potential profit; and SHVR, a "shaver" trader whose strategy is simply always to undercut the current best ask by one penny, and/or to always beat the current best bid by one penny – this strategy is intended as a minimal model of a pesky high-frequency trader.

To create a large dataset to train the model, many market-session configurations were devised where the proportions and types of traders were varied. Each market session had 80 traders (40 buyers and 40 sellers). Additionally, each market session

involved four different trading strategies. For each trading strategy, the number of buyers and sellers was always the same but there were five different proportion-groups of traders used. These proportion-group were: (20, 10, 5, 5), (10, 10, 10, 10), (15, 10, 10, 5), (15, 15, 5, 5), and (25, 5, 5, 5). Each number within a group denotes the number of buyers and sellers for a specific trading strategy within a market session. For example, the (20, 10, 5, 5) proportion group, indicates that there were 20 buyers and sellers of trading strategy 1; 10 buyers and sellers of trading strategy 2; 5 buyers and sellers of trading strategy 3; and 5 buyers and sellers of trading strategy 4 within this group.

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

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

Each individual market session takes approximately 30 seconds to complete, so running all 39,200 on a single computer would take approximately 13.5 days. For this reason, the decision was made to use Amazon Web Services's *Elastic Compute Cloud* (AWS EC2) service to parallelize data generation and collection processes amongst 32 virtual machines (VMs). The Python library *Boto3* v.1.13.3 [28] was used to create, manage and terminate the VMs. Work was automatically split amongst the VMs by making every VM run each market configuration once and via a separate custom utility, created for this project.

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

The LSTM network created consists of three distinct hidden layers. The first hidden layer is an LSTM layer containing 10 neurons. The final two hidden layers are both fully connected layers containing 5 and 3 neurons respectively. Each hidden layer uses the Rectified Linear Unit (Relu) as an activation function: further details are given in [68].

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 [36] 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 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 [68]. An epoch in training is the network being presented with each data-point within the training dataset once. We trained DeepTrader's LSTM network for 20 epochs, and the error measure typically fell rapidly in the first 10 epochs and was thereafter asymptotic approaching a very low value for the remainder of the training process. So, in total, DeepTrader would be trained via exposure to LOB data from 20 x 39,200 = 784,000 individual market sessions, and each of those sessions would typically involve roughly 20 LOB snapshots, so the total number of snapshots used in training was around 15 million.

V. 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 2 and 3 show results from the balanced group tests (BGTs), while Figures 4 and 5 show results from the one-in-many tests (OMTs). As is described in detail in Chapter 4 of [68], 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 2a shows BGT comparison of APPT scores between DeepTrader and ZIP, and Fig. 1b shows BGT comparison of APPT scores between DeepTrader and AA. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that in this case there is no significant difference between ZIP and DeepTrader, but AA does significantly outperform DeepTrader. However, as we will see in Figure 4, when AA is pitted against DeepTrader in one-in-many tests, AA is outperformed by DeepTrader: we discuss these AA results later in this section.

Figure 3a shows BGT comparison of APPT scores between DeepTrader and GDX, and Figure 3b 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 4a shows OMT comparison of APPT scores between DeepTrader and ZIP, and Figure 4b shows OMT comparison of APPT scores between DeepTrader and AA. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that DeepTrader significantly outperforms both ZIP and AA, although from

visual inspection of the graphs it is also obvious that DeepTrader has much more variability of response than either ZIP or AA.

