Deep Learning of Algorithmic Trading Strategies & Limit Order Book Dynamics

By

Justin Bercich

This thesis is submitted for the degree of

Doctor of Philosophy

Macquarie Graduate School of Management

Supervisor: Andrew Lepone

March 2018

Summary

Deep Learning of Algorithmic Trading Strategies & Limit Order Book Dynamics

Justin Bercich

Doctor of Philosophy

Macquarie Graduate School of Management

March 2018

The transcendental advancements over the past decade in automated trading systems and theoretical models of deep learning have led to the rise of machines as the primordial arbiter of decision-making in financial markets. Consequently, markets organised around limit order books (LOB) have come to be dominated by algorithmic trading strategies based on objective rules derived from the market microstructure. Traders are increasingly capable of wielding deep learning tools to extract complex abstract hierarchical representations from the plethora of high-frequency financial data available. Despite these manifest changes to the trader ecosystem driven by significant investment in automated technology and algorithms, there remain only fractured frameworks through which regulators, practitioners and academics view this new complex and intricate trader environment. The objective of this thesis is to develop deep learning models capable of defining algorithmic traders in this new environment by the type of *strategy* the firm is conducting, analysing the *impact* of these strategies on UK equity market quality and modelling the *nexus* between algorithmic trading strategies and LOB dynamics.

In the first part of this thesis, a series of neurologically-motivated Deep Neural Network (DNN) models are trained on a proprietary UK FTSE 100 dataset to capture the relationship between algorithmic traders' behaviour and the state of the LOB. An optimised model is derived through a comprehensive ablation analysis that evaluates the efficacy of the multitude of architecture, design component and configurations applicable when modelling the relationship between LOB features and trader actions. The optimised models are trained to accurately predict 96.4% of the actions executed by traders for the securities analysed, attaining superior results to comparative machine learning which highlights the DNN model's ability to extract complex hierarchical and abstract feature representations from the data.

In the second part of this thesis, a new framework for segmenting algorithmic traders by the specific *strategy* being executed is developed before analysing the positive and negative *impact* of these strategies on UK equity market quality. Axiomatic backpropagation-based feature attribution methods are employed to extract the relative saliency of each feature of the DNN models, which proxies for the algorithmic inputs into each trader's strategy. Unsupervised Spherical K-Means algorithms are then applied to infer structural patterns in the feature saliency datasets allowing each trader to be segmented by the specific algorithmic trading strategy they are conducting on the LOB. The resulting clustering of

firms into one of five interpretable and distinct strategies – Automated Market Makers, Execution, Microstructural, Momentum and Technical strategies – allows for an empirical examination of the aggregate impact of these strategies on market quality along dimensions of liquidity, volatility and price discovery.

In the third part of this thesis, we develop a deep learning model that utilises Deep Recurrent Neural Networks (RNN) and Deep Recurrent Q-Networks (DRQN) to predict the future non-linear price dynamics of the LOB. The LOB is modelled as a complex dynamic and adaptive system populated by heterogeneous traders conducting disparate trading strategies. We extract hierarchical features by building abstract temporal and spatial representations of the LOB state at any point in event-time, using a recurrent memory component to intrinsically incorporate dependencies from past states. An optimal RNN model is first developed through an ablation analysis that evaluates the ability of various architectural, design and network optimisation components to improve the generalisation power of the RNN. Various prediction experiments are then conducted which indicate that both the RNN and DRQN perform significantly better than a random naïve model at predicting the future dynamics of the LOB.

Declaration of Authorship

I, Justin Bercich, declare that the contents of this thesis, "Deep Learning of Algorithmic Trading Strategies and Limit Order Book Dynamics", are original and the work of my own research except where specific reference is made to the work of others. I hereby confirm to the best of my knowledge and belief that:

- The contents of this thesis have not been submitted in whole or part for consideration of any other degree in this, or any other university.
- This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, unless identified in the text and Acknowledgements.
- This thesis has been completed during the candidature for a PhD degree at this university.
- The work of any others has been quoted and sourced.

Signed,

Justin Bercich

March 2018

Acknowledgements

Firstly, I would like to thank the Macquarie Graduate School of Management, the University of Sydney, the Capital Markets Cooperative Research Centre (CMCRC), CASS Business School, the Financial Conduct Authority UK (FCA) and the Australian Government for the various scholarships, funding and partnership opportunities that have made this thesis possible over the past 3 years.

Secondly, I am deeply grateful to my supervisor Professor Andrew Lepone for the time, intellectual support and advice he provided throughout the research process. Furthermore, my gratitude is extended to Professor Mike Aitken and Professor Alex Frino from the CMCRC, Edward Schooling-Latter, Toby Wallis and Helen Boyd from the FCA, Professor Ian Marsh from CASS and many other colleagues across various institutions that have provided research opportunities and advice. To my friends in Australia, London and Europe, thank you for your stoicism, support and encouragement!

Most importantly, I would like to dedicate this thesis to my fiancé, Theresa, my parents, Giulio and Mira Bercich, and my siblings, Julian, Ashleigh and Joshua. To my mum, thank you for your support, patience, valued-advice and loving support. To my siblings, your unending kinship is a source of great happiness and always helped in challenging times. I would like to give my deepest gratitude to my father who passed away during the writing of my thesis for teaching my siblings and I the values of decency, family, dedication and ambition that I aspire to. No words can express the impact he has had on my life and my only wish is that he could be here today.

Finally, to my dearest fiancé, Theresa. With your love and dedication, anything is possible. I'd like to thank you for your passion, intelligence and care over the past 3 years. You are the source of my inspiration and a true role model.

For Theresa & my Family

Contents

SUMMARY ..... **I**

DECLARATION OF AUTHORSHIP ..... **III**

ACKNOWLEDGEMENTS ..... **IV**

CONTENTS ..... **VI**

1 INTRODUCTION **1**

1.1 Limit Order Books & Algorithmic Trading 2
1.2 Deep Learning 4
1.3 Motivation, Research Objectives and Contributions 6
1.4 Outline of Thesis 9

2 MARKET MICROSTRUCTURE & ALGORITHMIC TRADING **10**

2.1 Microstructure of UK Equity Markets 10
2.1.1 Trading Venues *12*
2.1.2 Order Types *14*
2.2 Limit Order Books 16
2.3 Algorithmic Trading Strategies 19
2.3.1 Automated Market Making *21*
2.3.1.1 Naïve Models 23
2.3.1.2 Inventory-driven Models 24
2.3.1.3 Information-driven Models 25
2.3.2 Microstructural Strategies *29*
2.3.2.1 Order Flow Autocorrelation 29
2.3.2.2 Aggressiveness Indicators 30
2.3.2.3 Market Depth 30
2.3.2.4 Limit Order Imbalance 31
2.3.2.5 Spreads 32
2.3.2.6 Volatility 33
2.3.3 Momentum Strategies *34*
2.3.4 Technical Strategies *35*
2.3.4.1 Bollinger Bands 35
2.3.4.2 Moving Average Convergence-Divergence 36
2.3.4.3 Relative Strength Indicators 36
2.3.4.4 Stochastic Oscillators 36
2.3.4.5 Chaikin Volatility 37
2.3.5 Execution Strategies *37*
2.3.5.1 Participation Rate Algorithms 40
2.3.5.2 Implementation Shortfall, Market Impact & Market Resiliency 42
2.3.6 Other Strategies *52*
2.4 Feature Engineering 53
2.4.1 Feature Scaling *53*
2.4.2 Feature Selection & Extraction *55*

2.4.2.1 Filter Models 56
2.4.2.2 Wrapper Models 56
2.4.2.3 Embedded Models 57
2.4.2.4 Feature Extraction & Dimensionality Reduction 57
2.5 SUMMARY 57

3 MODELLING ALGORITHMIC TRADING USING DEEP NEURAL NETWORKS 59

3.1 DEEP NEURAL NETWORKS 60
3.1.1 Neurological Framework 61
3.1.2 Generalisation & Stability 62
3.1.3 Feedforward Neuronal Architecture 63
3.1.4 Hierarchical Feature Representations 66
3.1.5 Activation Functions 67
3.1.5.1 Sigmoid Functions 69
3.1.5.2 Rectifier Functions 70
3.1.5.3 Maxout 72
3.1.5.4 Softmax 73
3.1.6 Cost Functions 73
3.1.6.1 Mean Error 74
3.1.6.2 Cross-Entropy & Kullback-Leibler 75
3.1.6.3 Log Hyperbolic Cosine 75
3.1.6.4 Exponential Loss 76
3.1.6.5 Hinge Loss 76
3.1.7 Weight Initialisation 76
3.1.7.1 Xavier Initialisation 77
3.1.7.2 He Initialisation 78
3.1.7.3 Orthogonal Matrix Initialisation 78
3.1.7.4 Layer-Sequential Uniform Variance Initialisation 78
3.1.8 First-Order Optimisation Algorithms 79
3.1.8.1 Common Issues with Gradient Descent 80
3.1.8.2 Stochastic Gradient Descent 80
3.1.8.3 Momentum 84
3.1.8.4 Nesterov Accelerated Gradient 84
3.1.8.5 AdaGrad 85
3.1.8.6 AdaDelta 85
3.1.8.7 Adaptive Moment Estimation 86
3.1.8.8 AdaMax 87
3.1.8.9 Nesterov Adaptive Moment Estimation 87
3.1.9 Regularisation 88
3.1.9.1 Weight Decay L2 Regularisation 88
3.1.9.2 LASSO L1 Regularisation 89
3.1.9.3 Dropout 90
3.1.9.4 Early Stopping 92
3.1.10 Unsupervised Pre-Training with Denoising Auto-Encoders 93
3.1.11 Batch Normalisation 94
3.2 COMPARATIVE MACHINE LEARNING MODELS 96
3.2.1 Generalised Linear Models 96
3.2.1.1 Multinomial Logistic Regression 97
3.2.2 Classification and Regression Trees 98
3.2.3 Random Forests 99
3.2.3.1 Ensemble Methods 99

3.2.3.2 Random Subspace Method 100
3.2.3.3 Bootstrap Aggregation 100
3.2.4 Support Vector Machines *101*
3.2.4.1 Soft-Margin SVM 101
3.2.4.2 Kernel Functions & Non-Linear SVM 102
3.2.5 Naïve Bayes Models *103*
3.3 Data 104
3.3.1 Limit Order Book Features & Actions *107*
3.3.2 ISIN-Date-Broker Tuple Datasets *109*
3.4 Baseline Model Experimental Evaluation 109
3.4.1 Rectifier vs Sigmoidal Activated Neurons *111*
3.4.1.1 Activation Function Hyper-Parameter Search 114
3.4.2 Sparse Representations & Bias Shift Correction *115*
3.4.2.1 Bias Shift Correction Using Exponential Linear Unit Activations 116
3.4.2.2 Rectifier Functions Induce Sparse Representations 118
3.4.3 Bias Node Irrelevance *121*
3.4.4 Initialisation Strategies *122*
3.4.5 Cost Function Evaluation *124*
3.4.6 Optimisation Algorithm Evaluation *127*
3.4.6.1 Learning Annealing 132
3.4.6.2 Bias Correction 133
3.4.6.3 Learning Rate Decay 135
3.4.7 Network Depth & Dimensionality *136*
3.4.8 Batch Size Evaluation *140*
3.4.9 Regularisation Evaluation *143*
3.4.9.1 Dropout Regularised Networks 143
3.4.9.2 L2 & L1 Norm Regularisation 146
3.4.10 Batch Normalisation Evaluation *147*
3.5 Optimised Model & Evaluation 149
3.5.1 Optimised Deep Neural Network Evaluation *150*
3.5.2 Confusion Matrix & Quality Metrics *153*
3.5.3 Internal Synaptic Weights Analysis *156*
3.5.4 Robustness Against Adversarial & Random Perturbations *157*
3.5.5 Sample Size vs Accuracy *159*
3.6 Comparative Machine Learning Model Evaluation 159
3.6.1 Generalisation & Predictive Power *160*
3.6.2 Confusion Analysis *162*
3.6.2.1 Confusion Matrix 163
3.6.2.2 Quality Metrics 163
3.6.2.3 Confusability Analysis 166
3.7 Conclusion 167

4 ALGORITHMIC TRADING STRATEGY IDENTIFICATION & MARKET QUALITY IMPACT USING UNSUPERVISED LEARNING & AXIOMATIC FEATURE ATTRIBUTION 169

4.1 Axiomatic Attribution for Limit Order Book Features 171
4.1.1 Feature Grouping *172*
4.1.2 Distance Correlation & Mutual Information Methods *174*
4.1.3 Synaptic Weight-based Methods *177*
4.1.4 Input Perturbation Methods *178*
4.1.4.1 Occlusion Analysis 178
4.1.4.2 Locally Interpretable Model-Agnostic Explanations 180

4.1.5 Backpropagation Approaches *182*
4.1.5.1 Gradient*Input 183
4.1.5.2 Layer-Wise Relevance Propagation 184
4.1.5.3 Deep Taylor Decompositions 185
4.1.5.4 DeepLIFT 186
4.1.5.5 Integrated Gradients 188
4.2 Backpropagation-based Feature Attribution Evaluation & Dataset Formulation 190
4.2.1 Saliency Distributions *190*
4.2.2 Delta Log-Odds *193*
4.2.3 Sensitivity-n *195*
4.2.4 Feature Attribution Dataset *197*
4.3 Algorithmic Trading Strategy Identification using Unsupervised Machine Learning 198
4.3.1 K-Means Models *198*
4.3.1.1 Sparse K-Means 199
4.3.1.2 Spherical K-Means 200
4.3.1.3 Hybrid Hierarchical K-Means 200
4.3.2 Model Evaluation & Clustering Properties *201*
4.3.2.1 Clustering Tendency 201
4.3.2.2 Optimal Number of Clusters 203
4.3.2.3 Cluster Validation for K-Means Partitioning 205
4.3.3 Algorithmic Trading Strategy Identification *208*
4.3.3.1 $\mathcal{FI}$ Structural Matrix 208
4.3.3.2 Dimensionality Reduction & Cluster Visualisation 212
4.3.3.3 Robust Analysis of Strategy Clustering 214
4.4 Impact of Algorithmic Trading on Market Quality 216
4.4.1 Liquidity *218*
4.4.1.1 Liquidity Supplying Characteristics of Algorithmic Trading Strategies 219
4.4.2 Volatility *225*
4.4.2.1 Contemporaneous Relationship 227
4.4.2.2 Dynamic Relationship 229
4.4.2.3 Causal Relationship 230
4.4.2.4 Fundamental Relationship 232
4.4.3 Price Discovery & Efficiency *233*
4.4.3.1 Impulse Response Functions & Market Impact 234
4.4.3.2 Information Share Analysis 237
4.4.3.3 Dynamic Linear Model Analysis 240
4.5 Conclusion 242

5 PREDICTING LIMIT ORDER BOOK DYNAMICS USING DEEP RECURRENT REINFORCEMENT LEARNING 246

5.1 Literature Review on Modelling Limit Order Books 248
5.1.1 Dynamic Equilibrium Models *249*
5.1.2 Stochastic Models *252*
5.1.3 Machine-Learning Models *256*
5.2 Deep Recurrent Neural Networks 257
5.2.1 Neurophysiological Conceptions of Memory *258*
5.2.2 Dynamical Systems *260*
5.2.3 Temporal Dependencies *260*
5.2.4 Deep Network Architectures *261*
5.2.4.1 Elman Recurrent Neural Network 263
5.2.4.2 Long-Short Term Memory 265

5.2.4.3 Gated Recurrent Units 269
5.2.5 Network Optimisation *272*
5.2.5.1 Adaptive Moment Estimation Algorithms 273
5.2.5.2 Backpropagation Through Time 273
5.2.6 Layer Normalisation *275*
5.2.7 Recurrent Dropout *276*
5.2.8 Self-Normalising Neural Networks *277*
5.3 Deep Reinforcement Learning 278
5.3.1 Markov Decision Process & Dynamic Programming *278*
5.3.2 Model-Free Reinforcement Learning *280*
5.3.3 Deep Recurrent Q-Networks *281*
5.4 Data 283
5.4.1 Features *284*
5.5 Model Design & Evaluation 287
5.5.1 Architecture Evaluation *288*
5.5.2 Depth & Dimensionality Evaluation *294*
5.5.3 Network Optimisation Evaluation *296*
5.6 Optimised Recurrent Neural Network 300
5.7 Experimental Results 301
5.7.1 Predicting Trader Actions at or Within the BBO *302*
5.7.2 Predicting Midpoint Price Changes *304*
5.7.3 Predicting Future Prices *305*
5.8 Conclusion 308

6 CONCLUSION 311

APPENDIX A – LIMIT ORDER BOOK & TRADER-SPECIFIC FEATURES 314

BIBLIOGRAPHY 320

Chapter One

1 Introduction

The past decade has seen transcendental advancements in technology driving transformations in financial market conventions and automated trading systems through which market participants execute algorithmic trading strategies. This continuum of evolution in financial markets is characterised by the ascendency of algorithms over humans as the dominant investment decision-making force, the spatial contraction of markets onto electronic limit order book (LOB) platforms, the temporal reduction of the investment process to the millisecond level and the exponential decay in latency rates. The progression towards advanced automated systems has been driven by the competitive forces in financial trading that place speed as the predominant determinant of trading strategies' profitability. Contemporary technological iterations have also facilitated the rise of machine and deep learning models in computational finance. Financial practitioners have the capacity to execute algorithmic trading strategies on LOBs using advanced models capable of learning complex representations with deep levels of abstraction from the plethora of financial data available. Consequently, there is an increasing need for researchers and regulators to deploy deep learning models that manifestly understand the complex intricacies and interdependencies of financial markets.

The objective of this thesis is to develop a deep and machine learning framework to interpret the dynamical nexus between algorithmic trading strategies and LOBs. To attain this objective, we develop deep learning models to define algorithmic traders by the type of *strategy* the firm is conducting, analysing the *impact* of these strategies on UK equity market quality and considering how algorithmic trading *influences* LOB dynamics. The question of what impact algorithmic trading has on LOB-driven equity markets has profound implications for capital formation in the real economy, requiring careful consideration by regulators tasked with the remit of supervising markets. Decisions regarding the allocation of finite financial resources, risk management and the formation of investment and trading decisions are increasingly made at the micro-level by automated trading systems and algorithmic trading strategies infused with machine intelligent capabilities. One caveat to consider is that whilst the outsourcing of decisions has led to the ascendency of machines in the financial system using manifold algorithms, the nature of how machines learn and act does not diverge significantly from humans. That is, all humans are faced with an environment beyond their comprehension, there are almost an infinite series of situations one can find themselves in and decisions that can be made, all played out in real-time, yet humans form intelligence as they learn to acquire and utilise knowledge. The crux of this intelligence is gained by interacting with one's environment to test what is possible, garner information and knowledge, and then to formulate what is desirable, in a similar vein to machines. Nevertheless, a primordial consideration of regulators is what implications machine intelligence and algorithmic trading have for the operation of fair and efficient financial markets. This thesis attempts to provide insight into this issue by analysing algorithmic trading strategies, their impact on market quality and LOB dynamics from multiple angles.

1.1 Limit Order Books & Algorithmic Trading

Electronic LOBs serve as the experimental domain of this thesis, with a particular focus on UK equity market order books. The LOB provides a laboratory in which machines can be readily deployed to generate profitable algorithmic trading strategies that execute order flow in financial instruments over micro-time periods. Contemporary UK equity markets are primordially organised through the collation of limit orders submitted by traders conducting disparate strategies onto a single LOB platform managed by a trading venue. This platform provides a global mechanism for solving the 'search' problem by matching buyers and sellers seeking to establish a trade for a financial security at a target price and quantity (Rosu, 2009; Gould, 2013). The evolution of market structure that has occurred over the past decade, through the electronification of trading and synthesis of divergent order flows onto transparent and accessible pools of liquidity, has improved the quality of financial markets significantly (Cont, 2010; O'Hara, 2015). From systems theory, one can view LOBs as analogous to a 'complex system' that spawns global phenomena through the synergistic interaction of local traders executing heterogeneous trading strategies (Gould, 2013). Modelling LOB dynamics as performed in this thesis is a primary domain of market microstructural academic and regulatory interest given its fundamental impact on resource allocation in financial markets. Attaining a system that allocates capital effectively requires quality secondary markets given that the manifest costs of trading in these markets are implicitly priced into primary market offerings when companies are raising capital (Amihud, 2006). Given this fact, there exist great benefits in deploying deep learning models capable of accurately capturing the complex non-linear dynamics of the LOB, a system with multi-faceted state transition possibilities and a multi-dimensional action space.

Market microstructure is a critical element in trading outcomes and the price formation process as participants interact on LOBs. Specifically, traditional market microstructure is concerned with the design, structure, and rules underlying the operation of trading venue exchanges, which facilitate transactions on secondary markets. Price formation arises naturally from the process of traders interacting in markets, generally on LOBs, which is governed by a set of trading rules that members must adhere to as a pre-requisite before engaging with both the exchange and other participants. Individual trading venues outline their market microstructural parameters in their explicit rules of trading, including the type of acceptable orders, messaging protocols and trader priority rules. Currently, trading venues in the UK that facilitate the trading of equity securities are regulated by the Financial Conduct Authority (FCA) and operate under the Markets in Financial Instruments Directive (MiFID II) regulatory regime implemented in 2018.[1] MiFID II regulates the organisational requirements and operating conditions for firms that provide financial services to clients and trading venues that manage the exchange of financial securities. Consideration must be given as to how market microstructure has come to dominate *both* the price formation process and the relevant inputs into algorithmic trading strategies on equity market LOBs (O'Hara, 2015). This thesis employs datasets derived from the trading activity of FTSE 350 securities which are traded across the primary UK equity venues including the London Stock Exchange, Bats Chi-X, and Turquoise. However, the concepts derived in this thesis have implications for how algorithmic trading and LOB dynamics can be analysed across markets globally.