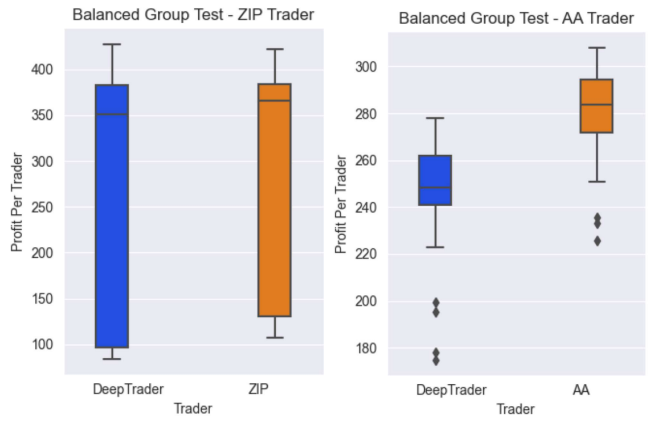

Fig. 2. Box-plots showing average profit per trader (APPT) from balanced-group tests (BGTs) for DeepTrader vs ZIP algorithmic traders (left: Fig2a) and AA algorithmic traders (right: Fig2b).

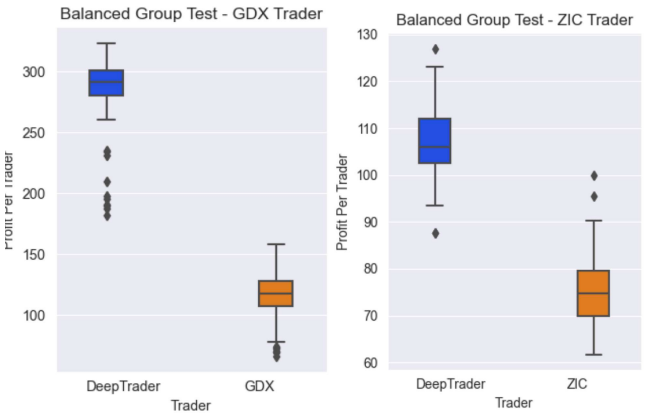

Fig. 3. Box-plots showing APPT from BGTs for DeepTrader vs GDX algorithmic traders (left: Fig.3a) and ZIC algorithmic traders (right: Fig.3b).

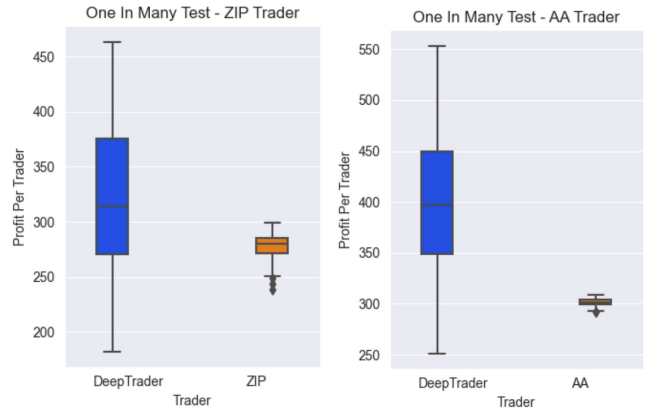

Fig. 4. Box-plots showing APPT from one-in-many tests (OMTs) for DeepTrader vs ZIP traders (left: Fig4a) and vs AA traders (right: Fig4b).

Figure 5a shows OMT comparison of APPT scores between DeepTrader and GDX, and Figure 5b shows OMT comparison of APPT scores between DeepTrader and ZIC. Comparison of 90% confidence intervals around the mean APPT scores for the two trading strategies in each graph indicates that DeepTrader significantly outperforms both GDX and ZIC, although again from visual inspection of the graphs it is also obvious that DeepTrader has much more variability of response than either GDX or SHVR.

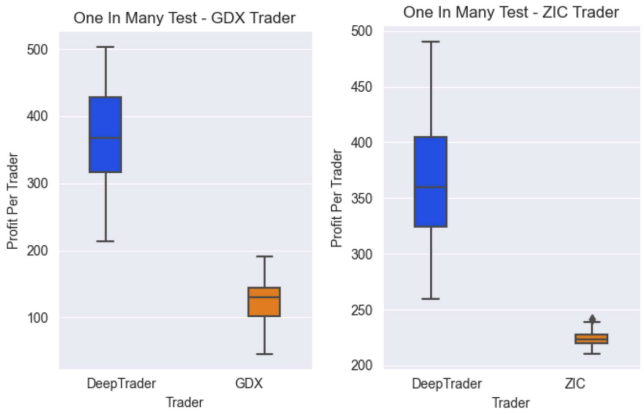

Fig. 5. Box-plots showing APPT from OMTs for DeepTrader vs GDX (left: Fig5a) and vs ZIC (right: Fig5b).

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 $\mathcal{T}$, where each order is associated with a snapshot of the Level2 market data available to $\mathcal{T}$ 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 $\mathcal{T}$ was operating in.

Given that DeepTrader does so well against the relatively advanced adaptive algorithms GDX and ZIP, it is perhaps no surprise to learn that it also does well against the other less sophisticated traders built into BSE, i.e.: SNPR, ZIC, GVWY, and SHVR. The results when pitted against ZIC were shown in Figures 3 and 5. For completeness, Figures 6 and 7 summarise the complete set of results: there is only one case (colored red) where DeepTrader is outperformed by one of BSE's built-in algorithms: the balanced-group test with AA. In two of the balanced group tests (with GVWY and with ZIP) no statistically significant difference was identified in the results; and in all other cases DeepTrader outperformed the built-in trader, although (as is starkly evident in Figures 4 and 5) the distribution of results from DeepTrader often has markedly more variance than that of its competitor.

	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

Fig. 6. Summary of the Balanced Group Tests (BGTs). In only one case is DeepTrader outperformed by its competitor (AA, coloured red); in two cases there was no statistically significant difference identified (GVWY and ZIP); and in the remaining four cases DeepTrader outpeformed its competitor.

	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

Fig. 7. Summary of the One-in-Many Tests (OMTs). In all cases DeepTrader signifcantly outperforms its competitor.

Most notably, we have shown here that DeepTrader learns to trade at least as well as ZIP, one of the two "super-human" algo traders for which code is already available in BSE. The results for DeepTrader when learning from (and then pitted against) the other super-human algo studied here, Vytelingum's AA, are somewhat less clear-cut: in the balanced group test, AA gets the better of DeepTrader; but in the one-in-many tests, DeepTrader roundly outperforms AA. As these two sets of tests result in a 1-1 tie, it seems fair to call it a draw.

So, in summary the results presented here (which are expanded upon in [68] and [43]) collectively show DeepTrader having six clear wins and one draw. While seven straight wins would naturally be preferable, these results nevertheless clearly demonstrate that the approach we have developed here has merit, and warrants further exploration.

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

VI. Analysis

Our initial analysis, explained in detail in [43] was an ablation study: that is, removing, disabling or masking specific aspects of the DeepTrader network and then noting the effects of that ablation. In general terms, we perform a sequence of ablation studies and the results from such a sequence can prompt us to hypothesise about how we could increase the efficiency of DeepTrader; we then test such hypotheses by studying the market performance of edited versions of DeepTrader, ones that have been altered to reflect our hypotheses – and this process can be iterated in an attempt at reducing the DeepTrader network to a minimally complex version that retains the ability to produce the desired level of trading behavior.

As is described in more detail in [43] we ran a set of 13 experiments where in each experiment one of the 13 input features was "ablated" by having the values in that column of the training data-set randomly shuffled (so the frequency-distribution of values in that column was unaltered, but any correlation between the value in that column and the other 12 features was very heavily disrupted); this gave us 13 sets of results where we could record the performance hit, the increase in error rate, when a specific one of the 13 features was ablated. A relatively large performance hit for a specific feature is an indication that DeepTrader's good trading behavior is indeed to reliant on that feature, whereas if the performance hit was sufficiently low then we considered that as an indication that the feature in question was, to a first approximation, irrelevant (or, at least, of sufficiently limited significance that it could be safely ignored) – that is, a low performance hit for ablating a feature led us to hypothesize that DeepTrader could operate successfully without access to that feature as an input; to check this, we re-ran our experiments with the feature (and its associated neurons within DeepTrader) absent, to check that the behavior of the resultant trader was consistent with our belief.