[1] Directive 2014/65/EU.

Equity market LOB ecosystems have come to be populated by a multifarious array of heterogeneous firms conducting algorithmic trading strategies bounded only by the dynamics of the LOB and market microstructural rules of trading venues and regulators. Firms conducting algorithmic trading suspend the Newtonian conception of time by utilising event-time based automated trading systems to perform statistical, machine learning, fundamental and technical analysis of high-frequency LOB and other financial data sources. This analysis informs a profit-driven trading strategy that holistically encapsulates the defined set of objective rules and algorithmic instructions on how orders are generated, routed and executed on multi-venue exchange LOBs. These objective rules are based on defined quantifiable parameters relating to the trading firm's perception of the state of financial markets which may be integrated into deep learning models that inform strategic execution. The execution of these strategies is generally conducted through algorithms encoded directly onto a trading firms co-located server which links physically to the exchange matching engine through an ultra-low latency direct market access (ULLDMA) connection. This thesis attempts to model algorithmic trading strategies by analysing the real-time algorithmic response, represented by an action on the LOB, to a simulated internal representation of the LOB specific to that trader, proxied by a multi-dimensional LOB feature vector.

Accelerated adoption of automated trading systems[2] and the ubiquitous deployment of algorithmic trading strategies on electronic LOBs provides exigencies for regulators and academics to understand the dynamics of this new trader ecosystem and potential risks flowing into the real economy. Quantifying the impact emanating from interactions and interdependencies between different algorithmic trading strategies first requires a model for identifying and categorising distinct market participants. Algorithmic trading is commonly recognised under the nom de guerre of high-frequency trading (HFT), a label infused into the financial lexicon that has garnered scrutiny within regulator, investor, and academic domains. HFT firms have been identified by the Securities and Exchange Commission (SEC) as proprietary firms with co-located servers executing highly 'sophisticated' and 'high-speed' order submissions, routing and execution strategies over extremely low latency automated trading systems.[3] The primary criticism levelled at HFTs by regulators is their predisposition for trading off micro-period imbalances in supply and demand that infer directional movements in the price. Some posit that this activity detracts from the orthodoxy that trading is beneficial for markets when it contributes to the price formation process by trading off long-term market or security fundamentals. However, we argue that the reality in modern financial markets is infinitely more complex than a binary HFT or non-HFT classification. Further, one could argue that an ultra-low latency automated trading system is a *prerequisite* to remaining competitive in modern equity markets rather than a point of differentiation. This thesis ignores the HFT classification and differentiates algorithmic traders based on their trading *strategy* as opposed to the characteristics of the individual trading *firm* to allow for a holistic representation of the trader ecosystem. Untangling the complexities of the trader environment by extracting the various algorithmic trading *strategies* being conducted allows regulators to more accurately understand how markets function in their contemporary iteration. Implementing this new paradigm of the trader ecosystem provides the tools for regulators to develop relevant policy prescriptions using an empirical and diagnostic approach at the *strategy* level, rendering suppositions based on nebulous generalisations of traders, such as HFT, irrelevant.

[2] ASIC (2013) found that 99.6% of orders on Australian equity markets were generated from an automated order processing system.
[3] Securities Exchange Act Release No. 34-61358, 75 FR 3594, 3606 (January 21, 2010).

1.2 Deep Learning

Deep learning and machine learning provide a computational framework for modelling the dynamics of high-frequency LOBs and the algorithmic trader ecosystem, allowing market practitioners and regulators to better understand the intricacies of modern financial markets. Machine learning is a derivative of artificial intelligence manifestly interested in constructing a system or model capable of learning some non-linear mapping function from which a learnt classifier, or output, can be trained to detect patterns in a set of transformed input features that exist within an internal representation space. Through the hierarchical abstraction of multiple layers of representations attained using non-linear mathematical operations, one can derive a deep learning model capable of learning more complex representations and mapping functions from an expanded hypothesis space (LeCun, 2015; Bengio, 2012; Hinton, 2006). The increased capacity of deep learning models is paired with an ability to untangle highly abstract discriminatory features within a dataset to improve the predictive and generalisation power of the model. This reduces the need for feature engineering, with deep learning optimised machines replacing humans as the arbiter of what features best explain real-world phenomena. Computational logic in this deep learning process largely draws its architectural roots in human cognition by replicating at a simplistic level the decision-making structures from neurophysiological patterns derived from the multifarious array of interconnected neurons in a human's central nervous system, particularly within the cerebral cortex. Whilst the capacity of human cognition is exponentially more advanced than contemporary computing power, it is in deep learning models' ability to *scale* well to highly complex tasks and utilise *parallel computing* techniques for matrix factorisation that allow for an analogous comparison with a human's biological capacity, piquing interest in the application of these models in almost all data-driven business domains.

The primary connectionist deep learning models employed in this thesis to model LOB dynamics and algorithmic trading are Deep Neural Networks (DNN), Recurrent Neural Networks (RNN), and Reinforcement Learning (RL). Furthermore, for various tasks conducted in this thesis we utilise other machine learning models including Random Forests (RF), Support Vector Machines (SVM), and unsupervised K-Means clustering. For the purpose of extending further concepts in this introduction, a brief exposition of the primary deep learning models is provided.

DNNs are complex self-learning algorithms that solve optimisation and supervised machine learning classification problems using models of biological neural structures which mimic human intelligence. The models employ a multi-layered system of *artificial neurons* characterised by a specific neural network *architecture or topology* that models the interconnections and transmission of data between neurons in different *layers* based on a corresponding *activation function* and *learning algorithm* that minimises some *cost function.* At a primitive level, neural networks provide a machine learning framework for approximating a complex non-linear mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$. The function relates input data in the form of a feature vector of neural units, $\mathbf{x}$, through multiple 'hidden' layers of neurons connected to adjacent layers via synapses, represented as weights, $\mathbf{w}$, to an output layer of values or classes, $\mathbf{y}$, based on some activation function, f. Optimisation of the cost function is performed using a learning algorithm in an iterative and continuous process that trains the neural network to find the optimal weight matrix parameters, $\mathbf{w}^*$, in a way that minimises a defined cost function, $\mathcal{L}$. Deep learning and DNNs more generally represent the leading edge of machine learning algorithms for solving non-convex multi-

classification problems in various financial domains (Sirignano, 2016; Dixon, 2017). The success of DNNs stem from their modularity, versatility, and ability to model relationships inherent in high-dimensional data with vast representation spaces and complex statistical dependencies. This thesis extends the application of DNNs into the realm of modelling algorithmic trading strategies based on the interrelationship between trader behaviour and LOB features.

In a similar vein to DNNs, deep RNNs employ optimisation algorithms to approximate the non-linear mapping function, $\mathbf{y} = f(\mathbf{x})$, that relates a set of sequential observations, $\mathbf{x} = \{\mathbf{x}^1, \dots, \mathbf{x}^T\}$, with elemental vectors comprising features that explain the current observable phenomena within the system, to its corresponding class label, $\mathbf{y} = \{\mathbf{y}^1, \dots, \mathbf{y}^t\}$. RNNs extend the temporal dimension of feedforward DNNs by relaxing the assumption that input and output vectors are independent of each other, allowing biological foundations of cognitive memory to be incorporated into the neural network architecture using recurrent feedback connections. The inherent flexible nature of RNNs derives from their ability to incorporate contextual information using a spatial-temporal dimension grounded in the models' explicit architectural feedback loops, allowing for time to be represented recursively. Thus, temporal dependencies in the data can be captured within the dynamical model using previous computations of the data sequence and some internal recurrent memory state representation, $\mathbf{h}$, which is updated as the model is exposed to new information. The highly expressive nature of RNNs extend from the integration of this recursive internal memory component into its architecture and is evident by their ability to compute any representable function under certain assumptions (Siegelmann, 1995). These assumptions include full network connectivity, a finite number of recurrent connections, and sigmoidal activated neurons. In this way, RNN systems are Turing complete, capable of simulating any arbitrary Turing machine for a given input. Thus, this thesis exploits the sequential nature of LOB time series data, which is ostensibly suited to models capable of learning a hierarchical system across the temporal manifold, by utilising am RNN to model the LOB dynamics of UK equity markets.

Deep RL is a computational approach to designing machine intelligence which extrapolates human conceptions of learning and optimal behaviour through interactions with the environment. LOB dynamics can be modelled as a stochastic optimal control problem in a dynamic system within a discrete event time framework in which traders make decisions to maximise some reward function. RL is the machine learning response to this type of *control* optimisation problem involving state-based agents making sequential decisions within a dynamic system analogous to a Markov Decision Processes (MDP) (Howard, 1960; Sutton, 1998). Mathematically, reinforcement learning systems involve a state space $\mathcal{S}$ and action space available to the agent $\mathcal{A}(s_t)$ in which agents receive information about the current state, $s_t \in \mathcal{S}$, execute an available action, $a_t \in \mathcal{A}(s_t)$, receive a reward signal, $r_{t+1} \in \mathcal{R}$, and transition towards a new state s_{t+1} with probability $\mathcal{P}_a$. The objective of the RL agent within this paradigm is to maximise the cumulative discounted returns by conducting a defined policy that maps states to actions, $\pi(a_t|s_t)$. Solving for the optimal policy requires the use dynamic programming and Q-functions to represent the desirability of being in state s_t and taking action a_t before following policy π. However, modelling complex systems with an MDP becomes infeasible when the state and action space become infinitely large, motivating the use of model-free and approximate methods of RL. In this thesis we utilise deep Recurrent RL in the form of Deep Recurrent Q-Networks to provide an alternative method to standard RNNs when modelling LOB dynamics.

1.3 Motivation, Research Objectives and Contributions

Finance has proven to be a critical pillar of the national political-economic construct since the establishment of civilisation in ancient Egyptian and Roman Empires, right through the institution of the banking system in Renaissance Italy, the explosion of consumer credit in the late twentieth century, up to the current epoch. Integrating the near-infinite complexities and intricacies of the modern financial system into a congenial representation of what is or even what should be, is currently beyond the comprehension of any mathematical model. In a similar, though tangential vein, the institution of artificial general intelligence to perform real-world tasks, be it financial or otherwise, alludes many thinkers. Nevertheless, the motivation of this thesis is inextricably connected to the symbiosis of these two strands of academic enquiry. The theoretical development of deep learning models over the past decade coupled with the availability of parallel computing power and implementation of scalable programmatic languages has empowered researchers with the tools to innovate in the domains of computational finance and general artificial intelligence. Any development, however minuscule, to solving one of these tasks provides a building block for foundational change in society that one could argue will be beneficial for the human race in general. Hence, the motivation of this thesis is to provide a small step towards better understanding the complex interaction between algorithmic trading and deep learning, providing insights on how LOB dynamics and specific *strategies* of algorithmic trading impact secondary markets, which in turn have real-world ramifications for capital formation on primary markets.

This thesis is presented as three interconnected chapters, following this introduction and a background chapter, that culminate in a new framework for modelling LOB dynamics and algorithmic trading strategies using deep learning methods. Each chapter is motivated by a different research objective and provides a unique contribution to the academic domain of computational finance and machine learning. We present the research objective and contributions of each chapter separately.

The objective of the first chapter is to develop a deep learning framework capable of defining the *strategy* being conducted by each algorithmic trading firm in UK equity markets. This is performed by optimising and analysing the internal dynamics of a DNN model trained to capture the non-linear relationship between the state of financial markets, represented by a holistic set of LOB features, and the actions executed by each individual algorithmic trader on the LOB. We utilise a proprietary broker-identified dataset provided by the FCA to capture all messages exchanged between algorithmic traders and UK equity trading venues for FTSE 350 securities. A consolidation of all trading activity across the major lit trading venue order books is performed to offer a panoptic view of the state of the market for a security that is exposed to traders. A holistic set of features are extracted from the consolidated LOB which are assumed to serve as inputs into firms trading algorithms, with each feature primarily related to a specific algorithmic trading strategy. Utilising a neurologically-motivated DNN to develop abstract representations of the feature space allows for a more accurate relationship to be modelled between untangled high-dimensional features and the actions executed by individual traders. Given the flexible and modular nature of the DNN, various architectural design components, configurations and hyper-parameters can be evaluated to arrive at an optimal DNN most capable of modelling the relationship between LOB features and trader actions. We then build and train an optimised DNN for each of the unique ISIN-Date-Broker tuple datasets, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, for firms we define as algorithmic traders, attaining

an average training accuracy of 96.4% and generalisation error of 6.9% for these models. The results strongly indicate that the features generated to define the financial environment provide relevant information in explaining traders' behaviour. To confirm the superiority of the model, we test the DNNs performance against comparative machine learning models, finding that the DNN is more suited to mapping the relationship between LOB features and trader actions than other models, attaining a significant reduction in generalisation error relative to both RFs and SVMs. These results highlight the beneficial aspects of DNNs which are able to extract complex hierarchical and abstract feature representations which other comparative models are not able to accomplish.

The manifest technological evolution in market microstructure and automated trading systems over the past decade has challenged researchers, practitioners and regulators, motivating the creation of new tools and methodologies for understanding increasingly complex markets. Evidently, the DNN developed in this thesis represents a powerful deep learning methodology open to regulators, with a remit for maintaining fair and effective markets, to comprehensively understand how and why algorithmic traders interact with the LOB. Furthermore, using techniques developed and presented in this thesis, regulators can employ DNNs to ascertain intricate representations in equity market trading data that drive specific algorithmic trading strategies. This enables regulators to perform oversight functions more efficiently and develop policy prescriptions relevant and necessary for contemporary financial markets.

The objective of the second chapter is to identify, cluster and analyse the impact of different algorithmic trading strategies on UK equity market quality. Algorithmic traders have traditionally been viewed as a monolithic concept within a trader ecosystem that placed precedence on segmenting traders by their primary type of *business* rather than the particular *strategy* that firm is conducting. However, equity market ecosystems have come to be populated by a multifarious array of heterogeneous algorithmic trading firms viewed externally as the amalgamation of market participants conducting disparate strategies to meet individual business objectives. Thus, this thesis aims to develop a *new framework* that analyses the ecosystem from a functional rather than organisational perspective, the type of *strategy* being conducted *across* markets rather than a *firm-level* categorical label.

Three separate procedures are performed to lineate the connection between the algorithmic trading strategy conducted by an individual trading firm and the impact of that strategy on UK equity market quality. First, the internal dynamics of the optimised DNN for each algorithmic trader trained in the previous chapter is deconstructed by performing an axiomatic feature attribution decomposition to understand the relative saliency of each LOB input feature used in the model. An assumption of this thesis is that features of the state and dynamics of the LOB serve as inputs into algorithms, with those DNN features that have higher feature importance representing core inputs into that firms' trading algorithm. Second, unsupervised spherical K-Means machine learning algorithms are employed to cluster each firm into one of five interpretable, exclusive and distinct algorithmic trading strategies – Automated Market Maker, Execution, Microstructural, Momentum, or Technical - based on the feature importance values extracted from the DNNs internal dynamics. Finally, a comprehensive analysis of the aggregated impact of each algorithmic trading strategy on UK equity market quality is performed. This thesis aims to provide input into the dichotomous debate within the academic and regulatory domain that has largely failed to reach a broad consensus on the role of actors in the trader ecosystem and the optimal level of regulation that should be applied to firms conducting algorithmic trading. This debate has significant implications for real-economy wealth creation given the increasingly important role that

algorithmic traders play in secondary market trading. We provide an empirical examination of what impact different algorithmic trading strategies have on various dimensions of UK equity market quality, including price discovery, market volatility, and liquidity.

The development of a new framework of modelling the trader ecosystem that places the algorithmic trading *strategy* as the unit of analysis will provide regulators with a unique perspective on the beneficial and negative effects of different trading strategies. Further, it empowers regulators tasked with maintaining competitive and efficient markets with new tools to adapt their traditional roles of supervision and policy formulation to the contemporary trading environment with advanced information of the dynamics and interrelationships that exist in the new algorithmic trader ecosystem. Regulators also want to understand how these strategies are deployed on markets in order to identify adverse strategies that reduce market quality, allowing for stronger policies to be developed that placates the investor communities' concern that certain traders have unfair competitive advantages in modern financial markets. The models implemented also provide a paradigm for future regulatory or market practitioner endeavours in the domain of 'algorithm testing' given their ability to segment firms into any arbitrary user-defined categories based on how traders interact with the LOB.

The objective of the third chapter is to design a new deep learning framework for modelling non-linear LOB dynamics to predict the future price evolution of a security. The approach implemented in this chapter is to utilise deep RNN and RL models to untangle and extract complex features by building abstract temporal and spatial representations of the LOB state at any point in event-time. These models intrinsically incorporate dependencies from past states, allowing for future midpoint price movements to be predicted with a higher degree of accuracy than standard feedforward DNN models. The proprietary broker-identified dataset provided by the FCA is also employed in this chapter, though the set of features extracted from the LOB are truncated to only those viewable from an external perspective of the order book and financial markets, with trader-specific features rescinded from the data manifold. To arrive at an optimised model, we perform a comprehensive ablation analysis of the various modular architectural schematics, configuration settings and design elements of an RNN trained to predict the next trader action on the LOB. Three separate experiments were then conducted using the optimised RNN to predict the next trader action at or within the best bid-ask price, the directional evolution of the LOB midpoint and a prediction of the midpoint price after x events. The results indicate that RNNs perform well at predicting the future LOB dynamics of UK equity markets relative to the results expected when adopting a naïve random model. Furthermore, the integration of RL techniques in the form of a Deep Recurrent Q-Network results in an improvement of predictive power over the optimised RNN.

This work represents the first known attempt to integrate deep RNN models with deep RL to predict the future dynamics of the LOB. The inherent discontinuities, stochasticity, and instabilities in LOB dynamics makes the task of modelling the various elements of contemporary high-frequency LOBs quite difficult, though of critical importance to regulators, policymakers and practitioners conducting algorithmic trading strategies. Regulators want to understand the phenomena driving aggregate market behaviour to ensure markets are operating correctly. Utilising this approach, policymakers are able to infer as to how the LOB evolves, and when combined with the results from the previous chapters, draw potential implications of the interrelationship between the LOB dynamics and the execution of various algorithmic trading strategies on the order book. This bears interest for policymakers considering 'algorithmic testing' procedures through potential future regulatory paths. For algorithmic traders, deep

learning methodologies for predicting LOB dynamics can be used by firms to improve their order placement algorithms, adjusting orders efficiently so as to minimise adverse selection costs, minimise implementation shortfall and improve best execution, whilst maximising profitability. The success of combining RNN with RL models to perform prediction tasks motivates the need for future research in this area that expands upon the techniques employed in this chapter.

1.4 Outline of Thesis

In Chapter 2 we introduce scientific concepts and relevant nomenclature across a variety of interdisciplinary domains that inform this thesis, including market microstructure, electronic limit order books, algorithmic trading and feature engineering. In Chapter 3 we introduce a deep learning framework capable of defining the *strategy* being conducted by each algorithmic trading firm in UK equity markets. In Chapter 4 we identify, cluster and analyse the impact of different algorithmic trading strategies on UK equity market quality. In Chapter 5 we develop a new framework for modelling non-linear LOB dynamics to predict the future price evolution of a security. Finally, in Chapter 6 we provide conclusory remarks on the thesis and present potential future areas of research in the domain of algorithmic trading, LOB dynamics and deep learning.

Chapter Two

2 Market Microstructure & Algorithmic Trading

This chapter introduces scientific concepts and relevant nomenclature across a variety of interdisciplinary domains that inform this thesis. Financial concepts relevant to the research question and applicable throughout the spectrum of deep learning models implemented in this thesis are expanded upon. Thus, this chapter initialises with a comprehensive introduction into the scientific field of market microstructure with a focus on UK equity markets in Section 2.1. The primordial system pertaining to this microstructure - the limit order book - is explored in Section 2.2. Algorithmic Trading systems and strategies executed on these limit order books represent a critical unit of analysis throughout this thesis and are holistically set out in Section 2.3. Finally, the process of extracting relevant features that arise naturally from a study of the literature related to algorithmic trading, which serve as inputs into the deep learning models employed throughout this thesis, are described in Section 2.4.

2.1 Microstructure of UK Equity Markets

Technological innovation has been transforming financial markets since their inception by constantly challenging the conventions and structure inherent in their market design. Modern financial markets are characterised by the ascendency of algorithms over humans as the dominant investment decision-making force, the spatial contraction of markets onto electronic platforms, the temporal reduction of the investment process to the millisecond level, exponential decay in latency rates, and the rise of market microstructure as the dominant component in trading strategies. In her seminal paper on market microstructure, O'Hara (1995) described the critical role that explicit trading rules have in the price formation process and trading outcomes when participants interact with financial markets. The current iteration of financial markets has come to be dominated by a limit order book (LOB) market structure, with the primary market participants utilising technologically advanced automated trading systems (ATS) to execute complex algorithmic and high-frequency trading strategies. In a more recent paper, O'Hara (2015) reflected on this manifest algorithm-dominated reality by theorising that modern market microstructural parameters are increasingly primordial components of these algorithmic trading strategies (O'Hara, 2015).