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

While the ablation study is an interesting first step, it is a relatively blunt instrument. In a subsequent independent replication and re-analysis of DeepTrader, reported in depth in [20], we employed a more mathematically sophisticated approach: Lundberg & Lee's [41] *SHapley Additive exPlanations* (SHAP) analysis.

To summarise briefly, SHAP analysis unifies several other mathematical analysis techniques aimed at helping to interpret or explain the otherwise opaque outputs of contemporary machine learning techniques such as the LSTM DLNNs used in DeepTrader. Lundberg & Lee's SHAP method is motivated by noting the underlying similarities between the analytic approaches LIME [50]; DeepLIFT [54]; layer-wise relevance propagation [3]; and classic game-theoretic methods such as Shapley regression values [40], Shapley sampling values [59] and Quantitative Input Influence [22], and their SHAP approach aggregates and integrates across such methods (for an introductory overview of these kind of interpretation-enabling approaches in machine learning, see [44]).

Shapley values are a measure of feature importance within a machine learning framework. They signify a measure of the distance between the predicted output of the network and the output if we didn't know each of the features. In this case it refers to the difference in price output caused by having each of the features supplied to the network, compared to the "baseline" price. This baseline price was considered here to be the price output when the network was applied to the mean of the

validation set (~$122.1632). Therefore, the Shapley value refers to the quantified increase or decrease in price from the baseline (caused by the according feature).

We applied the public-domain *gradientSHAP* [6] to the full validation set of our network (1,896,262 trades) to get a distribution of Shapley values across the data. This distribution shows the magnitude of effects the respective features had on the final prices quoted by DeepTrader.

The violin-plot shown in Figure 8 (where the 'sides' of the violin are a representation of the kernel density plot) shows that for many of the features they rarely had any large effect on the final price with many of their Shapley values clustered tightly around zero. Those features with nonzero gradientSHAP values reinforce the finding of the ablation study, i.e. that features F2, F5, F6, and F7 do have significant effect on model output, but this gradientSHAP plot also reveals that F8 and F12 play larger roles in the final price output than could be ascertained from our ablation study.

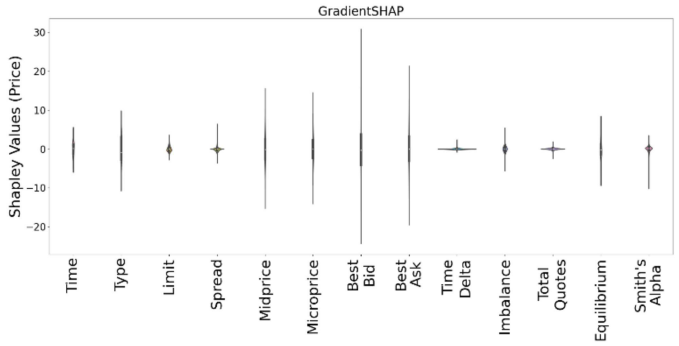

Fig. 8. Violin-plot of results from gradientSHAP analysis of a trained DeepTrader DLNN. The 13 input features F1-F13 for the DeepTrader LSTM DLNN are arranged along the horizontal axis in order: *Time; Type; Limit; Spread; Midprice; Microprice; BestBid; BestAsk; TimeDelta; Imbalance; TotalQuotes; Equilibrium*; and *Smith's α*. Vertical axis is gradientSHAP value: see text for discussion.

In an additional refinement, the public-domain *kernelShap* method [7] allows us to take an individual sample from the data and investigate its Shapley values specifically (as opposed to the global estimation method of gradientSHAP). It does this by training a linear model to approximate the network at that specific instance, using the LIME framework [50], and as such its output gives a good explanation of any particular trained instance that we would like to explore: an example is given in tabular form in Table 1.

We can see the impact of each feature when calculating this specific single output price but also can consider that the sum of the Shapley values overall approximates the difference between the output and baseline (outputPrice − baselinePrice), i.e. ~$105.85 − ~$122.16 = −$16.31.

This analysis helps us understand the impact of each of the features within the model and thus how DeepTrader parses the structure of the LOB through them. Going further, when reintroduced back into DeepTrader it allows each quote-price the network makes to provide Shapley Values which explain to some degree the decision process behind the price being quoted. At least in part this brings back a level of understanding and trust that had been sacrificed for the trading performance provided by the nature of the black-box, deep learning methods employed. As such we can see that there is a line of research to be explored for future work that exploits super-human trading performance whilst maintaining a reasonable level of interpretability by humans.

Table 1: Output from kernelSHAP analysis: see text for discussion.

	Value	Shapley Value (Price)
Time	257.754120	-0.7656
Type	0	+2.2098
Limit	78.004800	-0.5394
Spread	35.532800	+1.4442
Midprice	95.758250	-4.2804
Microprice	95.758250	-3.6018
Best Bid	113.539200	-3.1668
Best Ask	77.994800	-7.2906
Time Delta	0.637767	-0.0696
Imbalance	-0.130405	+0.5742
Total Quotes	22.999400	-0.3654
Equilibrium	118.210748	-0.1914
Smith's α	0.093177	-0.6612
Price	105.845800	
Column Sum		-16.7040

VII. Further Work

Although at the start of this paper we characterized our approach to the use of learning in DeepTrader as *behaviorist*, because we concentrate only on the observable inputs and resultant trading behavior of the system, and although this approach was demonstrated by the results presented here to be one that delivers successful outcomes, we do not intend to always treat DeepTrader as an impenetrable black box system. Instead, our next phase of work will be devoted to further analysing the internal mechanisms that make a trained DeepTrader so successful. In particular, we will investigate the extent to which each of the key inputs identified in Section V 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 most significant varies across all market conditions: we will report on our findings in this respect in a future publication; there is much to explore.

We have deliberately commenced our work by training DeepTrader with data that comes from relatively simple and unsophisticated adaptive automated trading systems such as AA and ZIP, and so a natural next step is to repeat these types of experiments but work with training data that comes from more sophisticated traders working in more realistic market environments. We are currently exploring how the methods reported here perform when the underlying market equilibrium price P^* follows a plausibly realistic random-walk path, by using a geometric Brownian motion process (see e.g. [42], [49]); and when the traders in the market, generating the trading data, can smoothly vary their responses to market conditions (e.g, in

the manner recently introduced with the PRZI trading algorithm [16]) and that are pre-emptively responsive to temporary imbalances between the levels of supply and demand in the market (e.g. in the manner described in [72]).

Finally, given the story that we opened this paper with, an obvious future step is to attempt to repeat the methods used here, with the training data coming from human traders rather than from trading agents.

VIII. Discussion and Conclusions

We have explored here the problem of using machine learning to automatically create high-performance algorithmic traders that are fit to operate profitably in a contemporary financial exchange based (as are all current major electronic exchanges) on a CDA process mediated by a continuously updated limit order book showing Level 2 market data. Our approach to this problem is imitative and 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.

Despite that pre-commitment to treating the DeepTrader learnt DLNN model as a black-box, we have also demonstrated here that an *a posteriori* analysis of the trained DLNN model can reveal explanatory insights about how the network functions, what inputs it is attending to and which are less important or ignored. We have made the Python source-code for the initial implementation of DeepTrader as described in [68] freely available as an open-source release on GitHub[2], and we have similarly made an open-source release of the Python source-code of Meades' independent replication of Wray's results, as described in [43], also on GitHub[3], for any readers who wish to explore or extend the work reported here.

The novel results presented in this paper have demonstrated our imitative 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.

[2] https://github.com/wray27/DeepTrader

[3] https://github.com/Matt0912/DeepTrader2

References

[1] F. Abergel, M. Anane, A. Chakraboti, A. Jedidi, and I. Toke, *Limit Order Books*. Cambridge University Press, 2016.

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

[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Muller, and W. Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation" PLoS ONE 10(7): e0130140. https://doi.org/10.1371/journal.pone.0130140, 2015.

[4] BSE: A LOB-Based Financial Exchange Simulator. Github open-source repository (code & documentation) https://bit.ly/2XdW184, October 2012.