Market microstructure represents the 'trading mechanism' of an exchange which has ramifications for price and liquidity formation, the price discovery process, and explicit and implicit transaction costs for traded financial securities (O'Hara, 1995). Specifically, traditional market microstructure is concerned with the design, structure and rules underlying the operation of trading venue exchanges, which facilitate transactions on secondary markets. Secondary equity capital markets are both a theoretical representation of the liquidity supply and demand dynamics of a listed financial security, and a practical mechanism for facilitating trades between buyers and sellers of those securities. Price formation arises naturally from the process of traders interacting in markets, generally on LOBs, which is governed by a

set of trading rules that members must adhere to as a prerequisite before engaging with both the exchange and other participants. However, the price formation process is increasingly shaped through structural shifts in financial market microstructure driven by technological advances in algorithmic trading and its component systems (O'Hara, 2015). Automated computer-driven algorithms now conduct the plurality of trading activity on both US[4] and UK[5] equity markets.

Individual trading venues outline their market microstructural parameters in their explicit rules of trading, including the type of acceptable orders, messaging protocols and trader priority rules. The bulk of equity trading in the UK is conducted on trading venues, lit and dark order books, or over-the-counter (OTC) directly between two parties. Currently, trading venues in the UK that facilitate the trading of equity securities are regulated by the Financial Conduct Authority (FCA) and operate under the Markets in Financial Instruments Directive (MiFID II) regulatory regime implemented in 2018.[6] MiFID II regulates the organisational requirements and operating conditions for firms that provide financial services to clients and trading venues that manages the exchange of financial securities. This thesis is focused on the analysis of periods under the previous regulatory framework of MiFID I.[7] Where relevant, we identify and explain the impact on the analysis being conducted from differences between the MiFID regulatory regimes.

The implementation of the MiFID I framework altered the ecosystem in which secondary market trading of equity securities was conducted with the objective of harmonising transparency requirements across the EU and driving competition between equity trading venues. The MiFID I legislation largely initiated an intense period of inter-exchange competition for order flow from trading participants resulting in a series of exchange consolidations with the London Stock Exchange Group taking a controlling stake in Turquoise MTF and BATS Global Markets buying out Chi-X Europe. Further, this framework largely facilitated the rise of new trading platforms in EU financial markets (Gresse, 2013). This included the formation of Dark Pools of liquidity and Multi-Lateral Trading facilities. New trading platforms were allowed following the repeal of the 'concentration rule' under the Investment Services Directive[8] -the MiFID I precursor - that required equity trading to be conducted on national stock exchanges. These platforms were developed largely as a response to the demand from participants for waiving pre-trade transparency requirements to minimise market impact. The growth of these platforms increased quickly following the liberalisation of trading venue requirements and removal of competition barriers under MiFID I with the ability to develop pan-EU trading venues benefiting from economies of scale (ECB, 2017).

Trading activity on UK equity markets is predominately focussed on liquid equity instruments that compose the FTSE 350 index. Throughout this thesis, the datasets employed use securities sampled from this index. All composite securities of the index have a primary listing on the London Stock Exchange and are traded on the exchanges electronic order book. Each FTSE350 security is allocated to a segment

[4] TABB Group estimated that 52% of trading is conducted by HFT in 2017, which is a decrease from a peak of 61% in 2009 (TABB Group, 2017). JP Morgan in 2017 estimated that 60% of trading activity on US equity markets involves quantitative or passive investing accounts whilst only 10% of trading is performed by traditional 'fundamental discretionary trader' accounts that base trading decisions off stock price fundamentals (CNBC, 2017).

[5] HFT accounted for 77% of order flow volume on UK equity markets in 2010 (TABB Group, 2011).

[6] Directive 2014/65/EU.

[7] Directive 2004/39/EC.

[8] Council Directive 93/22/EEC of 1993 on investment services.

– SET0 for the most liquid FTSE 100 securities, SET1 for all remaining FTSE 100 securities, and STMM for FTSE 250 securities. These segments are reviewed quarterly, with securities reassigned to an index or segment dependent on various factors including their average daily order book turnover, time-weighted spread and market capitalisation value. Each security is defined by their ISIN, SEDOL, Issuer Name, and country of issue, with all securities traded with the GBX currency, which refers to GBP in pence. Minimum lot size of one unit is required for all order books.

2.1.1 Trading Venues

Trading venues in the UK that facilitate the trading of equity securities operate under the rules of the Financial Conduct Authority (FCA) and regulatory framework of MiFID I.[9] These venues may be either a Regulated Market (RM), Multilateral Trading Facility (MTF), or a Systematic Internaliser (SI). The two primary trading mechanisms utilised by the trading venues are lit order books, where liquidity is transparent or semi-transparent, and dark order books, where liquidity is non-transparent. Trading conducted outside these venues is generally done so on an irregular or ad hoc basis and is referred to as Over-the-Counter (OTC) trading.

RMs[10] in the UK are authorised as Recognised Investment Exchanges (RIE) by the FCA[11] to conduct primary listings for equity securities and carry on the business of facilitating trading in secondary markets. This process involves aggregating third party buying and selling interests in financial instruments to generate a contract and transaction between willing counterparties. MTFs[12] are trading venues operated by market operators or investment firms that also facilitate trading in secondary markets in accordance with their non-discretionary rules. MTFs serve as a competing trading venue to RMs and have similar regulatory requirements, follow similar rules, and utilise analogous structures to those of RIE RMs. SIs[13] are investment firms that internalise client order flow on a frequent and systematic basis by matching buy and sell orders on an internal platform using their own account, rather than sending orders to a RM or MTF. Many of the large international investment banks operate SIs.

Dark pools (DP) are trading facilities operated by either a RM, MTF, or an investment firm that pool liquidity and match large anonymous orders. While DPs are not specifically captured under MiFID I, they were developed as a response to concerns of information leakage created by the requirement for trading venues to provide pre-trade transparency of the price and depth of orders for equity securities. DPs circumvent these transparency requirements by facilitating transactions between two parties submitting non-transparent orders that meet a *large in scale* (LIS) exemption threshold that allows pre-trade transparency requirements to be *waived* under MiFID I.[14] These legal exemptions allowed DPs to use a mechanism to issue waivers for transparency obligations for all orders on the order book meeting the LIS, hence creating dark pools of liquidity.

[9] MiFID I has been superseded by MiFID II as of 3 January 2018. However, this thesis focuses on periods of analysis predominately prior to 2018 under the previous MiFID I regime.
[10] As defined in Article 4(1)(14) of MiFID I.
[11] Under the Financial Services and Markets Act 2000 (UK).
[12] As defined in Article 4(1)(15) of MiFID I.
[13] As defined in Article 4(1)(7) of MiFID I.
[14] Large-In-Scale Waivers from Pre-Trade Transparency in accordance with Article 20 of MiFID I.

There are three primary stock exchange operators that offer a range of lit displayed and dark non-displayed electronic order book-based trading platforms which facilitate secondary market trading of liquid UK equity stock securities. In general, lit order books provide quasi-transparent aggregations of limit orders with displayed prices and market depths, while dark order books aggregate non-displayed liquidity that generally transacts at the mid-point of the primary exchange reference best bid-ask price (BBO). The three primary stock exchange operators and their operating ISO 10383 Market Identified Codes (MIC) are as follows - London Stock Exchange (XLON), Turquoise Global Holdings (TRQX) and Bats Chi-X Europe (BCXE). Operating MICs identify the trading venue at the exchange level and Segment MICs can be used to further break down the trading venue into multiple segments or order books.

The London Stock Exchange Group (LSEG) operates the London Stock Exchange (LSE) which is a RIE that provides equity primary market listing services for UK companies for capital formation purposes, and secondary markets over which equity securities are traded on the LSE lit order book (XLON). Components of the FTSE 350 and other highly capitalised companies are generally listed on the LSE Premium Listed Main Market (PLMM). These stocks are traded through the Stock Exchange Electronic Trading Service's (SETS) electronic order book on the LSE-owned Millennium Exchange trading platform. Trading in so-called 'upstairs' parallel OTC dealer markets is allowed under the rules of the LSE if the order meets certain criteria under the rules of the exchange.

Turquoise Global Holdings (Turquoise) operates a pan-European MTF that facilitates the trading of all FTSE350 component securities on two discrete electronic order books. Turquoise is majority owned by LSEG in collaboration with a consortium of investment banks. Turquoise Integrated Lit Order Book (TRQX) is a transparent on-exchange limit order book with fully displayed liquidity from persistent order types and non-displayed liquidity that meets the Large in Scale (LIS) threshold, on which participants can place orders and transact using the Millennium Exchange trading platform. Turquoise Plato Order Book (TRQM), formerly known as the Turquoise Midpoint Dark Order Book, is a non-displayed primary market price-referencing dark pool order book. The dark pool is used by participants to facilitate block trading where orders are prioritised by size and executed at the midpoint of the primary market BBO price through continuous matching of orders or through periodic matching during random intraday periods.

Bats Chi-X Europe (Bats) is a pan-European stock exchange operator formed in November 2011 through the merger of two MTFs, Bats Trading and Chi-X Europe, before registering as a RIE in May 2013. Conducting business as an RIE has allowed Bats to compete with the LSE in both the primary listing for company shares, in addition to secondary market trading order flow. Bats offers two displayed Integrated Order Books, Bats Europe BXE (BATE) and Bats Europe CXE (CHIX), that provide visible 'lit' displayed and non-displayed liquidity for participants, and separately two 'dark' Order Books - Bats Europe BXE Dark Order Book (BATD) and Bats Europe CXE Dark Order Book (CHID) - both with matching at the midpoint of the primary market.

Trading on UK LOBs is conducted during scheduled trading day sessions on all weekdays, except public holidays, between 07:50 and 16:40 GMT. The trading day generally follows the chronological process of four scheduled trading sessions; an opening call auction on the primary market, continuous trading on all lit and dark order books, a closing auction on the primary market, and a final closing price crossing

session. Unscheduled trading sessions may also arise during the course of the trading day pertaining to the occurrence of certain events.

The Millennium Exchange Business Parameters (LSEG, 2017), outlines the dynamic price format schedule or tick sizes for all securities traded on UK equity exchanges in the period of analysis in this thesis. An instrument's tick size is a numeric currency value that represents the minimum allowable increment in which prices can be entered onto the order book. During the MiFID I regime, tick sizes in European equity markets were generally defined in the trading parameters outlined by each individual exchange.[15] The absence of a harmonised EU tick size regulatory regime under MiFID I facilitated a period of intense inter-venue competition for order flow as venues allowed increasingly lower tick sizes. This situation culminated in a broad agreement brokered by the Federation of European Securities Exchanges (FESE) for industry self-regulation through the imposition of a quasi-harmonised tick size regime in Europe in 2009. Securities traded in Europe generally follow one of four dynamic tick size tables with tick sizes being a function of a security's order price such that a security will have different tick sizes for two orders which are submitted at prices on opposite sides of a certain price threshold. UK equity venues followed the FESE 1 price format schedule for the most liquid FTSE 100 securities and FESE 2 for the less liquid FTSE 100 and FTSE 250 securities.

2.1.2 Order Types

Modern UK equity markets support and accept a multifarious array of order types for securities. Order types can be segmented along several dimensions, including whether the order is persistent or not, whether the order is displayed or not-displayed, the orders 'time in force' which refers to how long the order remains on the LOB and the trading session in operation when the order is submitted. The proliferation of order types on UK stock exchanges has been driven by the increased complexity, fragmentation, innovation and competition within the secondary equity market following the MiFID I regulations. Innovative order types that allow traders to implement complex strategies more simply have become a key point of differentiation between venues to attract high-order flow traders. Order types differ depending on whether they are submitted to integrated lit order books (XLON, TRQX, BATE, CHIX), dark pools (TRQM, BATD, CHID), or across multiple order books.

Integrated Lit order books allow the submission of *persistent* limit-type orders, *non-persistent* market-type orders and *auction* orders during the trading day. Order submissions on lit UK order books are prioritised by the LOB matching engine on a price, display type then time basis. Traders generally attach a 'time in force' attribute to orders that define whether the order is available all day (DAY), until a certain time (GTT), or until a certain date (GTD).

Persistent orders submitted by traders remain active on the LOB until cancelled or executed when matched against either a competing resting or aggressive order. Five common persistent order types that are employed on UK equity markets include:

- *Limit orders* are fully displayed with an attached target price and volume.

[15] MiFID II introduced a pan-EU regime for setting tick sizes in equity markets under RTS 11.

- *Iceberg orders* are partially displayed limit orders that partition their volume into a publicly visible 'peak quantity' and a non-displayed portion. These orders are available through pre-trade transparency waivers built into MiFID I.
- *Hidden orders* are non-displayed orders that must exceed the minimum LIS volume threshold required under the regulations and set by the exchange.
- *Executable quotes* are displayed double-sided orders available to registered market makers that generally must meet a minimum spread and size threshold set by the trading venue.
- *Stop limit orders* are orders that remain unelected until a specific stop price has been reached at which point the order is automatically converted into a limit order and entered on the LOB.

Non-persistent orders enter the LOB as non-displayed orders and require immediate execution or expire. Three of the most common non-persistent orders used on UK venues include:

- *Market orders* are both unpriced and non-displayed orders that immediately execute against resting liquidity at the best price available on the LOB up to a volume specified. Market orders are guaranteed to be filled during continuous trading, if enough liquidity is present on the LOB, though there is no execution price guarantee leading to the risk of significant slippage costs.
- *Immediate or cancel (IOC) limit orders* are priced orders that execute immediately either partially, or in full, against resting persistent opposite-side orders at the target price with unexecuted volume expiring immediately.
- *Fill or Kill (FOK) limit orders* are priced orders that must be executed in full against resting orders otherwise the whole order expires.

Additional auction specific orders may be entered during trading on the LSE but are only injected onto the LOB during the specified auction periods. These include *at the open (ATO), at the close (ATC)* and *good for auction (GFA) limit orders* that are entered at the beginning of the next opening auction, closing auction, or any auction period, respectively.

Dark order books allow for transactions to be executed at the midpoint of the primary market best bid offer (PBBO). Non-displayed orders allowed on UK dark pools include:

- *Midpoint pegged limit orders* with an attached price that represents the cap or floor price at which a buy or sell order may be executed
- *Midpoint pegged IOC order* that is executed immediately against marketable orders resting on the dark pools LOB with remaining quantity expiring immediately.

Inter-book order types allow market participants to execute a specific inter order-book strategy that routes orders through multiple order books. Several common orders employed on the Bats, Chi-X and Turquoise dark pools include:

- *Dark Lit Sweep Limit orders* access both dark and lit liquidity by placing a limit on the order book with the best available price. If both order books have the same price, liquidity flows to the dark pool to benefit from potential midpoint matching.
- *Dark Lit Sweep IOC orders* are IOC limit orders that sweep lit and dark order books and execute at the best price up to the volume specified, with remaining volume expiring.
- *Parallel Strategy orders* allow market participants to develop a strategy for accessing liquidity across multiple lit and dark order books. These orders are primarily offered by BATS Chi-X.

2.2 Limit Order Books

Electronic Limit Order Books (LOB) are the primordial trading mechanism in contemporary UK equity markets for solving the 'search' problem by matching buyers and sellers seeking to establish a trade for a financial security at a target price and quantity. Optimal trading strategies are designed to perform complex quantitative analysis to 'search' for a counterparty willing to meet the terms on which the two parties agree to trade a financial security. Trading venues' LOBs provide a global solution to this problem by collating limit orders submitted by traders conducting disparate strategies onto a single platform. This results in the aggregation of liquidity, composed of ex-ante commitments to buy or sell a security at a specified price and volume, which serves as a representation of that securities supply and demand dynamics.

Electronic order-driven LOBs have become the dominant trading platform to exchange securities on both UK and global equity markets, with over half of all financial trading activity conducted through electronic LOBs (Rosu, 2009; Gould, 2013). This evolution in market structure has improved the execution time for market participants significantly (Cont, 2010). Trading venues generally have differing business models in how they attract liquidity and trading activity on their exchange, with diversity in microstructural rules between different venues. However, regulatory requirements under MiFID I and MiFID II, in addition to oversight of markets performed by the FCA, require objective trade matching rules, processes and transparency requirements to be explicitly defined within the venues trading rules. In addition, under MiFID II venues must adhere to regulatory laws, and guarantee fair access for trading firms to ensure information asymmetry does not dislocate quality, liquid and fair markets.

From systems theory, LOBs are analogous to a 'complex system' that spawns global phenomena through the synergistic interaction of local traders executing heterogeneous trading strategies (Gould, 2013). Modelling LOB dynamics is a primary domain of market microstructural academic and regulatory interest given its fundamental impact on resource allocation in financial markets. The complex non-linear dynamics and high-dimensionality of the LOB arise from traders' ability to submit orders at near-infinitesimal price points and price levels, constrained only by microstructural resolution parameters, the tick size, π, and minimum order lot size, ϕ. Traders entering a LOB market are faced with a multi-dimensional action space with complex state transition dynamics that result in a system predicated on volatile market interactions (Shek, 2011). The set of possible actions, $\mathcal{A}$, comprises the type of order, price, volume and side of the order. These LOBs manifestly involve traders submitting, amending, or cancelling bid or ask orders, $O_x^{\mathcal{A}}(P_x, V_x, T_x)$, at time T_x, with discrete price P_x, and volume V_x.[16] A *bid* or *ask* order commits the trader to buy or sell $|V_x|$ units of the security at a price no less than or greater than, respectively, the order price P_x. Trading venues generally permit a series of order types that can be entered on the LOB at various price points in tick increments. Upon submission of a bid or ask order onto an exchange's LOB during *continuous trading periods*,[17] the trade matching engine algorithm will determine whether the order can be matched with a similarly priced resting ask or bid order already sitting on the LOB. If two orders are matched then a trade is executed, otherwise, the order will 'rest'

[16] See Gould (2013) for a mathematical formulation of the mechanics underlying LOBs.

[17] *Periodic* order matching is used during auction periods by comparing buy and sell orders at a specific point in time and matching those at an execution price point that maximises the transaction volume.

passively on the LOB until it is cancelled or expires. Lit trading venues in the UK generally first follow a price, then a display, and lastly a time queuing priority *matching principle,* which regulates order matching processes between market participants. Thus, resting orders at a more aggressive price are executed against first, with equally priced orders then split by transparency with a precedence of displayed over non-displayed orders, before time priority rules apply with orders executed against in the chronological order in which they entered the venues matching engine.

UK equity LOBs organised by these matching methods and principles allow transactions to formulate when two commensurate opposite-sided orders match. The resting 'passive' order is filled up to the volume of an incoming 'aggressive' order, $|V_x|$, and the order is deleted from the LOB when the volume is fully filled, or it is cancelled. If a new incoming order does not match against another order, it becomes 'active' and is placed as a passive order on the LOB where it rests until it is cancelled or matched against a future order. The amalgamation of active persistent orders that are unfilled or only partially filled together formulate the LOB, $L(t)$. This term fully represents the composition of liquidity on the LOB available to traders at time point t which may be based on chronological or event time. The LOB $L(t)$ can also be decomposed into active bid orders, $B(t)$, and active ask orders, $A(t)$, with the LOB mathematically formulated as:

$$L(t) = \{O_x\}_x \qquad O_x \in L(t)$$

$$O_x^{\mathcal{A}} = (P_x, V_x, T_x) \qquad P_x \in \{\pm j\pi | j = 0,1,2,\dots\} \quad |V_x| > 0 \qquad V_x \in \{\pm k\phi | k = 1,2,3,\dots\}$$

Several critical features related to liquidity, spreads and market depth can be extracted from the LOB. The LOB *depth* for a bid, $D^B(p,t)$, and ask, $D^A(p,t)$, is the summation at time t of all active bid and ask orders, respectively, submitted at price point p. The depth profile for the bid-side, $(p, D^B(p,t))$, and ask-side, $(p, D^A(p,t))$, represent the depth available at each price point for both sides of the LOB. Other relevant characteristics of the LOB arise from an inspection of the LOB's depth profile. Important price measures include the *best bid price*, $BBP(t)$, the highest price a trader is willing to pay for an asset, and the *best ask price*, $BAP(t)$, the lowest price a trader is willing to sell an asset. Together these two prices form the *bid-ask spread*, $S(t)$, which represents the difference between the best bid and ask (BBO), and can be used to compute the mid-point, $MP(t)$, an arithmetic average of the best bid and ask. Cumulated liquidity at the BBO is decomposed into the *best bid depth*, $D^B(BBP(t),t)$, the total depth at the best bid price, and the *best ask depth*, $D^A(BAP(t),T)$, the total depth available at the best ask price. Computing depth requires consideration of all orders placed during the preceding periods t' before current time t. Mathematical formulations follow:

$$D^B(p,t) = \sum_{(O_x \in B(t') | P_x = p)} |V_x| \qquad D^A(p,t) = \sum_{(O_x \in A(t') | P_x = p)} |V_x|$$

$$BBP(t) = \max_{O_x \in B(t)} P_x \qquad BAP(t) = \min_{O_x \in A(t)} P_x$$

$$S(t) = BAP(t) - BBP(t) \qquad MP(t) = \frac{BAP(t) + BBP(t)}{2}$$

The submission, amendment or cancellation of orders represent the sole actions available to a market participant when interacting with the LOB. In modern markets, orders have become increasingly complex with a multitudinous array of order type permutations available within and across trading venues. Further, orders on LOBs can be decomposed into passive and aggressive orders. Passive orders

are limit orders that do not immediately match with a resting order on the LOB, and hence, themselves end up resting on the LOB as active orders. Passive bid or ask orders submitted at price, P_x, meet the conditions, $P_x < BAP(t)$, or, $P_x > BBP(t)$, respectively. Alternatively, aggressive bid or ask orders are those that execute immediately against resting active limit orders already on the LOB and that meet the conditions, $P_x \geq BAP(t)$, or, $P_x \leq BBP(t)$, respectively. These orders can be referred to as market orders and execute against liquidity on all prices in a sequential process up to the point that the market orders total volume is filled. If incoming aggressive orders match against the entirety of the best bid or best ask volume then a new best bid or ask price is established. Traders generally use both passive orders, which allow for price over execution certainty, and aggressive orders, which enable traders to execute sell or buy trades with certainty at the best bid or ask price, respectively, depending on the context of the environment and needs of the trader.