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

[6] Captum, *GradientShap* Python: https://captum.ai/api/gradient_shap.html, 2021.

[7] Captum, *KernelShap* in Python: https://captum.ai/api/kernel_shap.html 2021.

[8] A. Cartea, S. Jaimungal, and J. Penalva. *Algrothimic and High-Frequecny Trading.* Cambridge University Press, 2015.

[9] S. H. Chen, "Varieties of agents in agent-based computational economics: A historical and an interdisciplinary perspective," *Journal of Economic Dynamics and Control,* 2011.

[10] S. H. Chen, *Agent-based computational economics: How the idea originated and where it is going*. Routledge, 2018.

[11] T. Chong, W. Ng, and V. Liew. Revisiting the performance of MACD and RSI Oscillators. *Journal of Risk and Financial Management*, 7:1-12.

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

[13] D. Cliff. "BSE: A Minimal Simulation of a Limit-Order-Book Stock Exchange". In: M. Affenzeller, A. Bruzzone, E. Jimenez, F. Longo, Y. Merkuryev, and M. Piera (eds) *Proceedings of the European Modelling and Simulation Symposium (EMSS2018)*, pp.194-203.

[14] D. Cliff. "An Open-Source Limit-Order-Book Exchange for Teaching and Research". In *Proceedings of the IEEE Symposium Series on Computational Intelligence SSCI2018: Computational Intelligence in Financial Engineering (CIFEr)*. SS-1296, pp.1853—1860, 2018b.

[15] D. Cliff and M. Rollins, "Methods matter: A trading algorithm with no intelligence routinely outperforms AI-based traders," in *Proceedings of IEEE Symposium on Computational Intelligence in Financial Engineering (CIFEr2020)*, 2020.

[16] D. Cliff. *Technical Note: Parameterized-Response Zero Intelligence (PRZI) Traders.* Manuscript on Arxiv: http://arxiv.org/abs/2103.11341, March 2021.

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

[18] R. Cont, A. Kukanov, and S. Stoikov. The price impact of order book events. *Journal of Financial Econometrics*, 12(1):47–88, 2014.

[19] R. Cont and A. Kukanov. Optimal order placement in limit order markets, *Quantitative Finance*, 17(1):21-39, 2017.

[20] O. Coyne. *An Exploration into Generalisable Deep Learning Trading Agents*. Masters thesis, Department of Computer Science, University of Bristol, forthcoming May 2021.

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

[22] A. Datta, S. Sen, and Y. Zick. "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems". In: Proc. 2016 IEEE Symposium on Security and Privacy, pp.598–617, 2016.

[23] M. De Luca and D. Cliff, "Agent-human interactions in the continuous double auction, redux: Using the OpEx lab-in-a-box to explore ZIP and

GDX," in *Proceedings of the 2011 International Conference on Agents and Artificial Intelligence (ICAART2011)*, 2011.

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

[25] M. De Luca, C. Szostek, J. Cartlidge, and D. Cliff, *Studies of interaction between human traders and algorithmic trading systems*. UK Government Office for Science, London, Tech. Rep., Sep. 2011.

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

[27] J. D. Farmer, P. Patelli, and I. Zovko, "The Predictive Power of Zero Intelligence in Financial Markets," *Proceedings of the National Academy of Sciences*, vol. 102, no. 6, pp. 2254–2259, 2005.

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

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

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

[31] M. Gould, M. Porter., S. Williams, M. McDonald, D. Fenn, and S. Howison,. "Limit Order Books". *Quantitative Finance,* 13(11):1709-1742, 2013.

[32] L. Harris, *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2002.

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

[34] C. Hommes and B. LeBaron, editors, *Computational Economics: Heterogeneous Agent Modeling*. North-Holland, 2018.

[35] J. Kagel and J. Roth, *The Handbook of Experimental Economics.* Princeton University Press, 1997.

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

[37] D. Ladley, "Zero Intelligence in Economics and Finance," *The Knowledge Engineering Review*, 27(2): 273–286, 2012.