The LOB for a single lit trading venue is illustrated as a queuing system in Figure 2.2.1 with each resting passive order represented by 'blocks' indicating the price of the order, defined along the horizontal axis, and the orders absolute volume $|V_x|$ of shares, scaled to the vertical axis. Ask orders are coloured in red and bid orders in blue. Queues at each price point may involve orders of diverse sizes submitted by multiple traders. Using conventions inherent in microstructure research, queues are viewed from a relative perspective to the best bid-ask, rather than as a series of arbitrary actual monetary prices. The figure indicates that the securities current best ask price of 32.16, best ask depth of 300 shares, best bid price of 32.14 and bid depth of 200 shares, with a commensurate quoted spread of 2 basis points and midpoint price of 32.15. Evident from the figure, an aggressive bid order for 300 shares with a limit price of 32.16 is received by the venues trade matching engine. This involves a trader crossing the spread, thus, paying more than the prevailing mid-price. The resulting action leads to a transient deficit that depletes depth at 32.16, setting of a new best ask price of 32.17, widening the prevailing quoted spread and increasing the mid-price. This order can be considered in conjunction with a passive bid order submitted at 32.11 for 200 shares that has no commensurate impact on the dynamics of the LOB around the region encompassing the quoted spread.

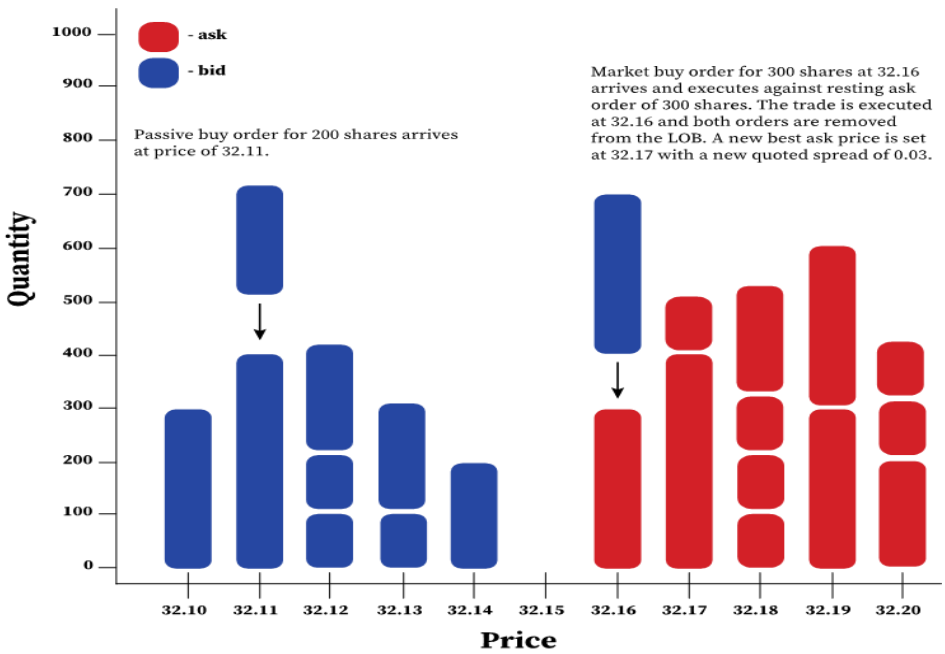

Figure 2.2.1 – Representation of a simulated limit order book. Bid orders are represented between the price points of 32.10-32.14, and ask orders are represented between the price points of 32.16-32.20.

2.3 Algorithmic Trading Strategies

Algorithmic Trading (AT) firms utilise automated trading systems to perform statistical, machine learning, fundamental and technical analysis of high-frequency limit order books (LOB) and other financial data sources. This analysis informs a profit-driven trading strategy that defines the objective rules on how orders are submitted, amended and cancelled on an exchanges LOB. These objective rules involve defined quantifiable parameters relating to the trading firms' perception of the state of financial markets. Rules are incorporated into algorithms that are executed automatically and autonomously using the firms ATS. Algorithms are generally encoded and processed through the AT firms co-located exchange server using a logic translating computer language such as C++ and Java, or directly through Filed-Programmable Gate Array (FPGA) chips. Complex trading strategies can generally be refined into a set of related input parameters analysed iteratively in real-time, serving as the algorithm's logic and informing its set of basic outgoing instructions that explain when and how to interact with various trading venue LOBs. These algorithmic trading inputs are represented by the set of features developed in this section, summarised in Section 2.4, and mathematically formulated in Appendix A.

The past few decades have seen transcendental advancements in technology driving improvements in automated trading systems through which market participants execute AT strategies. The evolution towards complex automated systems has been driven by the competitive forces in financial trading that places speed as the predominant determinant of trading strategies' profitability. Modern AT firms invest in technologically sophisticated ultra-low-latency direct market access (ULLDMA) and co-located automated systems. The contemporary iteration of competitive automated systems are capable of performing real-time quantitative analysis that informs algorithmic instructions on how to generate, route and execute orders in multi-venue markets. These functions can holistically be referred to as an algorithmic trading strategy, with programmatic instructions of the algorithm being the quintessential point of differentiation between various firms. ULLDMA has fundamentally altered the dimensions of human involvement in the trading decision, order management and risk management process. Trading algorithms are generally hardcoded into hardware systems, through technology such as FPGA chips, within the firms co-located server located physically within trading venues proximity hosted data centres. Market participants succeeding in liquid order-driven markets require the capability to adapt their trading strategies in conformance with the fast-paced nature of modern LOBs driven by structural shifts in the market microstructure.

Equity market ecosystems have come to be populated with a multifarious array of heterogeneous Algorithmic Trading firms bounded by the dynamics of the LOB. The market's trader ecosystem can be viewed externally as an amalgamation of disparate market participants, which each executing optimal order placement algorithmic-based strategies to meet their individual business objectives. These strategies may be derived both endogenously, based on unique intrinsic parameter formulations and evaluations, or exogenously, based on the dynamics and analytical formulations from the current state of the LOB. Optimal AT order placement algorithms involve formulating a pre-conceived trading strategy framed as an optimisation problem by minimising some objective function, subject to defined constraints related to the specific market or problem, over the parameter variables of the order placement algorithm. The order placement algorithm must decide what type of order to place, generally a limit or market order, the volume and price of the order, normally expressed as a distance from a reference point

such as the best bid or ask, and the specific trading venue LOB the order should be placed on, generally referred to as the routing decision.

The AT ecosystem has traditionally been separated at the highest level into buy and sell-side firms. Buy-side firms with long-term horizons, such as pension or mutual funds, that generally utilise broker execution algorithms to rebalance portfolios. Sell-side proprietary trading firms, such as market makers, arbitrage traders and quantitative hedge funds, generally have shorter-term horizons. Regulators and market practitioners have increasingly expanded their scope of analysis and understanding of complex equity markets' trader ecosystems to consider the role of algorithmic trading in financial markets, within a framework that seeks to maintain the tenets of fairness and efficiency. However, AT is a complex phenomenon with any one firm capable of executing a plethora of strategies across multiple desks in the same security at the same time.

Regulators and researchers need to adapt to the contemporary trading environment and ecosystem by considering a framework for policy, oversight and analysis that places the AT *strategy* being executed as the fundamental unit of analysis rather than the traditional concept of the trading *firm* as the point of inquiry. The trading ecosystem can no longer be dichotomised into generic categories of buy and sell-side firms, or high-frequency trading (HFT) and non-HFT, given that any firm may be conducting a multitude of trading strategies in or across many financial products, regulatory jurisdictions and geographic boundaries, at a single point in time. Further, the fragmented nature of equity markets requires a panoptic view of *consolidated* limit order books to truly understand how markets function and can be improved. Thus, this thesis recommends assessing trading activity from a functional rather than organisational perspective, the type of *strategy* being conducted *across* markets rather than a *firm-level* categorical label, when analysing the impact of trading on financial markets.

Various frameworks for segmenting AT strategies have been developed (ASIC, 2013; Aldridge, 2013; Gomber, 2013). To analyse the impact of AT strategies on market quality this thesis develops a *new framework* of segmenting strategies at the broker-level based on the *underlying strategy* being conducted by the firm, which is proxied by a firm's trading algorithm's response to various *features of the LOB* market microstructure phenomena and the larger macro-financial market. AT strategies are categorised as employing either an Automated Market Maker, Execution, Microstructural, Momentum or Technical type algorithms.

Automated Market Maker algorithms supply liquidity by positing double-sided bid-ask quotes that match against traders that demand liquidity. They profit from capturing the spread and focus predominately on managing inventory and execution risk in the presence of asymmetric information. Execution algorithms have traditionally been the domain of buy-side AT brokers attempting to liquidate a large trade for their clients in an optimal manner that minimises execution costs and market impact. These firms generally trade on an agency basis with defined stocks, directions and time frames specified by the clients and profits driven by beating a specific benchmark or attaining price improvement. Microstructural algorithms generally trade off microstructural signals prevalent on the LOB. Momentum strategies focus on micro changes in momentum LOB signals when placing orders. Conversely, Technical strategies use higher level momentum, price reversion and other technical indicators to drive order placement.

2.3.1 Automated Market Making

Automated Market Making (AMM) or Electronic Liquidity Provision (ELP) strategies use algorithms that ostensibly supply liquidity on both sides of the LOB through the submission of passive resting bid and ask limit orders. Theoretically, AMMs can be viewed as a microcosm of the market itself, as they provide continually updated inside passive double-sided resting limit order quotes that follow the mid-price throughout the trading day. The traditional manifest role of AMMs in financial markets is to solve the temporal dislocation of fundamental supply and demand dynamics between buyers and sellers through liquidity provision. These firms absorb transitory excess demand or supply through their inventory holdings and extract profits from capturing the pervading bid-ask spreads through multiple round-trip transactions, or through liquidity rebates that trading venues may offer AMMs as incentives for providing liquidity. AMMs dynamically update their quotes throughout the trading day based on real-time data of the LOB and generally hold only small net inventory positions aiming to end the trading day with flat or very low overnight holdings to minimise risk.

Equity trading venues categorize AMM traders through Designated Market Maker (DMM) and Supplemental Liquidity Provider (SLP) agreements where AMMs are contractually obligated, by the exchanges on which they trade, to maintain an orderly market by providing sufficient liquidity in certain stocks even during times of severe market volatility. Some trading venues offer liquidity rebates for each share sold or purchased from a passive bid or ask order, respectively, to cover AMMs' trading costs and exchange fees in addition to the profit generated from the bid-ask spread. This practice has led to the development of rebate trading strategies (Aldridge, 2013). While many equity markets generally offer DMM and SLP agreements to registered market makers, many proprietary trading firms operate an AMM trading strategy without an explicit DMM agreement with the trading venue. The London Stock Exchange (LSE) requires member firms acting in a market making capacity in a specific financial security to be registered with the exchange (LSEG, 2018). Registered AMMs in order-driven markets are required to maintain a two-sided executable quote, which includes a bid and ask order of sufficient size and within a maximum spread, for 90% of the daily regular trading period. Bats Europe offer a Liquidity Provider Program (LPP) for market makers willing to provide liquidity on their integrated lit CXE and BXE order books by adhering to certain parameters. The LPP program requires market-makers for a specific security to maintain executable quotes larger than 5,000 Euros within 0.25% of the relevant order books best bid-offer (BBO), for 80% of the time during regular hours of the trading day. Bats Europe CLP MMs do not receive any rebates or pay any fees in addition to those that are received and incurred during standard trading. Bats Europe also offers a rebate-based scheme for issuer sponsored securities, through the Competitive Liquidity Provider (CLP) program which is designed to increase the depth of the order book and tighten spreads by incentivising market maker liquidity providers through rebates based on their quoting rather than trading activity. The parameters of the CLP program are determined on a security-by-security basis by Bats Europe in conjunction with the security issuer.

The nexus between modern equity market microstructural rules, the fragmentation of equity markets and shifting technological barriers to entry in algorithmic trading, has served as the fundamental catalyst for the automation of market making strategies. The competitive friction between trading venues competing for order flow in highly fragmented equity markets has increased the necessity for AMM strategies to transgress previous intra-venue trading frictions by instituting intra-venue strategies

applied across multiple trading platforms in various jurisdictions. Further, modern market microstructure settings provide opportunities for AMM strategies to be deployed by firms without any contractual DMM or SLP agreements with exchanges. Thus, these firms have no permanent quoting obligation or requirement to provide liquidity in stressed markets. Recent legislative changes under MiFID II RTS 8 will impact how AMM strategies are conducted as the legislation will provide additional requirements on firms quoting behaviour, activity level, legal requirements in relation to entering AMM agreements with trading venues, and level of rigour in their systems and controls to fulfil their obligations.

AMM strategies cannot be categorized as homogenous passive trading strategies, as these trading firms commonly employ aggressive trades for the purpose of rebalancing inventory, profiting of trading signals outside the general confines of an AMM strategy or for some other reason implicit to the trading firm. However, various idiosyncrasies are inherent to AMM strategies. AMMs are characterized by very low target intra-day and closing inventory positions to avoid the risk of holding stocks overnight. These firms profit from executing a high volume and low margin strategy in markets where risk is generally commensurate with the level of stock liquidity, with more liquid stocks providing tighter spreads. Further, AMM firms' profitability has been shown to proportionally increase with the level of trading activity of the firm rather than the time spent providing liquidity on the LOB (Sandas, 2001). For these reasons AMM traders, particularly firms trading with high-frequency and market-leading trading systems, trade mostly large, well-capitalised issuers whose securities trade in liquid markets and for which they can turn over their initial capital many more times than for less liquid stocks, capturing the spread more often with lower risk. These strategies are largely replicable across different stocks and also scalable across asset classes employing a LOB methodology. Generally, AMM algorithms can require a relatively small subset of market microstructure features for input into their algorithms, though the complexity may increase when the firm's specific risks and inventory management processes are incorporated into the algorithm.

AMM strategies profit by capturing the bid-ask spread which compensates the MM for the explicit and implicit transaction costs incurred in executing this strategy. The three primary cost components of an AMM strategy are the premium that risk-averse MM traders require due to *inventory risk* of holding a non-zero net position in a particular security, the *explicit costs* charged by trading venues in the form of fees from placing and executing orders on the exchange and the cost incurred from the *risk of adverse selection* of one of the MM's orders once their limit order is submitted to the exchange (Menkveld, 2013). Inventory risk represents the potential loss incurred by the AMM from liquidating their holdings when the price moves beyond an acceptable threshold defined in their risk management settings due to short-term natural market movements. Implicit in inventory risk is the opportunity cost incurred by AMMs from missing profitable trading opportunities while waiting for their limit orders to be executed. Explicit costs from trading on an exchange are generally known beforehand and factored into the MM algorithm. Risk from adverse selection between the AMM and other trading participants emanates from the divergence of information sets between the two traders (Madhaven, 2000). The risk of adverse selection is positively correlated with the length of a limit orders placement on the order book of a trading venue. Thus, the risk of being adversely selected or the price moving in an uncongenial direction decreases for orders with minimal time duration the order book, a fact that motivates an AMM traders speed of reassessing and repricing their orders (Aldridge, 2013).

There are several strands of thought on modelling AMM trading strategies – including naïve inventory models, inventory-driven models and information-driven models. Naïve Inventory models involve basic strategies where AMMs place double-sided quotes with fixed offset from the BBO. This offset may vary with the level of volatility, aggressiveness or informed trading observed on the LOB. Inventory models factor in inventory risk and opportunity costs when optimising AMM strategies (Menkveld, 2013; Ait-Sahalia, 2017). Information models take a different angle by focussing on AMMs minimising the probability of being adversely selected. The following section introduces each model before using these models to derive relevant LOB and trader-specific features that may be used in the algorithms of AT firms conducting AMM strategies.

2.3.1.1 Naïve Models

AMM strategies based on naïve inventory models can be implemented with very basic algorithms that provide a set of rules on how to position a traders' double-sided quotes based on metrics derived from input features relating to the price, volatility, level of aggressiveness and distributions of orders in the LOB (Aldridge, 2013). *Fixed Offset strategies* maintain passive and continuous bid and ask limit orders k ticks from the current BBO. The proximity of the orders to the inside bid-ask spread, a function of k, determines the probability of execution of the AMM's orders. *Volatility-Dependent Offset strategies* vary the limit order placement strategy of the AMM depending on the level of volatility, σ, of the security's price, as lower volatility environments favour a reduction in k such that orders are placed closer to the BBO and thus more likely to be executed. *Frequency of Aggressive Order-Dependent Offset strategies* adjust the order price depending on the average arrival rate of aggressive orders, μ (Avellaneda, 2008).

Several LOB and trader-dependent features are built to model this strategy. Fixed quote offset metrics for the bid, QOB_t^x, and ask, QOA_t^x, side of the LOB measure the percentage difference between the current bid and ask quote offset for trader x, $CQOB_t^x$ and $CQOA_t^x$, and their time-weighted average quote-offset from the BBO over period k, $TQOB_t^x(k)$ and $TQOA_t^x(k)$. Volatility normalised quote offset features for the bid, $QOVB_t^x$, and ask, $QOVA_t^x$, are built by normalising the current quote offset by k-period volatility, σ, amplified by a z of 3 standard deviations. Similarly, aggressiveness normalised quote offset metrics, $QOAB_t^x$ and $QOAA_t^x$, are normalised by the k-period volatility, σ, scaled by a z of 1%. Several features relevant to AMM quote-offset strategies are derived from the datasets used in this thesis. They are then used as inputs into the machine learning algorithms employed to predict the next action a trader will execute. For example, an AMM executing a fixed offset strategy will place orders closer to the BBO when their fixed offset metrics move significantly from zero. Quote-offset features include:

$$QOB_t^x = \frac{CQOB_t^x - TQOB_t^x(k)}{TQOB_t^x(k)} \qquad CQOB_t^x = BBP_t^m - BBP_t^x \qquad TQOB_t^x = \frac{\sum_{i=1}^{k} CQOB_{t-i}^x}{k}$$

$$QOA_t^x = \frac{CQOA_t^x - TQOA_t^x(k)}{TQOA_t^x(k)} \qquad CQOA_t^x = BAP_t^x - BAP_t^m \qquad TQOA_t^x = \frac{\sum_{i=1}^{k} CQOA_{t-i}^x}{k}$$

$$QOVB_t^x = \frac{[CQOB_t^x + z\sigma(k)] - TQOB_t^x}{TQOB_t^x(k)} \qquad QOVA_t^x = \frac{[CQOA_t^x + z\sigma(k)] - TQOA_t^x}{TQOA_t^x(k)}$$

$$QOAB_t^x = \frac{[CQOB_t^x + za(k)] - TQOB_t^x(k)}{TQOB_t^x(k)} \qquad QOAA_t^x = \frac{[CQOA_t^x + za(k)] - TQOA_t^x(k)}{TQOA_t^x(k)}$$

2.3.1.2 Inventory-driven Models

Inventory-driven models of AMM strategies provide a framework for modelling how traders minimise inventory risk given a defined level of risk tolerance of holding non-optimal positions, informing the strategies optimal inventory management and trade execution policies. The classical inventory control problem was derived to model the behaviour of 'traditional' market makers in an environment where liquidity takers and providers both have informational advantages. Market makers are able to view the LOB, while liquidity takers are potentially more informed regarding the intrinsic asset price (Amihud, 1980; Gloston, 1985). Recent research on inventory-based models has focused on how AMM strategies are applied in high-frequency markets where certain AT firms have informational and speed advantages over low-frequency traders (Menkveld, 2013; Ait-Sahalia, 2017).

Menkveld (2013) decomposes AMM strategies into the 'operation' that generates the gross trading profit and the capital that is required to execute the strategy as a way of solving for maximum AMM profitability:

$$\bar{\pi} = \frac{1}{T}\sum_{t=1}^{T} n_{t-1}\Delta p_t - |\Delta n_t^a| p_t(s_t + \tau^\alpha) + |\Delta n_t^p| p_t(s_t + \tau^p)$$

Where $\bar{\pi}$ is gross profit for each unit of time after trading fees, n_t^a and n_t^p are the respective cumulative aggressive and passive trades through time *t,* such that the net position at time t is, $n_t = n_t^a + n_t^p$. In this case, p_t is the transaction price, s_t is the relative effective half-spread, while τ^α and τ^p are the proportional exchange and clearing fees for aggressive and passive orders respectively. The first term of the profit equation encapsulates the realised 'positioning' component of profit which is dependent on whether the AMM conducts speculative trading, is adversely selected on its quotes (Glosten, 1985), or is willing to lose in order to mean-revert out of a non-zero inventory position. The last two terms refer to the paid effective spread for aggressive orders and the earned effective spread for passive orders, both accounting for fees and rebates from trading exchanges (Menkveld, 2013). Overall and Menkveld (2013) find that the HFT AMMs make money on the net spread but lose money on its inventory positions.

Ait-Sahalia (2017) provides another modern interpretation of the classical AMM inventory control optimisation problem by including new features prevalent in high-frequency markets and incorporating the assumption that AT firms hold microstructure-based informational and systematic speed advantages over low-frequency traders (LFT). HFT AMMs are limited to submitting and cancelling only limit orders at the BBO or one tick from the BBO with non-strategic fixed quantities placed on LOBs with no liquidity rebates. The authors model the optimal quoting policy for an HFT AMM, π^*, using an objective profit function comprising various traders' arrival rates, microstructural signals predicted by the HFT, HFT inventory variables and other fixed hyper-parameters exogenous to traders' behaviour. Patient LFTs', impatient LFTs' and arbitrageurs' arrival rates are a Poisson process with variables, λ_P, λ_I, and λ_A, respectively. HFTs are only able to modify their quotes under a random Poisson distribution with arrival rate of μ, over which they have no control, but rather depends on the dynamics of the LOB. The first microstructural signal incorporated in the model is a Bernoulli random variable that takes either a buy or sell value and predicts the direction of the LFTs next market order with commensurate accuracy probability p. The second signal is another i.i.d Bernoulli random variable that predicts the type of LFT, patient or impatient, that submits the next order with accuracy probability q. These signals change

and are drawn from a new unconditional distribution when either a new signal is received or an LFT order is placed, which exists on a Poisson scale θ and λ, respectively. HFTs net inventory position at t is x_t with an inventory risk aversion factor, Γ, that provides a penalty against holding net inventory and represents an important consideration for risk mitigation within the AMM algorithm. HFTs with net long or short positions may change their distribution so that they do not provide liquidity on the side they have the net position on as a means of drawing down inventory and minimising inventory risk. Additional terms include the stocks tick size, C, the relevant discount rate, D, and an arrival rate term, σ, that counts the price jumps and represents price volatility.

The development of inventory-driven models of market making provide a series of features that may be relevant for application in a machine learning model of trader behaviour. First, a net inventory ratio feature, $INVN_t^x$, is computed by measuring the ratio between cumulative net volume and total volume traded by trader x over the trading day. Second, a relative inventory position feature, $INVR_t^x$, is built to measure the deviation of the current net inventory position from a proxy target ratio, $INVT_t^x$, which assesses in what position a trader ends the day across all UK equity venues. Third, effective half-spread measures for passive and aggressive orders, $ESPT_t^x$ and $ESAT_t^x$, relate to the volume-weighted relative price difference between the trade received by a trader x for an execution, P_{t-i}^x, and midpoint at trade, MP_{t-i}^x. Fourth, trading cost features for passive and aggressive traders, $CPAS_t^x$ and $CAGG_t^x$, measure the net trading costs for trader x when factoring in effective spreads, rebates, REB_t^x, and exchange fees, EC_t^x. Fifth, inventory risk aversion factor, $IRAF_t^x$, is a trader-dependent feature that proxies for the penalty imposed by traders for holding a non-zero net inventory position. It measures the traders' willingness to make a loss in order to mean-revert out of a non-zero position. Inventory-related features can be defined as:

$$INVN_t^x = \frac{\sum_{i=0}^{t}\left(V_i^{x,B} - V_i^{x,S}\right)}{\sum_{i=0}^{t}\left(V_i^{x,B} + V_i^{x,S}\right)} \qquad INVR_t^x = \frac{INVN_t^x - INVT_t^x}{INVT_t^x} \qquad INVT_t^x = \frac{\sum_{i=0}^{T}\left(V_i^{x,B} - V_i^{x,S}\right)}{\sum_{i=0}^{T}\left(V_i^{x,B} + V_i^{x,S}\right)}$$

$$ESPT_t^x = \frac{\sum_{i=0}^{T}\frac{\left|P_{T-i}^x - MP_{T-i}^x\right|}{MP_{T-i}^x}V_{T-i}^x}{\sum_{i=0}^{T}V_{T-i}^x} \qquad ESAT_t^x = \frac{-\sum_{i=0}^{T}\frac{\left|P_{T-i}^x - MP_{T-i}^x\right|}{MP_{T-i}^x}V_{T-i}^x}{\sum_{i=0}^{T}V_{T-i}^x}$$

$$CPAS_t^x = ESPT_t^x(k) + REB_t \qquad CAGG_t^x = ESAT_t^x(k) + EC_t$$

2.3.1.3 Information-driven Models

Information-driven models of AMM strategies focus on the traders' objective of minimising the risk of adverse selection by building the dichotomy between informed trades and noise trades into the model, providing the trader with the ability to reduce the probability of being adversely selected by the better-informed trader. These models can alleviate the risk of adverse selection by including signals that proxy for the presence of well-informed traders on the LOB based on trading patterns, the shape of the LOB and order-flow autocorrelation, informing the optimal AMM strategy.

Carmona (2012) develops an AMM optimisation solution in which the AMM strategy appears endogenously based on the market views of the agents with which the AMM trades. Agents are differentiated by both the timescale used to determine the expected discounted future price of the security based on their market views, referred to as the agents implied alpha, and the variance of errors

in estimating the future discounted price. The model identifies AMM traders as purely passive liquidity providers that profit through spread and rebate capturing, and have no view on the market, but rather base their profitability function on the implied alpha of the agent with which they transact. Their framework first identifies the market view of agents through a robust model that solves the agent control problem:

$$\alpha_t = E_P\left[\int_0^\infty e^{-\beta(s-t)} dp_s | \tilde{F}_t\right]$$

where α_t is the client's 'implied alpha', the difference between the current price of p and the projected future value, using the individual client's probability measure, P. Filtration, $\tilde{F}_t$. $e^{-\beta(s-t)}$, represents the volatility of the estimation and, dp_s, represents the drift of the implied alpha, which is a function of self-correlation over the timescale β^{-1} and the last price variation to re-centre the estimation. Clients can be differentiated by the variance of the errors they make estimating prices and the time scales of implied alphas.

The model next develops a procedure to infer AMM actions based on how their clients 'view' the market. A profitability optimisation function is then estimated that determines the pricing strategy and thus, the liquidity curve associated with the AMM trading strategy. Incorporated into the model is a penalising term that accounts for possible feedback from clients whose market views were impacted by the AMM's decisions. The solution of the AMM optimisation control problem is:

$$g_t(\alpha) = \frac{e^{m_t(\alpha)/\epsilon}}{\int e^{m_t(\alpha)\epsilon} d\mu_t(\alpha)}$$

Hence, the AMM aims to maximise $g_t(\alpha)$, a measure of how AMMs' order distributions should change relative to the distribution of clients' alphas. In this case, $m_t(\alpha)$ represents the profitability function of the AMM and ϵ represents the feedback effect of how new information extracted from an AMM action impacts the order book. Drift term, $d\mu_t(\alpha)$, represents the drift of expected profitability. In conjunction with the client control optimisation problem, solving the AMM optimisation problem enables the construction of a profitability curve to determine what section of the order book has the highest profitability. This analysis informs what updates the AMM should make to their order distributions.

Cartea (2018) develops an optimal order placement information-based model for AMMs trading on LOBs that incorporate into the strategy the effect of microstructural signals on the LOB, in particular, signed trade order flow autocorrelation, $OFS_t(k)$, and volume order flow autocorrelation, $OFV_t(k)$. Signals are measured over k time periods. Information on price dynamics and information asymmetries is expressed in the form of a 'short-term alpha' measure that is derived as a function of the signed net order flow causing price drift. This phenomenon motivates AMMs to adjust their strategy by potentially withdrawing from one side of the LOB to take advantage of both inventory appreciation when the mid-price changes, as predicted from the order flow, and to minimise the risk of adverse selection from informed traders, as proxied by the level of asymmetric information in the market given the trade order flow autocorrelations.

Two adverse selection LOB features, introduced below, proxy for the prevalence of informed trading as derived from information-driven models of AMMs and can be incorporated into these firms trading algorithms.

Probability of Informed Trading

Probability of Informed Trading (PIN) is a measure of the probability that one's counterparty to a trade possesses asymmetric superior information (Easley, 1996). The presence of informed trading on the LOB creates an adverse selection risk for firms transacting against those with superior information. The PIN metric is derived from a model of strategic interactions on the LOB between a heterogeneous population of informed and uninformed traders with different information sets. Informed traders have access to privileged information events, such as an analyst report, that impacts the value of an asset. These events are independently distributed and occur during a certain period with probability α, with the probability that the event has a positive impact on the asset's value, β, or a negative impact on value, $1-\beta$. Informed traders can accurately estimate the impact of the information on the asset price by updating their Bayesian beliefs regarding the new information. This information diffuses into the LOB as they place trades on the side of the market which the price is expected to move towards at a rate of μ. Uninformed traders do not observe these information events but rather trade off public information, such as company announcements, placing orders on both the bid and ask sides of the LOB at a rate of θ_B and θ_S, respectively. Both the informed and uninformed order arrival rates are modelled as independent Poisson processes with constant intensities.

Two PIN metrics are considered. The traditional PIN metric, $PINT_t$, is based on transactions. An extension of the metric, $PINM_t$, is developed based on limit orders, which is necessary to consider given the high-frequency nature of modern LOBs. Optimisation of the PIN algorithm is performed using hierarchical agglomerative clustering (HAC) initialisation of parameter values (Ersan, 2016) and the Lin-Ke (LK) likelihood factorisation specification (Lin, 2011). The level of asymmetric information in the LOB is measured as the unconditional probability that informed traders enter the market and transact in an asset at time t, that is, the point when an information event occurs. Heuristically, PIN measures the ratio of expected number of transactions or BBO orders, depending on the metric, driven by private information to the total number of expected transactions or BBO orders. PIN metrics are measured over the past k time periods, set at 10, 50, and 100 minutes. PIN features are defined for trades and orders as follows:

$$PINT_t = \frac{\alpha_t^T \mu^T}{\alpha_t^T \mu^T + \theta_B^T + \theta_S^T} \qquad PINM_t = \frac{\alpha_t^O \mu^O}{\alpha_t^O \mu^O + \theta_B^O + \theta_S^O}$$

$$where \quad a_t, \beta \in [0,1] \ , \quad \mu, \theta \in [0, \infty)^2, \qquad for \quad \forall k$$

Volume-synchronised probability of informed trading (VPIN) adapts PIN to the high-frequency nature of modern markets which are increasingly based on volume clocks by measuring the real-time level of informed trading, an indicator for order flow toxicity, in a given security (Easley, 2012). VPIN takes a value between zero and one with values closer to zero reflecting a well-balanced market for the security, while higher values closer to one indicate a significant imbalance in the market and a higher probability of order flow toxicity. High order flow toxicity is manifest in securities with high VPIN levels and can serve as a proxy for the probability that liquidity is fleeting in nature and may evaporate causing significant market volatility. VPINs model information events using volume clocks where events occur regularly after a certain volume of shares are transacted, rather than chronological clocks as is the case with PIN. In volume time, trades executed are segmented into multi-bar volume 'buckets' of equal size, v, with each bucket broken into smaller volume 'bars' with the ith bar in the lth bucket having volume

$V_{i,l}$. The order flow trade imbalances for each of the j bars are calculated as the absolute value of the difference between the buy, $V_{i,l}^{B}$, and sell, $V_{i,l}^{S}$, volumes for bar i in bucket l. The absolute proportional imbalance for bucket l, OFI_l, is defined as the aggregated order flow trade imbalances for each of the individual bars in the bucket, divided by the bucket's total volume:

$$OFI_l(v,j) = \frac{1}{v}\sum_{i=1}^{j}\left|V_{i,l}^{S} - V_{i,l}^{B}\right| \qquad V_l^S = V_{1,l}^S + \cdots + V_{j,l}^S \quad V_l^B = V_{1,l}^B + \cdots + V_{j,l}^B \qquad v = V_l^S + V_l^B$$

Several alternative trade classification schemes have been devised for measuring the order flow trade imbalance between buys and sells, and consequently the VPIN metric. The first two approaches take a holistic view of aggregate order flow across each bar which Easley (2012) argues can cause significant errors in high-frequency environments, while the third approach classifies each trade individually. Based on the 'tick rule' (TR) developed by Easley (2011), the total volume for individual bar $V_{i,l}$ is attributed as buys when the price change, $\Delta P_{i,l}$, between the bars closing, $P_{i,l}$, and opening trade price, $P_{i-1,l}$, is positive and a sell bar when the price change is negative. Balanced price changes when the opening equals the closing price of the bar are determined to have the same trade classification as the previous bar. Easley (2012) introduces the Bulk Volume Classification (BVC) strategy for trade classification that proportions a bars volume as both buys and sells according to a cumulative normal distribution function, ϕ, of the volume bars price change, $\Delta P_{i,l}$, normalised by the standard deviation, $\sigma_{\Delta P_l}$, of trade price changes between adjacent bars for the sample bucket. A qualitatively different approach taken by Andersen (2013) utilises LOB data information to classify the 'true' initiator (TI) of each individual trade based on which trader acts aggressively for each transaction. If a trade involves a buyer placing an aggressive market or limit order that matches against a resting passive sell order, then the trade is classified as buyer initiated and the volume is counted toward buy volumes within the bar. Classifying the volume using this approach requires the summation of volume for all n trades, with volume for the mth trade in the bar given as $V_{i,l,m}$, depending on whether the trade is initiated aggressively by the buyer or seller. VPIN is measured at time t as the rolling average of order flow trade imbalances for the preceding k buckets. VPIN features are defined for buys using the TR, $V_{i,l}^{B,TR}$, BVC, $V_{i,l}^{B,BVC}$, and TI, $V_{i,l}^{B,TI}$, volume calculation methodology as follows:

$$VPIN_t(v,j,k) = \frac{1}{k}\sum_{l=1}^{k} OFI_l\,(v,j)$$

$$where\ V_{i,l}^{B,TR} = \begin{cases} V_{i,l} & if\ \Delta P > 0 \\ 0 & if\ \Delta P_{i,l} < 0 \end{cases}, \qquad V_{i,l}^{B,BVC} = \phi\left(\frac{\Delta P_{i,l}}{\sigma_{\Delta P_l}}\right), \qquad V_{i,l}^{B,TI} = \sum_{m=1}^{n} V_{i,l,m}^{B}$$

$$V_{i,l}^{S} = v - V_{i,l}^{B}$$

Both PIN and VPIN metrics are used in the machine learning models developed in this thesis. They serve as features to describe the current state of the LOB, that is the level of informed trading in the market and are implemented in the models to help predict the actions of traders conducting an AMM strategy.

2.3.2 Microstructural Strategies

Microstructural strategies are generally executed by proprietary trading firms and are designed to profit from the state of the market microstructure and trading environment itself. Proprietary trading firms are characterised by cutting-edge technological algorithmic trading systems with ultra-low latency, co-located access to market data feeds and trading venue matching engines that give them speed advantages during strategic interactions with other traders. Further, these firms tend to trade based on volume clocks rather than chronological clocks traditionally used by execution algorithms (Aldridge, 2013). Microstructural strategies are deployed by firms that primarily trade off microstructural signals that derive naturally from the state and transition of the LOB, microstructure and other micro-period financial metrics.

AT firms commonly interpolate microstructural signals within their algorithms. Microstructural signals emanate from the inherent LOB dynamics of the trading venue or multiple venues with which the algorithm interacts, in the context of the exchange trading rules, market regulations and technological constraints. AT firms that trade off microstructural signals are analogous to high-frequency traders as their strategy is based on micro-time period microstructural phenomena rather than longer-run information regarding the fundamental asset price. Microstructural algorithms are generally modelled with a response function, $R(I_t, O_t, \theta)$, at time t, which generates a trading action based on a current indicator function value, I_t, the individual traders' strategy parameter, θ, and the traders' current order distribution, O_t. The indicator function, $I(t, V_t, \dots, V_0, \varphi)$, is a dynamic measure composed of the values of the current, V_t, and lagged microstructural signals explained below, and a model-specific strategy parameter, φ. We derive ultra-high frequency microstructural signals that are likely to impact price dynamics and hence, the behavioural characteristics and actions of firms executing these types of strategies. Microstructural signals expressed below have been derived from academic literature and economic fundamentals. These signals have been shown to have a relationship with expected price movements, levels of liquidity, volatility, or other market metrics, all which can be used to model predictions regarding the actions and behaviour of trading firms.

2.3.2.1 Order Flow Autocorrelation

Signed transaction autocorrelation signals, or order flows, occur when market order signs have the statistical property of serial autocorrelation as they tend to persistently occur in the same direction over sequential trade events. When a trading exchange receives a buyer or seller-initiated market order, the order flow of subsequent market orders tends to be driven by additional buyer- or seller-initiated market orders, respectively (Biais, 1995). Microstructural strategies seek to extract order flow signals from high-frequency data and incorporate this information into their algorithms. Persistent order flow inculcates information into these algorithms as the behaviour ingrained in the irrevocable execution of market orders reflects the traders' fundamental valuations of these securities. This phenomenon of signed order flow autocorrelation has been shown to persist over both short-time lag periods (Axioglou, 2011) and longer periods of days (Lillo, 2004). Several hypotheses regarding this behaviour have focused on order flow resulting from 'order splitting' by strategic brokers with private information (Lillo, 2005), or due to agent-based cross-broker 'herding' (LeBaron, 2008; Toth, 2015) caused by the slow diffusion of information into prices due to delayed reaction to public or slow reaction to private information. Toth

(2015) uses broker level data on the LSE to show that persistent order flow on short timescales, less than an hour, is dominated by auto-correlated order splitting strategies by individual brokers rather than herding order flow from interactions between many brokers. Signed transaction autocorrelation features are built using the order flow sign, $OFS_t(k)$, and the order flow volume, $OFV_t(k)$, as an average function of the aggregate sign, S_i, and respective volume, V_i, of the past $k \in 1, 5, 10, 50, 100$ trades. Additionally, an order flow volume exponentially moving average (EMA) feature is developed with a degree of weighting decrease, $a \in 1, 3, 7$. Order flow features are defined as:

$$OFS_t(k) = \sum_{i=1}^{k-1} S_{t-i}$$

$$OFV_t(k) = \frac{\sum_{i=1}^{k-1} S_{t-i} V_{t-i}}{\sum_{i=1}^{k-1} V_{t-i}}$$

$$OFVE_t(a) = \frac{2(OFV_t) + (a-1)(OFVE_{t-1})}{a+1}$$

2.3.2.2 Aggressiveness Indicators

Order aggressiveness signals that measure the level of aggressiveness of traders on the LOB are relevant components of a microstructural-based AT strategy's algorithm. Aggressiveness is measured by the proportion of market orders to limit orders on the LOB over a specific period, k, and provides information on traders' expectations of future price movements (Pasquariello, 2007). Aggressive market orders are executed immediately against passive resting limit orders and represent traders seeking to transact quickly to impound information or trading gains at the cost of potentially adverse prices, while limit orders are submitted to attain better prices but with the risk of not being executed. Hence, market orders indicate the presence of private information or informed trading, while the speed of execution and the level of aggressiveness serve as a proxy predictor for the direction of future price changes (Pasquariello, 2007). Aggressiveness level features look at the count, $AGGC_t(k)$, and volume, $AGGV_t(k)$, of market orders to passive limit orders over the past $k \in 100, 1000, 10000$ events, as a function of the absolute value of the number of orders submitted at the BBO, O_i, trades, S_i, and their respective volume, V_i, during the period. Aggressiveness features are defined as:

$$AGGC_t(k) = \frac{\sum_{i=0}^{k-1} |S_{t-i}|}{\sum_{i=0}^{k-1} |O_{t-i} - S_{t-i}|}$$

$$AGGV_t(k) = \frac{\sum_{i=0}^{k-1} |S_{t-i} V_{t-i}|}{\sum_{i=0}^{k-1} |(O_{t-i} - S_{t-i}) V_{t-i}|}$$