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

[39] C.-A. Lehalle and S. Laruelle, *Market Microstructure In Practice (Second Edition)*. World Scientific, 2018.

[40] S. Lipovetsky and M. Conklin. "Analysis of regression in game theory approach". Applied Stochastic Models in Business and Industry 17(4):319–330, 2001.

[41] S. Lundberg and S.-I. Lee. "A Unified Approach to Interpreting Model Predictions" in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS2017)*. 2017.

[42] R. Marathe and S. Ryan. On the validity of the geometric brownian motion assumption. *The Engineering Economist,* 2005.

[43] M. Meades, *An Investigation into the Success of Deep Learning Trading Agents*. MSc Thesis, September 2020.

[44] C. Molnar, *Interpretable Machine Learning: A guide for making black-box models interpretable.* Lulu.com, 2019.

[45] I. Nolte, M. Salmon, and C. Adcock, Eds., *High Frequency Trading and Limit Order Book Dynamics*. Routledge, 2014.

[46] M. O'Hara, *Market Microstructure Theory*. Wiley, 1998.

[47] J. Osterrieder, *Arbitrage, Market Microstructure, and the Limit Order Book*. Suedwestdeutscher Verlag fuer Hochschulschriften, 2009.

[48] C. Plott and V. Smith, Eds., *Handbook of Experimental Economics Results, Volume 1*. North-Holland, 2008.

[49] K. Reddy and V. Clinton. Simulating Stock Prices Using Geometric Brownian Motion: Evidence from Australian Companies, *Australasian Accounting, Business and Finance Journal*, 10(3):23-47, 2016.

[50] M. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you? Explaining the predictions of any classifier". In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp.1135–1144, 2016.

[51] M. Rollins and D. Cliff, "Which trading agent is best? using a threaded parallel simulation of a financial market changes the pecking-order," in *Proceedings of the 32nd European Modeling and Simulation Symposium (EMSS2020)*, 2020.

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

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

[54] A. Shrikamur, P. Greenside, and A. Kundaje, "Learning Important Features through Propagating Activation Differences" in *Proc. 34th International Conf. Machine Learning (ICML2017)*, pp.3145-3153, 2017.

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

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

[57] V. Smith, editor, *Bargaining and Market Behavior: Essays in Experimental Economics*. Cambridge University Press, 2000.

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

[59] E. Štrumbelj & I. Kononenko, "Explaining prediction models and individual predictions with feature contributions". *Knowledge and Information Systems,* 41(3):647–665, 2014.

[60] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection". In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, 2:2553–2561. Curran Associates, Inc., 2013.

[61] T. Terasvirta, *An Introduction to univariate GARCH models.* SSE/EFI Working Paper Series in Economics and Finance, No.646. Stockholm School of Economics, the Economic Research Institute, 2006.

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

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

[64] L. Tesfatsion and K. Judd, editors, *Handbook of Computational Economics Vol.2: Agent-Based Computational Economics*. North-Holland, 2006.

[65] D. Vach, *Comparison of double auction bidding strategies for automated trading agents*, Master's thesis, Charles University, Prague, 2015.

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

[67] P. Vytelingum, D. Cliff, and N. Jennings, "Strategic bidding in Continuous Double Auctions". *Artificial Intelligence*, 172(14):1700-1729, 2008.

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

[69] A. Wray, M. Meades, and D. Cliff "Automatced Creation of a High-Performing Algorithmic Trader via Deep Learning on Level-2 Limit Order Book Data", in *Proceedings of the IEEE Symposium on Computational Intelligence for Financial Engineering (CIFEr)*, December 2020.

[70] K. Xu, M. Gould, and S. Howison. "Multi-Level Order-Flow Imbalance in a Limit Order Book". *Market Microstructure and Liquidity 4(3&4), 1950011*. 2018.

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

[72] Z. Zhang and D. Cliff, "Market impact in trader-agents: Adding multi-level order-flow imbalance-sensitivity to automated trading systems," in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART2021)*, 2021.