2.3.2.3 Market Depth

Market Depth signals are one of several proxy measures of liquidity on a trading venue's LOB. Depth measures the total absolute quantum of volume at relevant bid and ask price points j ticks from the BBO spread, with higher depth signalling a more liquid market that may be more amenable to aggressive orders. The level of depth on the LOB composed of all buyers and sellers with open limit orders provides an additional microstructural signal for AT firms to model LOB dynamics and price movements. Foucault (2005) provides empirical evidence of a positive statistical relationship between the level of

market depth on the LOB and the expected forecasted price volatility in future periods. As discussed, limit orders do not represent an obligation to transact, rather merely a willingness to trade at a certain price, cancellable at any time before matching. Thus, liquidity in the form of market depth is an amorphous concept that may not materialise in the form of passive liquidity given its evanescent nature, particularly during volatile market conditions. Market depth can be viewed within a dynamical supply and demand framework as AT strategies replenish the LOB with liquidity in the form of limit orders when the associated premium of the strategy is sufficiently profitable. Depth from this perspective represents the dynamics of passive liquidity supply on the LOB that would increase when the cost of supplying liquidity is cheap. The level of liquidity on competing trading venues is also a key signal for AT strategies as it directly impacts the probability of the order getting executed on a venue and hence impacts the optimal order routing strategy. AT firms will likely route liquidity to trading venues with relatively less market depth at the same price point as a competing exchange given that this liquidity will be relatively higher in the queue and more likely to be transacted against. Market depth feature, $MD_t(j,k)$, for a price point j over the period k, is the sum of all non-cancelled limit orders placed at the price point. Related market depth EMA feature, $MDE_t(a)$, is also developed with a degree of weighting decrease, $a \in 1,3,7$. Market depth features are defined as:

$$MD_t(j,k) = \frac{1}{k}\sum_{i=0}^{k-1} V_{t-i}^{j}$$

$$MDE_t(a), = \frac{2(MD_t) + (a-1)(MDE_{t-1})}{a+1}$$

2.3.2.4 Limit Order Imbalance

Limit order imbalance signals are constructed from online high-frequency datasets of LOB dynamics composed of all open limit orders resting on trading venues that are capable of being matched by incoming orders from market participants. The distribution of these orders can be modelled as the 'shape' of the LOB and used to infer microstructural signals as part of an AT strategy. Limit order imbalances between bid and ask orders are a common signal featured in the LOB dynamics and AT strategy literature as they provide simple, numerical and comparable single dimension representations of the LOB order distribution (Sirignano, 2016; Dixon, 2017; Aldridge, 2013). Limit order imbalances are calculated as the difference between the number of shares or depth of orders that traders are willing to buy and sell at listed prices. Common models of LOB order imbalances will take the difference between the depth of bids and asks at their best listed prices. This process is repeated for each price point one tick further from the previous order imbalance until j order imbalances have been calculated at $j-1$ ticks from the best bid and ask prices that comprise the spread. Microstructural signal based AT strategies take order imbalance levels as a manifestation of the market sentiment towards the future price direction of a particular stock and will impound into their algorithm a stronger weight towards taking a long or short position in stocks with positive or negative order imbalance positions, respectively. Studies have found a strong positive relationship in equity securities between the LOB order imbalance and both short-term price changes (Cont, 2013) and daily returns (Chordia, 2004). This follows other studies that find limit orders contain private information, with imbalances, even far from the BBO, evidence towards an investor's valuation once this information diffuses into the market (Berber, 2004). Limit order imbalance, $LOI_t(j,k)$, is measured as the difference between aggregate limit order market depth for buys, $MD_t^{B,j}$,

and sells, $MD_t^{S,j}$, an equal number of $j \in 0, 1, 2, 5$ ticks from their respective best bid and best ask prices, over the past $k \in 0, 10, 100$ events and $1\,ms, 5\,ms, 1\,sec$ time periods. Related market depth EMA feature, $LOIE_t(a)$, is also developed with a degree of weighting decrease, $a \in 1, 3, 7$. Limit order imbalance features are defined as:

$$LOI_t(j,k) = \frac{1}{k}\sum_{i=0}^{k-1}\left(MD_{t-i}^{B,j} - MD_{t-i}^{S,j}\right)$$

$$LOIE_t(a) = \frac{2(LOI_t) + (a-1)(LOIE_{t-1})}{a+1}$$

2.3.2.5 Spreads

Quoted spread signals are measured by the distance between the best bid and ask prices prevailing on the LOB. The tightness of the quoted spread is generally seen as a function of volatility, trading volume (Glosten, 1988), the level of informed trading prevalent in the market (Kim, 1994; Gregoriou, 2005), the degree of competition between market participants supplying and demanding liquidity (Aldridge, 2013) and the interplay between market impact and order liquidity (Sarkissian, 2016), among other factors. For example, wider spreads may represent the presence of asymmetric information as market makers respond to informed liquidity takers by widening their quotes (Ait-Sahalia, 2017). Quoted spreads are the absolute distance between the best ask, P_t^{BA}, and best bid, P_t^{BB}, of passive orders resting on the LOB. Features are defined for the absolute spread over the past k periods, $QS_t(k)$, and the EMA of the quoted spread, $QSE_t(a)$, with, $k = 0, 1, 5, 10, 50, 1000$ events and $1\,ms, 5\,ms, 1\,sec$ time periods, and weighting decrease, $a \in 1, 3, 7$. Quoted spread features are defined as:

$$QS_t(k) = \frac{1}{k}\sum_{i=0}^{k-1} P_{t-i}^{BA} - P_{t-i}^{BB}$$

$$QSE_t(a) = \frac{2(QS_t) + (a-1)(QSE_{t-1})}{a+1}$$

Effective spread signals proxy for the presence of both informational asymmetry between traders and the cost of trading on the LOB (Smith, 1994; Roll, 1984). These spreads proxy for transaction costs as they represent the divergence of the actual trade prices from the midpoint. Further, perfect markets with no informational asymmetry are likely to have effective spreads distributed evenly with mean zero as trades are on average conducted at the midpoint, thus, the size of the effective spread imbalance on one side of the LOB is used as a measure of informational asymmetry. Features for effective spread, $ES_t(k)$, are computed as double the absolute difference between the transaction price, P_{t-i}, and the quoted mid-price, MP_{t-i}, when the trade was executed, divided by the quoted mid-price and weighted by the volume of the trade, V_{t-i}. Effective spreads are measured over the past $k \in 0, 5, 10, 50, 100, 1000$ trade events. Effective spread features are defined as:

$$ES_t(k) = \frac{2\sum_{i=0}^{k-1}\frac{|P_{t-i} - MP_{t-i}|}{MP_{t-i}}V_{t-i}}{\sum_{i=0}^{k-1}V_{t-i}}$$

2.3.2.6 Volatility

Volatility signals measure the dispersion or standard deviation of the stocks logarithmic returns over a defined period k. Realised volatility, $VOLR_t(k)$, takes the standard deviation of the last k logarithmic returns, R_t, based on the opening, P_{t-j}, and closing, P_t prices separated by event or period length j. A mean log return of zero is assumed to negate the need for a Bessel correction. Volatility measures generally require the assumption that security prices follow a random walk (Cootner, 1964) or Brownian Motion, with Parkinson (1980) modelling logarithmic price changes. Parkinson (1980) developed a high-low volatility measure, $VOLHL_t(k)$, that incorporates a logarithmic return measure based on the maximum, P_i^{max}, and minimum, P_i^{min}, prices attained in the period of length j. Features are engineered using event-time periods of $k \in 10, 100, 1000, 10000$ order events, with the attached returns calculated over event window periods of $j \in 1, 1, 5, 5$ events. The theoretical justification for using max-min volatility measures is that when modelling price changes as a random walk, the diffusion constant can be laterally equated with the stocks return variance, with extreme values at the minimum and maximum of a period providing a better estimation for the diffusion constant than opening and closing prices. In this measure, logarithmic returns are scaled by $1/4ln2$ to align with the diffusion constant in random walk models (Parkinson, 1980). Standard volatility features with relevant return equations are defined as:

$$VOLR_t(j,k) = \sqrt{\frac{\sum_{i=0}^{k} R_{t-i}(j)^2}{k}} \qquad R_t(j) = \ln\left(\frac{P_t}{P_{t-j}}\right)$$

$$VOLHL_t(j,k) = \sqrt{\frac{\sum_{i=0}^{k} R_{t-i}(j)^2}{4\ k\ ln2}} \qquad R_t(j) = \ln\left(\frac{P_j^{max}}{P_j^{min}}\right)$$

High-Low-Open-Close (HLOC) volatility, $VOLRS_t(k)$, is another measure that utilises maximum, minimum, open and closing prices, P_i^{max}, P_i^{min}, P_i^{open}, P_i^{close}, respectively, to estimate volatility (Garman, 1980). Features are engineered using event-time periods of $k \in 10, 50, 100, 1000$ order events. The rationale behind this metric is its integration of stable close and opening prices into the volatility metric that are free from bias induced by high-frequency market activity during a specific period analysed. HLOC volatility features are defined as:

$$VOLRS_t(k) = \sqrt{\frac{\sum_{i=0}^{k}\left(\ln\left(\frac{P_i^{max}}{P_i^{min}}\right)\right)\left(\ln\left(\frac{P_i^{max}}{P_i^{open}}\right)\right) + \left(\ln\left(\frac{P_i^{min}}{P_i^{close}}\right)\right)\left(\ln\left(\frac{P_i^{min}}{P_i^{open}}\right)\right)}{k}}$$

Finally, quoted spread volatility, $VOLQS_t$, measures the difference between the maximum quoted spread, $QS_{t\in k}^{max}$, and minimum quoted spread, $QS_{t\in k}^{min}$, over period $k \in 5, 20, 50$ events. Quoted spread volatility features are defined as:

$$VOLQS_t(k) = QS_{t\in k}^{max} - QS_{t\in k}^{min}$$

2.3.3 Momentum Strategies

Momentum strategies impound momentum-based signals computed over micro-time frames into a traders' strategy of how and when to update their order distributions. Traditionally, momentum strategies have focussed on the most effective trade to execute, given movements or acceleration in market dynamics or technical trends. This thesis extricates technical signals from momentum strategies into their own separate category introduced in the next section. Further, momentum and acceleration signals are derived from the high-frequency order and trade dynamics of the LOB, as opposed to traditional methods that only utilise transaction data, due to the fast-paced nature of modern equity markets. Momentum strategies use signals to predict short-term supply-demand price imbalances.

Momentum signals are derived by conducting technical analysis of shifts in micro-period high-frequency LOB dynamics. Momentum features are developed by quantifying price changes over very short periods. Trade momentum features, $MO_t(k)$, compare the current trade price, P_t, versus the trade price k trades previously, P_{t-k}, where the last $k \in 2, 5, 10$ trades are analysed. Additional momentum signals are developed for the best bid, $MOB_t(k)$, and best ask, $MOA_t(k)$, that measure the change in the BBP, P_t^{BBP}, and BAP, P_t^{BAP}, over the past $k \in 5, 10, 100, 1000$ events. Momentum features are defined as:

$$MO_t(k) = P_t - P_{t-k} \qquad MOB_t(k) = P_t^{BBP} - P_{t-k}^{BBP} \qquad MOA_t(k) = P_t^{BAP} - P_{t-k}^{BAP}$$

In a similar manner to momentum features, acceleration features are derived from signals relating to the acceleration of trades, $AC_t(k)$, best bids, $ACB_t(k)$, and best asks, $ACA_t(k)$, which are developed by comparing the change in momentum values for each of the respective features, using k event lags. Acceleration features are defined as:

$$AC_t(k) = MO_t - MO_{t-k} \qquad ACB_t(k) = MOB_t - MOB_{t-k} \qquad ACA_t(k) = MOA_t - MOA_{t-k}$$

Volume signals, $V_t(k)$, quantify the total absolute number of shares traded on the LOB, V_{t-i}, over the past $k \in 5, 10, 50, 100, 1000$ events. Volume features are defined as:

$$V_t(k) = \frac{1}{k}\sum_{i=1}^{k} |V_{t-i}|$$

Latency arbitrage (LA) signals are also developed to determine how algorithmic traders react when there is a degree of mispricing between the limit orders on multiple exchanges resulting in quasi-arbitrage order opportunities. These signals do not relate to the case where a profitable arbitrage opportunity exists. For example, when a stock on LSE is priced cheaper than its comparable listing on BXE, then one could purchase the stock on the LSE and short the stock on BXE until the supply and demand dynamics shift the price in alignment according to the Law of One Price. Rather the LA signals relate to quasi-latency arbitrage opportunities for bid orders, $LAOB_t$, when cross-venue divergences exist between the maximum global BBP across all venues, $\max(BBP_t^y)$, and the current BBP for trading venue y. Similarly, ask order opportunities, $LAOB_t$, measure the difference between the trading venue and global BAP, $\min(BAP_t^y)$. Firms conducting Momentum strategies with advanced automated trading systems are expected to respond to disequilibrium in the best bid and ask prices between venues by placing orders at the BBO of the trading venues with the stale price. Whilst this does not indicate the firm will execute a riskless arbitrage profit, it indicates their speed capabilities in reflecting BBO discrepancies within their algorithms.

2.3.4 Technical Strategies

AT firms that deploy technical strategies accumulate a series of technical indicator signals derived from movements and patterns in LOBs over a period of time and use these signals as instructional inputs into a trading algorithm that decides when to aggressively trade, cancel an order, or update an order distribution. Technical indicator signals can be segregated into price reversion signals that use Bollinger bands, price pattern signals that use a variety of price movements and, oscillator signals and short-term volatility signals, with the common thread that they utilise technical indicators around how the state of the LOB is evolving and fluctuating, to strategically place orders to profit based on predictions of future transitions in the LOB. The primary differentiator between momentum and technical strategies is the time or event period of analysis. Momentum algorithms use micro-time movements in the LOB dynamics to update order distributions, while technical strategies follow short or medium-term fundamental trends or patterns in security prices and LOB dynamics to make predictions about price changes. Generally, traders conducting technical strategies will monitor stocks for abnormal price or volume movements that are likely caused by firm-specific events.

2.3.4.1 Bollinger Bands

Price reversion signals indicate when the current security price has reached a bounded level, such as a Bollinger band, beyond which it is expected to mean revert back to its equilibrium value. Bollinger bands are modelled by the continuance of trends in a stock or correlated market over a short time-frame encapsulated by the level of buying or selling in a signed direction until a resistance level is reached, represented by the Bollinger upper and lower bands, at which point the price is expected to mean-revert. Bollinger bands measure the difference between the current price, P_t , and the upper and lower expected bounds of the price. Technical strategies using price reversion signals are executed by trading in the diametrically opposite direction to the price momentum once the price reaches the Bollinger band, with trading continuing until the price reverts significantly back towards its mean. Generally, traders conducting technical strategies will monitor stocks for abnormal price or volume movements that are likely caused by firm-specific events. This thesis develops two price reversion features for the upper, $BOLU_t(j,k)$, and lower, $BOLL_t(j,k)$, Bollinger bands that ATs executing a technical strategy would potentially code into their algorithm in some form. Features are measured by the excess price, measured in ticks τ, from the current exponential moving average (EMA) price, $AP_t(k)$, plus $j \in 2$ standard deviations, σ, from this reference price. The k-period EMA price calculates a proxy for the average price over the past k events or time periods, however, given the infinite component in the equation, prices since the beginning of the trading day are encapsulated in the measure to some degree. Bollinger band features are defined as:

$$BOLU_t(j,k) = \begin{cases} (P_t - [AP_t(k) + j\sigma])/\tau & if\ P_t > [AP_t(k) + j\sigma] \\ 0\ otherwise \end{cases}$$

$$BOLL_t(j,k) = \begin{cases} ([AP_t(k) + js] - P_t)/\tau & if\ P_t < [AP_t(k) + js] \\ 0\ otherwise \end{cases}$$

$$AP_t(k) = \delta(P_t) + (1-\delta)(P_{t-1}) \qquad \delta = \frac{2}{k+1}$$

2.3.4.2 Moving Average Convergence-Divergence

Moving average convergence/divergence (MACD) are technical indicator signals that compare the percentage difference of the EMA for a stock with short-term lags with the EMA of the same stock using longer-term lag periods. The MACD serves as a technical signal for whether stocks are over or under priced when their short-term prices with lag, k, diverge significantly from the stocks long-term price with lag, j. Higher values of the MACD are indicative of strong current market momentum. Features are built for MACD based on trade prices, $MACD_t(j,k)$, the best bid price, $MACDB_t(j,k)$, and the best ask price, $MACDA_t(j,k)$. Features for the order flow at the BBO adapt technical indicators that have traditionally been used for macro-level analysis of tick price patterns, to the micro-level to find patterns in short-term LOB dynamics. MACD features are defined as:

$$MACD_t(j,k) = \frac{[AP_t(k) - AP_t(j)]}{AP_t(j)} \qquad k > j$$

2.3.4.3 Relative Strength Indicators

Relative strength indicators (RSI) are technical signals that are measured by the ratio of the average EMA during periods of length k when the price increases, P^U, and the average EMA in periods when price decreases, P^D. This signal indicates the relative magnitude of gains to losses in equally sized periods which can be interpreted as whether the market has become overextended in a particular direction. Investors executing technical strategies may have a defined level above or below which traders may view as a signal to buy or sell. Similar to MACD, features of RSI are developed for trades executed, $RSI_t(k)$, best bid prices, $RSIB_t(k)$, and best ask prices, $RSIA_t(k)$. RSI features are defined as:

$$RSI_t(k) = 100 - \frac{AP_t(k, P^U)}{AP_t(k, P^D)}$$

2.3.4.4 Stochastic Oscillators

Stochastic oscillator (SO) is an oscillator signal, $SO_t(k)$, derived from technical analysis that compares the closing price of a stock at time t, P_t, to the highest, $\max(P_t^k)$, and lowest, $\min(P_t^k)$, price traded for that stock over the period from $t-k$ until t, based on a period of length $k \in 5, 10, 15$. One would expect a SO signal close to one to indicate that the price has trended upward with strong momentum, whilst a SO value closer to zero indicates the price trended downwards with strong momentum. Slow stochastic oscillator (SSO) signals are calculated as the average SO over the last $j \in 5$ periods, $SSO_t(j,k)$, and are useful for smoothing out sudden shifts in the SO technical indicator signal. Stochastic oscillator cross-over (SOC) points, $SOC_t(j,k)$, represent transaction signals when the SO and SSO cross-over. Stochastic oscillator features are defined as:

$$SO_t(k) = \frac{P_t - \min(P_t^k)}{\max(P_t^k) - \min(P_t^k)} \qquad SSO_t(j,k) = \frac{1}{j}\sum_{i=1}^{j} SO_{t-i}(k) \qquad SOC_t(j,k) = SO_t(k) - SSO_t(j,k)$$

2.3.4.5 Chaikin Volatility

Chaikin volatility, $VOLC_t(j,k)$, is a volatility signal generally employed in technical strategies to proxy for instances where market prices are highly volatile. The metric measures the percentage change of the current EMA of the breadth between the maximum, $P_{t\in k}^{max}$, and minimum, $P_{t\in k}^{min}$, price over the past k events, $HLA_t(a,k)$, from the previously computed EMA j event lags previously, $HLA_{t-j}(a,k,j)$. This volatility measure compares the spread of high and low prices over time, representing a form of moving average level of volatility. Chaikin volatility provides an interpretation of how unstable a security's price is over small-time frames, with higher values indicating that prices are becoming more volatile. Chaikin volatility features are defined as:

$$VOLC_t(j,k) = \left(\frac{HLA_t(a,k)}{HLA_{t-j}(a,j,k)} - 1\right) * 100$$

$$HLA_t(a,j,k) = \frac{2\left(P_{t\in k}^{max} - P_{t\in k}^{min}\right) + (a-1)(HLA_{t-1})}{a+1}$$

2.3.5 Execution Strategies

Execution algorithms (EA) are deployed by buy-side AT firms to optimise the execution of a large 'parent' order for their client over a specified period. Optimisation requires the management of price impact and price uncertainty through the minimisation of both explicit and implicit trading costs within the temporal and client-specific constraints when completing these orders. Parent orders are not able to be executed immediately against trading venues available liquidity without prohibitive costs, so instead, they are partitioned into smaller ancillary 'child' orders that are routed to multiple trading venue LOBs over a longer period. The time required for execution is dependent on the client's heterogeneous objectives that may vary widely between clients rebalancing an investment portfolio and those taking a directional position. This thesis focuses on execution strategies in the context of 'lit' trading venue LOBs.

Execution algorithms differ from proprietary algorithms identified above as the third-party client decides the stock, volume, trading direction and execution period parameters of the algorithm, and can impose additional boundaries that may require adaptations of the algorithms programmatic methodologies. Further, non-execution AT strategies require automated trading systems with sub-millisecond functionality when responding to changing market states. While execution strategies can operate off longer time-lag distributions, there is growing demand for modern algorithms to be able to comprehend and respond to microstructural signals to improve the execution quality of the algorithm (O'Hara, 2015; Aldridge, 2013).

Automated trading systems' operating execution algorithms extract data in real-time from trading venues that is processed by the algorithm within the constraints of its programmatic instructions to devise an execution schedule. Competitive execution algorithms will pool market liquidity by integrating multiple market access and data feeds from available trading venues into a centralised system from which the execution algorithm operates and routes orders. Different latency rates and fee structures of the trading venues are factored into routing algorithms that navigate a plethora of liquidity across financial markets to achieve a minimal cost execution schedule that meets the investor's risk profile. Performance of the algorithm will compare the execution price to the decision price when the trade was

made or some other benchmark, within the context of risk management and trading cost minimisation. Sophisticated algorithms will encompass genetic components that learn through iterative training based on real-time performance and market conditions that lead to parameters or variables changing within the algorithm.

Investment firms operating in EU jurisdictions are obligated under MiFID II to take 'all sufficient steps'[18] to attain best execution. Further, additional considerations around the 'fairness' of the execution for the client and the inclusion of 'market data' in their measure of best execution, increase the requirements for investment firms.[19] An additional change under MiFID II is the requirement for investment firms to assess best execution in the context of 'total consideration' of cost which incorporates explicit external costs, such as exchange fees and settlement costs, and explicit internal costs, such as the firm's remuneration for executing the order and third-party fees. Under both MiFID and MiFID II, the factors considered by firms when executing their clients' orders and measuring best execution includes the price of the order, costs involved, speed and likelihood of execution and settlement, order size and any other client or external consideration relevant for the orders execution. In addition to meeting the spirit of MiFID II, institutional investors seeking to attain best execution when liquidising a large parent order require execution algorithms employed through an automated system that minimises explicit costs, implicit execution costs, composed of market impact and opportunity costs and minimising slippage or the risk of deviations of the execution price from the decision price.

Explicit execution costs are transparent and factored into the algorithm before trading is conducted. Explicit costs include broker commissions for providing exchange access to clients, broker execution costs for facilitating trades, exchange fees for liquidity and matching services, clearing and settlement fees and taxes from income or capital gains. These trading costs are minimised by executing algorithms that route orders to trading venues with advantageous fee structures.

Implicit execution costs are amorphous in nature and are generally controlled for by minimising market impact when aggressively liquidating large positions, minimising the opportunity cost from non-executed limit orders, maximising execution speed, maximising order size and minimising the bid-ask spread which is paid when trading.

Market impact is a dominant implicit cost that naturally arises from adverse price movements caused by information leaked by traders executing large liquidity-demanding trades relative to the depth available on the LOB. Information is leaked from market order signals and impounded into prices as competitors gain knowledge of the traders' view of the stocks fundamental price and future price movements. Competitors may also derive signals from limit orders though these tend to be less credible with lower informational content given that these orders are merely cancellable indicators of trader expectations, resulting in smaller market impact magnitudes than market orders (Aldridge, 2013; Hautsch, 2012). Impact is viewed externally as a correlation between an incoming market order and subsequent price movements. Market impact can be decomposed into various forms, including its permanent, where prices impound fundamental information, and temporary components, where prices overshoot the fundamental price before decaying to equilibrium. Optimal execution requires control over a trader's market impact by splitting larger parent orders into smaller ancillary trades that apportion

[18] MiFID II (RTS 27/28).
[19] ESMA/2014/1569.

the instantaneous impact from market orders over longer time periods. Execution algorithms attain this objective by optimising the size of orders and speed of submission in a way that provides minimal information leakage of the traders' market views, thus, moving the market price of the stock as little as possible.

Opportunity costs measure the profit foregone from placing non-executed limit orders when the price moves adversely to the strategy and the orders remain unfilled. Modern execution algorithms impound the trade-off between market impact from market orders and opportunity cost from limit orders into an optimisation equation to solve for the optimal trading schedule, statically or dynamically. Maximising execution speed allows firms to trade based on current market states and conditions, and is largely dependent on market liquidity, the firms order flow strategy, market or limit order scheduling, and the AT systems they employ, with latency between the trading decision and submission on the trading venues LOB being the relevant metric. Maximising order size within the optimal execution problem requires consideration of the potential market impact from large trades, the relative size of a client's parent order, the required period of execution and requirements demanded by the client. Certain clients such as pension funds and insurance companies may be careful about minimising execution costs when rebalancing portfolio positions, whilst hedge funds taking directional bets may be more interested in executing large trades quickly with less focus on the cost of trading. Spreads represent an implicit cost for traders seeking immediate and certain execution through aggressive market orders whilst it can be modelled as compensation for traders with delayed execution that gain the spread by placing limit orders.

The third primary cost of execution is slippage, a non-tangible implicit cost of execution that is generally modelled by the investor based on academic literature or proprietary research before being imputed into the algorithm. Slippage costs arise when the market price moves adversely to the decision price when the investor decides to transact in the stock. These costs are greater for firms with higher latency in their AT systems, regardless of whether the latency arises from their internal network architecture or physical bottlenecks in connections with brokers or trading venues. Slippage can be measured as a random distribution inferred from analysis of past trading costs.

Execution algorithms performance is determined based on a relative benchmark, congruous with both client and broker. These algorithms can be classified based on their level of complexity and the type of benchmark they use to attribute performance to the execution broker. Meeting performance metric benchmarks drive broker profits which drive demand for the development of execution strategies that meet these metrics, with algorithms evolving over time, based on the performance metrics used by clients. Participation-based execution algorithms were an initial attempt at managing the slicing and execution of large parent orders to minimise impact from trading large volumes in a speedy manner. These algorithms were benchmarked against basic market generated volume, time or price data independent of the client's order or the LOB situation at the time of the order placement, such as the market Volume-Weighted Average Price (VWAP). Implementation Shortfall (IS) execution algorithms grew from the demand for brokers to control execution costs based on the individual characteristics of a client's order, in particular, the decision price benchmark at which the order arrives at the market. The demand for execution algorithms that manage implicit costs from trading large orders in high-frequency and fragmented markets led to the development of IS benchmarks with market impact and market resiliency components, which have evolved as the theoretical foundations and empirical capabilities of estimating the impact from execution algorithms have changed.

2.3.5.1 Participation Rate Algorithms

Participation rate strategies requisite execution algorithms to partition parent orders into trading schedules executed within a defined period according to relatively passive statistically rooted volume, time or price dependent benchmarks. Participation strategies pre-date the contemporary iteration of high-frequency markets, however, they remain a valid strategy and are still used by some execution brokers. The focus of these algorithms is to trade a certain number of shares that meets a pre-defined benchmark, such as percentage of total volume traded, number of shares traded over a given period, or average volume-weighted price achieved by the execution algorithm. Whilst the following algorithms may have pre-defined instructions, any modern participation algorithm trading in liquid markets will involve some dynamic or random component infused into the algorithm to avoid detection.

Percentage of Volume

Percentage of Volume (POV) algorithms aim to trade a fixed percentage of the expected total market trading volume over a defined period. The level of *volume* traded is the relevant benchmark for POV algorithms to meet. These algorithms will calculate the predicted daily volume turnover based on the activity over previous window periods, which may be short - 30 minutes; or long - several days, and then place child orders in the sequential period based on a pre-defined participation rate of the expected volume to be traded. This process continues by dynamically updating order sizes to trade a fixed percentage of the previous period's volume traded until the parent order is filled. For example, an execution algorithm may have the objective of trading 2% of the daily volume traded in the stock, which is achieved by breaking the days into increments of 10 minutes, continually trading 2% of the previous 10-minute volume, in the subsequent 10-minute interval. The relevant feature for determining whether a trader is executing a POV strategy is to analyse their actions conditional on their current POV imbalance, $POVI_t^x$, which measures the difference between the current POV of trader x , $POVT_t^x$, and their end of period target POV, $POVX_t^x$, based on the volume traded over certain periods, V_{t-i}^x. Traders conducting a POV strategy would be expected to augment their actions and orders placed on the LOB to align their current POV with their end-of-period desired POV during the trading day. POV features are defined as:

$$POVT_t^x = \frac{\sum_{i=0}^{t} V_{t-i}^x}{\sum_{i=0}^{t} V_{t-i}} \qquad POVX_t^x = \frac{\sum_{i=0}^{T} V_{T-i}^x}{\sum_{i=0}^{T} V_{T-i}} \qquad POVI_t^x = POVT_t^x - POVX_t^x$$

Time-Weighted Average Price

Time-Weighted Average Price (TWAP) algorithms slice parent orders of size, X, into N child orders with equally sized portions, X/N, that are submitted to the market uniformly over a total time period, T, in predefined and equally distributed time intervals, T/N, until the total order is executed. These algorithms aim to obtain superior performance over the market TWAP benchmark which is dependent on the investor's decision regarding the trading *time*, T. For example, a TWAP algorithm may receive instructions to trade 10,000 shares over a 2-hour period, then can decide to execute a TWAP strategy by splitting the order into lot sizes of 2,500 shares which are submitted every half an hour. TWAP algorithms aim to meet the $TWAPM_t$ benchmark which is calculated as the time-weighted arithmetic average of the opening, P_i^O, high, P_i^H, low, P_i^L, and closing, P_i^C, prices in periods of time length, k, over the period up to t.

The market TWAP imbalance feature, $TWAPMI_t$, calculates the percentage difference between the market TWAP over k, $TWAPM_t(k)$, and the current trading price, P_t. Additionally, the trader TWAP imbalance feature, $TWAPTI_t^x$, measures the percentage difference between the TWAP for trader x, $TWAPT_t^x$, and the market TWAP over k, $TWAPM_t(k)$. Traders conducting a TWAP strategy with a TWAP imbalance will be expected to execute more passive or aggressive orders depending on the imbalance between the relevant TWAP and price. TWAP features are defined as:

$$TWAPT_t^x(k) = \sum_{k=1}^{t} k(P_k^x) \qquad TWAPM_t(k) = \frac{1}{4t}\sum_{k=1}^{t} k(P_{t-k}^O + P_{t-k}^H + P_{t-k}^L + P_{t-k}^C)$$

$$TWAPTI_t^x = \frac{TWAPT_t^x - TWAPM_t}{TWAPM_t} \qquad TWAPMI_t = \frac{TWAPM_t - P_t}{P_t}$$

TWAP algorithms may incorporate other relevant cost components, such as stock liquidity and volatility profiles. Optimisation of the TWAP algorithm requires the selection of an order size that minimises market impact, whilst being large enough to ensure the execution of the parent order in its entirety.

Volume Weighted Average Price

Volume Weighted Average Price (VWAP) is a common volume participation execution algorithm that controls the order price, P_t, and volume, V_t, of child orders submitted at time t in a way that meets the holistic market VWAP benchmark over the execution period, T (Berkowitz, 1988). VWAP algorithms correlate trading execution with movements in market activity such that the algorithm trades a similar proportion of volume to that of the market during the trading period at a comparable average price, balancing the volume traded with the execution price. Similar to TWAP, trade sizes can be calibrated passively based on a pre-computed schedule using historical data volume profiles or dynamically based on prior period trading activity and intra-day liquidity variations and volume patterns. The objectives of VWAP algorithms is to meet the market $VWAPM_T$ benchmark over the execution period, T, defined as the total value traded, $\sum_{i=1}^{T} P_i V_i$, divided by the total volume of all permissible trades, $\sum_{i=1}^{T} V_i$.

AT firms executing a VWAP execution strategy will attempt to align their current k-period $VWAPT_t^x(k)$ with the moving market $VWAPM_t(k)$ throughout the trading day. Similar to TWAP, we develop features for the trader and market VWAP imbalances as $VWAPTI_t^{x,I}$ and $VWAPMI_t$, respectively. Indicator I reflects whether the trade is a buy or sell. We expect that traders conducting these strategies to fluctuate aggressive and passive orders depending on the relative imbalance. VWAP features are defined as:

$$VWAPT_t^x(k) = \frac{\sum_{k=1}^{t} P_{t-k}^x V_{t-k}^x}{\sum_{k=1}^{t} V_{t-k}^x} \qquad VWAPM_t(k) = \frac{\sum_{k=1}^{t} P_{t-k} V_{t-k}}{\sum_{k=1}^{t} V_{t-k}}$$

$$VWAPTI_t^{x,I}(x) = \frac{VWAPT_t^{x,I} - VWAPM_t}{VWAPM_t} \qquad VWAPMI_t = \frac{VWAPM_t - P_t}{P_t}$$

Additional VWAP features are developed for the absolute and volume-weighted difference between the traders sell, $VWAPT_t^{x,Sell}$, and buy, $VWAPT_t^{x,Buy}$, VWAPs. These features are defined as $VWAPTD_t^x$ for absolute difference and $VWAPTW_t^x$ for volume-weighted difference. Trader VWAP features include:

$$VWAPTD_t^x = VWAPT_t^{x,Sell} - VWAPT_t^{x,Buy}$$

$$VWAPTW_t^x = \frac{\left(VWAPT_t^{x,Sell} V_t^{x,Sell}\right) - \left(VWAPT_t^{x,Buy} V_t^{x,Buy}\right)}{\left(V_t^{x,Sell} + V_t^{x,Buy}\right)}$$

2.3.5.2 Implementation Shortfall, Market Impact & Market Resiliency

Evolution over the past decade in technological capacity, quantitative capabilities and diagnostic trading tools have facilitated the adoption of more complex execution algorithms with the central objective of minimising implementation shortfall (IS). These algorithms have largely replaced traditional participation strategies defined in the above section. IS algorithms extend the framework developed by participation strategies, particularly VWAP, which assume negligible market impact from individual trades and view price as a stochastic martingale sequence of random variables by incorporating implicit costs of trading, including price impact and opportunity costs, when formulating optimal strategies.

Shifting dynamics in how transaction cost analysis (TCA) is employed by investment funds in quantifying the cost of executing trades led to the derivation of the IS formula that has become ubiquitous in measuring the performance of a trading algorithm. Optimisation of the IS execution algorithm requires the selection of a trading strategy that minimises the objective cost function represented by the slippage from a decision price benchmark when trading a fixed volume of shares over a specified period (Perold, 1988; Almgren, 2001). IS can be viewed as the difference between the cost of trading in real-world markets, where one pays a final execution price that includes commissions, frictional and other explicit and implicit costs and the cost of trading in a market with unlimited liquidity where the price attained is the decision price. The benchmark decision price is generally the price of the stock at the time of the parent order arrival when the execution algorithm begins trading or may be decided in agreement between the fund manager client and broker executing the trade. IS benchmarks and slippage are measured ex-post based on actual transactions and dependent on the order size, liquidation time, LOB liquidity, the impact on the LOB from market orders, and other relevant factors.

Generic IS execution algorithms involve the trading of X shares of a stock currently trading at decision price p_0 over N equally spaced time periods. These algorithms aim to minimise an objective cost function, $C(\mathbf{V})$, which in its raw form, with no implicit costs, represents the shortfall between the weighted execution price and the assigned decision price benchmark. The weighted decision price is computed as the linear combination of volume transacted, V_k, and the traders effective price $\tilde{p}_k$, for all periods N. The assigned decision price benchmark equals the parent order size, X multiplied by the decision price, p_0, leading to the objective cost function and optimisation problem:

$$C(\mathbf{V}) = \frac{1}{N}\sum_{k=1}^{N} V_k \tilde{p}_k - X p_0 \qquad \min_{V} \mathbb{E}[C_A(\mathbf{V})]$$

Execution algorithms are generally judged by their ability to minimise implicit, explicit, and market impact transaction costs using a post-trade implementation shortfall (IS) benchmarked transaction cost analysis (TCA) framework. IS is a common benchmark utilised by brokers to judge the effectiveness of a particular execution algorithm or strategy. Sophisticated execution algorithms will perform scheduling of child orders using size, price and order frequency parameters that minimises the market impact of the execution arising from increased liquidity demand whilst ensuring both execution costs and risk are minimised. TCA is a tool employed by brokers to attribute costs of trading related to the traders' demand for liquidity when executing a trade and the costs from the market timing decisions which relates to the impact of price changes over the execution period.

We utilise a common industry framework for TCA that develops intuitive metrics of traders' execution costs using an IS benchmark. These metrics serve as features for real-time inputs into execution algorithms. TCA identifiers for each individual trade benchmarked against IS includes liquidity charges, liquidity premiums, spread charges and timing consequences. Individual liquidity charge (ILC) measures the volume-weighted difference between the current execution price, P_i^E, and the previous execution price, P_{i-1}. ILC represents the 'charge' or cost for executing a trade at period i relative to executing the trade at the previous period, $i-1$. Individual liquidity premium (ILP) measures the difference between the best price on the side of the LOB opposite the trade and the execution price. ILP proxies for the 'premium' paid by the trader for demanding liquidity greater than that available at the best bid-ask market depth. Individual spread charge (ISC) is a metric defined as the minimum function of ILC minus ILP, for trade i, bounded by zero. The cumulative running sum of these three measures provides proxy features of transaction costs, CLC_i, CLP_i, and CSC_i, for the Cumulative Liquidity Charge (CLC), Cumulative Liquidity Premium (CLP) and Cumulative Spread Charge (CSC), respectively. CLC effectively separates the price impact component of IS from the timing execution cost component for a trader employing an execution algorithm. Finally, the timing consequence (TC) is measured as the residual execution, or IS differential, after accounting for the individual traders CLC. Implementation shortfall transaction cost features for each trader x are defined as:

$$ILC_i^x = \begin{cases} V_i^E\left(P_i^E - P_{i-1}\right) & if\ buy \\ V_i^E\left(P_{i-1} - P_i^E\right) & if\ sell \end{cases} \quad ILP_i^x = \begin{cases} \max\left(0, V_i^E\left(P_i^E - P_i^{AP}\right)\right) & if\ buy \\ \max\left(0, V_i^E\left(P_i^{BP} - P_i^E\right)\right) & if\ sell \end{cases} \quad ISC_i^x = \min(ILC_i^x - ILP_i^x, 0)$$

$$CLC_T^x = \sum_{i=1}^{T} ILC_i^x \quad CLP_T^x = \sum_{i=1}^{T} ILP_i^x \quad CSC_T^x = \sum_{i=1}^{T} ISC_i^x \quad IS_T^x = \sum_{i=1}^{T} V_i(P_i - P^A) = CLC_T^x + TC^x$$

Optimising IS-based execution algorithms used to split large orders is necessary for competitive brokers in modern equity markets. These algorithms largely solve for the optimal order schedule strategy by minimising some objective function mathematically formulated through an underlying model of price dynamics with an imputed market impact feedback component that accounts for how price changes are conditioned on order flow. The theoretical foundations of these algorithms derive from the formulation of models that explain the optimal process of splitting large orders to minimise price impact.

Optimal execution can be viewed as a multiscale problem where investors decide how to proportion child orders during macroscopic timing intervals and what strategy one should take to execute these child orders during the period, including what type of orders to use and the optimal level of aggressiveness. The following sections extend the generic IS execution algorithm to incorporate the costs and benefits of completing a large parent order by trading aggressively or passively. Aggressive trading in the form of deploying market orders for execution immediacy serves as the catalyst for detrimental market impact depending on the resiliency of markets, which must be weighed against the use of passive execution that decreases the risk from adverse price fluctuations but leads to higher opportunity costs and price risk from the placement of potentially unexecuted limit orders. Hence, execution algorithms aim to split orders in real-time in a way that minimises market impact of the parent whilst accounting for opportunity cost or timing risk from delaying execution as prices move through the execution process (Johnson, 2010).

The following sections analyse three approaches to modelling IS and execution scheduling that incorporates the concept of market impact. Two other approaches relevant for execution algorithms that model the dynamics of the LOB following liquidity shocks, market resiliency and impulse response functions, are presented thereafter. These models will inform the development of relevant market impact and resiliency features used in this thesis, given that they would likely form a component of trading algorithms for firms conducting execution strategies. The first generation of market impact models segment overall market impact into a permanent and temporary component (Bertsimas, 1998; Almgren, 2001; Almgren, 2005). Transitory Impact Models (TIM) extend these models by highlighting the non-linear transient nature of market impact which is factorised into instantaneous and decay market impact components (Gatheral, 2012; Curato, 2015; Dang, 2014; Taranto, 2016). History Dependent Impact Models (HDIM) deviate from TIM models by positing that only price-changing events should be impounded into a market impact model rather than all events as is the case in the TIM. The final two sections introduce market resiliency models and impulse response functions that take a holistic view of market impact and the level of resiliency in markets by measuring the impact of different events on the LOB dynamics.

Temporary and Permanent Market Impact

Initial price dynamic models decompose the price impact from a market order into its permanent market impact, which pushes the price in a direction that impacts all future trades and temporary market impact, which triggers a price movement with no duration that only effects the trade being executed (Bertsimas, 1998; Almgren, 2001; Alfonsi, 2010).

Bertsimas (1998) derive optimal execution strategies using stochastic dynamic programming that achieves optimal execution by minimising an objective function of execution costs which is modelled to include a price impact component. Almgren (2001) build upon Bertsimas (1998) by incorporating a permanent market impact, a temporary market impact, a spread component and a random noise term into the IS optimal execution function. Price dynamics are modelled as a stochastic process with temporary price impact decaying instantaneously. They model the effective price paid for shares, $\tilde{p}_k$, as a function of the previous price paid, p_{k-1}, the volume traded, V_k, a linear permanent market impact term, θV_k, a linear temporary impact term, ρV_k, a quoted spread term, S, and a random variable drawn from a Gaussian i.i.d distribution with mean zero and unit variance, η_k. Temporary market impact is analogous to the price impact caused by the demand for liquidity from trading, whilst permanent price impact emanates from information leakage arising from the order execution over time. The expected objective function of the optimal execution problem, assuming all trades are placed in the same direction, is given in the form of:

$$\mathbb{E}[C_A(\mathbf{V})] = \frac{\theta}{2}X^2 + (\rho + \theta)\sum_{k=1}^{N} V_k^2 + \theta \sum_{i \neq j} V_i V_j + \frac{X \cdot S}{2} \qquad \arg\min_{V} \mathbb{E}[C_A(\mathbf{V})]$$

With the objective of only minimising transaction costs, the model determines that the optimal strategy is a constant strategy of trading fixed amount $\frac{X}{N}$ for each period, which reduces to a VWAP strategy in the continuous case. Almgren (2001) also consider the traders coefficient of risk aversion, λ, that represents the level they penalise variance over expected cost, to derive an execution algorithm that optimises the volume traded schedule, V^*. This optimisation problem seeks to minimise market impact costs from large aggressive trades, $\mathbb{E}[C_A(\mathbf{V})]$, and opportunity costs from small unexecuted limit orders,

$\lambda Var[C_A(\mathbf{V})]$, by developing an optimal efficient frontier of execution schedules for a series of risk aversion levels. The risk or variance of the objective cost function is the squared difference between the actual and expected cost function. When risk aversion is positive, the optimal efficient frontier of execution schedules is a convex curve that represents the time-dependent trading strategies where expected IS is minimised for a given level of IS variance and trader risk aversion. Given the assumption of linear, fixed and permanent market impact, the strategies on the optimal efficient frontier will meet the condition:

$$\arg\min_{V^*}(\mathbb{E}[C_A(\mathbf{V})] + \lambda Var[C_A(\mathbf{V})]) \qquad \lambda Var[C_A(\mathbf{V})] = \sigma^2 \int_0^N V_k^2 \, dk$$

These optimal execution strategies incorporate temporary and permanent market impact terms into their model of price dynamics. Solving for the optimal strategy requires a transformation of the frontier into a Euler-Lagrange equation, which depends on the trading firms' level of risk aversion, λ, such that:

$$V_k = X\frac{\sinh(\beta(N-k))}{\sinh(\beta N)} \qquad where\ \beta = \sqrt{\frac{\lambda\sigma^2}{\eta}}$$

Evidently, trading firms with low levels of risk aversion will tend to load their trading schedule with less trades at the beginning of the execution period to take advantage of potential positive changes in the execution price. However, a risk neutral trader will trade a fixed volume of shares throughout the period.

Almgren (2005) develop a model to estimate the temporary, $PITA_t$, permanent, $PIPA_t$, and realised, $PIRA_t$, market impact features for the whole market based on large-cap US equity stocks. These price impact features are analogous to variables θ and ρ developed in the Almgren's (2001) IS minimisation function. Building from Almgren (2005), we develop a framework for estimating permanent, temporary and realised price impact features for each individual trader. Transactions require an aggressive counterparty to each trade that submits a market or limit order which crosses a resting passive order. These transactions invariably induce market impact. All orders within the period being analysed are included. All three price impact measures assume an exponent functional form and are expressed in terms of the total volume executed, X, the volume duration over which the order is liquidated, T, with trade rate, X/T, the 5-day volume traded moving average, V, the sign of the execution order, $sgn(X)$, noise term, η, and a normalisation factor calculated as the intraday standard deviation, σ. Almgren (2005) incorporates a liquidity factor into their model to control for the liquidity characteristics of the stock being traded. They also use the inverse turnover, calculated as a ratio of the number of shares of the firm outstanding, ϕ, to the firms' volume traded each day, V, to control for activity levels. Permanent price impact, temporary price impact and resiliency features can be defined by estimating the regressions:

$$Observables:$$

$$PIPA_t(k) = \frac{P_{t+k} - P_t}{P_t} \qquad PIRA_t(k) = \frac{\bar{P} - P_t}{P_t}$$

$$Regressions:$$

$$PIPA_t = \gamma\sigma T sgn(X)\left|\frac{X}{VT}\right|^{\alpha}\left(\frac{\phi}{V}\right)^{\delta} + \eta \qquad PIPAX_t^x = \gamma_x\sigma T sgn(X)\left|\frac{X}{VT}\right|^{\alpha}\left(\frac{\phi}{V}\right)^{\delta} + \eta$$

$$PIRA_t = \frac{PIPA_t}{2} + \psi\sigma T sgn(X)\left|\frac{X}{VT}\right|^{\beta} + \eta \qquad PIRAX_t^x = \frac{PIPA_t^x}{2} + \psi_x\sigma T sgn(X)\left|\frac{X}{VT}\right|^{\beta} + \eta$$

$$PITA_t = PIRA_t - PIPA_t \qquad PITAX_t^x = PIRAX_t^x - PIPAX_t^x$$

Permanent price impact, $PIPA_t$, includes estimated exponents α and δ, with a parameter γ estimated by a linear regression using all orders submitted through the trading day. Coefficient γ is estimated using a regression model that incorporates observable variables of the permanent market impact as the percentage change between the transaction price at t, P_t, and the future post-trade price, P_{t+k}, that captures the impact from the trade once temporary liquidity effects dissipate. Similarly, realised price impact, $PIRA_t$, is a function of the permanent price impact, $PIPA_t$, and an estimated exponent variable, β, whilst the coefficient ψ is also estimated using a linear regression, based on the percentage change between the transaction price at t, P_t, and the volume-weighted average price, $\bar{P}^{-}$. Regressions are performed using the heteroskedastic weighting of the impact expressions. Temporary market impact, $PITA_t$, is not directly observable but rather is computed as the difference between realised and permanent market impact. The universal coefficients of market impact, γ and ψ apply to all orders over the trading period. Hyper-parameters α, β and δ are estimated using a Marquardt-Nash optimisation algorithm to determine the values of the hyper-parameters that minimises residuals. The gamma variable, γ, which is estimated in the permanent price impact regression, is defined as an additional feature for permanent price impact, $PIGA_t$.

We develop additional analogous features for the linear market impact estimated for each individual trader x, including the trader's temporary, $PITAX_t^x$, permanent, $PIPAX_t^x$, and realised, $PIRAX_t^x$, price impact using Almgren's (2005) methodology. Further, the features we develop in this section for both holistic market impact and individual trader impact are estimated on a rolling basis using a rolling regression of periods of $k \in 10, 20, 30$ minutes. Developing features of linear market impact is relevant for modelling how specific traders react to changes in market impact values over short-term periods in terms of updating their order distribution and interacting with the LOB.

We also utilise a data-driven Generalised Method of Moment (GMM) model to provide comparative features for permanent market impact, $PIPRR_t$, and temporary market impact, $PITRR_t$, parameters for each stock. In addition, the GMM model estimates the feature of LOB resiliency, $RESRR_t$. First, an l-order auto-regression (AR) model of the signed order imbalance volume in period k, OI_k, is fitted with the residual error term q_k imputed to memory. Signed order imbalance at k is taken as the difference between the buyer and seller-initiated volume in the past $l \in 5$ periods, normalised by the daily signed order volume average and standard deviation over the period. Second, the market impact and resiliency parameters are estimated using a statistical GMM model of the price change from the opening midpoint, MP_0, to the current price, P_k, over j lagged intervals with individual lengths of 5 seconds. The current price P_k is taken as a multivariate random variable with the assumption that the price is generated by an ergodic stationary stochastic process. The regression to estimate the market impact and resiliency coefficients uses five second bucket periods, with the regression modelled over $k \in 10$ minute intervals, and rolling regression performed over a $j \in 40$ minute period. The regression equation for the GMM model can be defined as:

$$P_k - MP_0 = \vartheta_k + PITR_k \sum_{i=0}^{k} RESR_i OI_i + PIPR_k \sum_{i=0}^{k} q_i + \varepsilon_k$$

$$where\ \ OI_k = \alpha_k + \sum_{i=1}^{l} \beta_i OI_{l-i} + q_k$$

Transient Impact models

Non-linear transient impact models (TIM) relax the assumption of Almgren (2001) by assuming that price dynamics have a non-linear market impact component that is no longer fixed and permanent, but rather assumes a transient nature of price impact that decays over time (Gatheral, 2012). TIM decomposes market impact into an instantaneous market impact function, $f(V_k)$, which represents the immediate impact of trading in V shares in period k, and a decay kernel propagator function, $G(k)$, that represents the price impact of previously executed market orders that decay over time (Bouchard, 2004; Gatheral, 2010). The concept of price impact decay is modelled using a propagator function that treats current prices as a supposition of the decaying impact from the auto-correlated order flow of all previous trades (Gatheral, 2012; Eisler, 2011; Taranto, 2016). These models are in line with findings that show market impact as a concave shaped function of the volume traded on LOBs over short time frames, such that larger volumes of trades lead to higher market impact in the direction of the trades that decay over time (Bouchaud, 2009). Optimal execution algorithms that incorporate a non-linear TIM of the price, P_n, model the price dynamics of the initial price, P_0, as a continuous stochastic Wiener process where the price follows a Brownian motion, $dW(k)$, with volatility, σ, and is influenced by both instantaneous and decaying market impact, with volume traded in period k, V_k, such that:

$$P_n = P_0 + \int_0^n f(V_k)G(n-k)\,dk + \int_0^n \sigma\,dW(k) \qquad where\ \mathcal{G}(k) = G(k+1) - G(k)\ \ and\ \ j < k$$

The continuous-time price dynamics model with non-linear TIM posits that traders active in the stock will impact the market by their trades, whilst the stock price follows a Brownian motion as a result of trades not related to market impact, such as company announcements, when the trader is not active. Thus, the model is fully specified by the instantaneous and decay market impact functions. Instantaneous market impact is measured by the order flow from market order trades in period k, $f(V_k)$, which takes positive values when the market order is a buy and negative when the market order is a sell. The instantaneous market impact function has been shown through empirical studies to be a strongly concave function of the market volume traded by the investor, well fitted by a power-law function with exponents depending on the size of the volume traded (Lillo, 2003). Thus, this gives rise to the 'non-linear' and 'transient' nature of instantaneous market impact. Market impact is weighted by the decay propagator function, $G(n-k)$, which depicts the market impact's rate of decay at each period. The propagator function can phenomenologically be viewed as being influenced by limit orders submitted to the LOB which have a cushioning effect that counteracts recently executed market orders, minimising the market impact. The kernel function, $\mathcal{G}(k > 0)$, is a negative function decreasing over time which indicates that the market impact decreases as the number of market orders in the same direction increase, a phenomenon of 'asymmetric liquidity' where price impact of an order is inversely related to the probability of that order occurring (Lillo, 2004). IS optimal execution algorithms modelled with non-linear transient price impact require the formulation of an optimal strategy, $\mathcal{S} = \{V_k\}_{k \in N}$, derived from minimising the execution cost of trading X shares over period N represented by the IS objective cost function, $\mathbb{E}[C_T(\mathcal{S})]$. The optimal execution strategy in period n will choose the volume of shares to trade, V_n, that minimises the cost function, $C_T(\mathcal{S})$, with the assumption that all trades are made in the same direction, such that:

$$\mathbb{E}[C_T(\mathcal{S})] = \int_0^T V_k\,dk \int_0^n f(V_k)G(n-k)\,dk\,dn = expected\ IS \qquad \arg\min_{V_n} C_T(\mathcal{S})$$

Bouchaud (2004) use empirical data to derive power-law estimates of the instantaneous market impact, $f(V_k) = sign(V_k) * |V_k|^{\alpha}$, and decay market impact, $G(k) = k^{-\beta}$, terms as functions of volume traded and the number of periods k, with both α and β approximated to values of 0.5. Gatheral (2010) proves two constraints on the hyper-parameters, $\alpha + \beta \geq 1$, and, $\beta \geq 0.42$, by constructing a continuous time TIM model with non-linear price impact that assumes traders cannot execute price manipulation strategies with positive revenues, such that traders executing a large buy order are not able to increase their expected revenues by submitting intermediate sell trades. Price dynamic models with market impact components that assume no price manipulation also must assume that the cost function is greater than zero, thus, there is no negative expected liquidation costs (Klock, 2011). Dang (2014) transforms the objective cost minimisation problem expressed in the previous equation for $\mathbb{E}[C_T(\mathcal{S})]$, into a non-linear integral equation which meets the necessary condition of stationarity. Thus, given certain assumptions and constraints of the equation by parameter λ, the equation can be expressed as a non-linear weakly singular Urysohn integral equation of the first kind with the form:

$$\int_0^N F(V_k, n) G(|n-k|)\, dk = \lambda \qquad where\ \ F(V_k, n) = \begin{cases} f(V_k) & if\ k \leq n \\ V_k f'(V_k) & if\ k > n \end{cases}$$

Curato (2015) provide a piecewise approximation of the exact solution for the above equation using a Homotopy Analysis Method (HAM). The HAM is discretised by splitting the total period into N intervals, to devise an mth order deformation HAM local minimum optimal execution solution for the integral equation above, assuming non-linear transient market impact. HAM requires an initial estimation of the optimal solution, generally VWAP, and applies continual deformations of the guess to improve the approximation of the solution. Curato (2015) solves for the mth order deformation optimal HAM solution, $V_k^{(n)} * h_{opt}^m$, that indicates the volume to trade in the kth period, by first determining the optimal convergence control parameter, h_{opt}^m, that minimises the mth order squared residual error, $\varepsilon^m(h)$, of the equation for λ, giving:

$$h_{opt}^m = \arg\min_h \varepsilon^m(h) \qquad where\ \ \varepsilon^m(h) = \sum_{k=1}^{N} \left[-\lambda + \sum_{j=1}^{N} G_{jk} F_{jk}(V_k) \right]^2 \quad and\ \ \lambda = \sum_{j=1}^{N} G_{jk} F_{jk}(V_k)$$

Thus, the zero-order optimal HAM execution strategy over the period N will slice the parent order into a trading schedule where $V_k^{(0)} * h_{opt}^0$ shares are executed in the kth period, for user-defined hyper-parameters α and β. Curato (2015) proposes that the optimal solution using the continuous deformation process takes a non-time-symmetric front-loaded shape when market impact is concave, with these solutions shown to empirically outperform more basic VWAP strategies. They extend the above continuous deformation of a guess solution by modelling trades over a discrete grid of N equal intervals and perform numerical optimisation of the objective cost function using Sequential Quadratic Programming (SQP) to solve for the global minimum of the cost function directly. Curato (2015) finds that the optimal solution at the global minimum involves a strategy of mixed buying and selling. This involves decomposing a large buy parent into bursts of intense buying of shares which is followed by weak and interspersed sell orders over an extended period, in a continuous cycle until the parent order is exhausted. These results are particularly interesting given that they rationalise the behaviour of firms in high-frequency markets to both buy and sell shares throughout the trading day, even when that firms interest is in building up a position in a single direction.

History Dependent Impact models

Eisler (2011) proposes the history dependent impact model (HDIM), which posits that only price-changing events should be impounded into a market impact model rather than all events as is the case in the TIM. The dynamics of the LOB are dependent on the type of order, π, with only price changing events, π', having a market impact which is modulated by the history of previous events using an 'influence kernel', $\mathcal{G}_{\pi_k,\pi_t}$, that measures the average immediate impact for a specific price changing event. Including non-market orders as LOB events in the model of price impact allows for a framework capable of analysing all events that influence the LOB. This diverges from TIM's focus on market orders, only incorporating other events indirectly in the decay component of market impact which acts as a liquidity refill mechanism as limit orders begin to arrive to counterbalance instantaneous market impact (Bouchard, 2006; Gatheral, 2012). HDIM assumes that price impact is both permanent and history dependent on the type of event. Eisler (2011) models price return at time t, R_t, as follows:

$$R_t = P_t - P_{t-1} = \Delta P^R_{\pi_t}\epsilon_t + \int_{k=1}^{t} \mathcal{G}_{\pi_k,\pi_t}(t-k)\,\epsilon_t + \eta_t \quad where:$$

$$\mathcal{G}(k) = G(k+1) - G(k)\ ,\ \ \Delta P^R_{\pi_t} = \langle \Delta P_t | \pi_t = \pi \rangle\ , and\ \ \epsilon_t = \begin{cases} 1 & for\ buys \\ -1 & for\ sells \end{cases}$$

where $\Delta P^R_{\pi_t}$ is the average realised immediate midpoint price change of the order event type π_t executed at time t. For example, market orders that are sized less than the opposite best price volume, or cancellation orders, are not expected to move the immediate midpoint, whereas market orders larger than the best price depth or a limit order placed in the spread will. ϵ_t represents the sign of the trade with value one for buys and minus one for sells. η_t is a term that models independent noise from price movements not caused by trades, such as company news. $\mathcal{G}_{\pi_k,\pi_t}$ is the negative 'influence kernel' or matrix whose size depends on the total number of actions in the model and the number of actions that cause price changes. The influence kernel's elements define the amplitude of the price impact for a specific event based on the historical price changes.

Evidently, the price impact dynamics of the HDIM can be decomposed into three different processes. First, the term $\Delta P^R_{\pi_t}\epsilon_t$ represents the immediate realised price jump caused by the event π_t at time t. The other two components are derived within the integral term of the return function. Returns induced by the current event at t will have influence over the price movement probabilities of future events which is defined by the correlation between events, which is expected to be very small and not modelled within the above return function. Finally, the impact of the event at t on the 'gaps' behind the best price is described by the indicator matrix $\mathcal{G}_{\pi_k,\pi_t}$ and its lagged dependencies, which are measured using a vector autoregression, calibrated using past data. HDIM provides execution algorithms with a framework for modelling the market impact from sequences of events on the LOB. Taranto (2016) empirically compares HDIM and TIM and finds that HDIM performs slightly better at modelling price impact dynamics for small tick stocks, whereas Eisler (2011) do not find any evidence of this.

This thesis utilises the HDIM developed by Eisler (2011) to build a series of history-dependent price impact features using the influence kernel $\mathcal{G}_{\pi_k,\pi_t}$. Market impact features for bid orders, $PIIKBR^{\pi}_t(k)$, and ask orders, $PIIKAR^{\pi}_t(k)$, are defined at t depending on the type of order placed on the LOB π and lagged intervals periods of $k \in 10, 60$ seconds. The metrics are developed using a rolling correlation of the measures over the relevant period to measure the estimated market impact of a specific order event.

Market Resiliency

Market resiliency is an important dimension of liquidity, price impact, and general execution costs of market participants that measures the LOBs ability to dynamically absorb order flow, reverting to an equilibrium level in a timely manner following a significant liquidity shock. Resiliency is concerned with the durability of a LOB's replenishment mechanism, the speed at which net liquidity flows onto the LOB following the submission of a new limit order, cancellation of a resting order, or execution of an aggressive order. The dynamic nature of market resiliency differs from the static dimensions of liquidity when analysing market depth and quoted spreads. Depth provides a quantitative framework for estimating the markets liquidity and ability to absorb large orders with minimal price impact at a single point in time. Similarly, quoted spreads represent a price dimension of liquidity at a single point. Market resiliency measures the dynamics of these two liquidity measures when a liquidity shock event occurs. Market participant expectations regarding the LOBs response, level of fragility and ability to neutralise liquidity shocks is a critical input into execution algorithms. Resiliency has a marked effect on IS, trading costs and execution risk as liquidity levels fluctuate, hence, its relevance for algorithmic trading strategies.

Various academic studies and theoretical works have been conducted in this area. Foucault (2005) builds a theoretical LOB model populated with dynamic traders of various degrees of patience, finding that the resiliency of the quoted spread is positively correlated with the proportion of traders that are patient and the tick size, whilst resiliency is negatively related to the order arrival rate. Kempf (2015) develops a framework for measuring resiliency, finding evidence that the resiliency of quoted spreads and market depth is significantly reduced for stocks with high levels of informed trading or information flow. Obizhaeva (2013) model the resilience of the LOB that recovers back to a steady state after the large price impact of a market order, as new limit orders are submitted close to the bid-ask spread. Alfonsi (2010) use a non-uniform distribution of LOB market depth with a given density function to model the exponential spread and volume reversion following large market orders on resilient LOBs with non-linear price impacts. Reversion of spreads and volume are empirical facts exhibited by resilient LOBs (Bouchaud, 2004). The distribution of limit orders on the LOB at prices above the martingale best ask price is defined by the term $f(x)dx$, with f a non-linear continuous density and constant shape function. Optimal strategies for LOBs with resilience are bucket-shaped with large unidirectional front and end loaded orders, with minimal trading during the interim period. This optimal strategy will size sequential market orders by the exact amount that the LOB has recovered since the previous large market order using an exponential decay term $e^{-\rho\tau}$. The resilience speed, ρ, is a positive constant that calibrates the decay in the wider spread and replenishment of volume depth following a market order, with τ equal to the length of trading periods.

We develop a framework for measuring resiliency on the LOB using an auto-regression mean-reversion model. The high autocorrelation between liquidity levels and their lagged values requires the fitting of an autoregressive model of lagged liquidity values that eliminates the serial correlation of the error residuals. Market resiliency is a dynamic measure of the speed of liquidity replenishment or rate of mean reversion of liquidity metrics following liquidity shocks over a given time frame. These liquidity shocks come in the form of a temporary depletion of market depth or shifting of the best bid-ask quoted spread following the uninformative inflow of orders. Resilient LOBs involve the submission of corrective limit orders placed by informative and competitive traders that neutralise market impact from the order flow

and reverts market prices and liquidity levels back to equilibrium. We develop resilience features that extend from the frameworks of Kempf (2015) and Foucault (2005). Liquidity fluctuations over the past period, ΔL_t, are modelled using a mean-reversion framework as a function of past period liquidity levels, L_{t-1}, the long-run mean reversion liquidity level, θ, and the market resiliency or speed of reversion of the liquidity metric back to its long-run value with random median zero error term, ε_t, giving:

$$\Delta L_t = \rho(\theta - L_{t-1}) + \varepsilon_t$$

Resiliency of the LOB is analogous to the strength of the liquidity adjustment factor, ρ, which is estimated using an l-order autoregression model that incorporates past liquidity values to capture and eliminate autocorrelation. This model is implemented to minimise the risk of biased estimates of the resiliency variable. We build features that calculate liquidity and resiliency for both quoted spreads, $RESQR_t$ and L_t^{QS}, and market depth, $RESDR_t$ and L_t^{MD}. Measures are calculated using a rolling regression technique. The autoregression model is estimating time intervals of length $k \in 1, 5$ minutes, with each interval using $j \in 5$ lagged variables of past liquidity levels, giving regression model:

$$\Delta L_t^{QS/MD} = \alpha_t + \rho L_{t-1}^{QS/MD} + \sum_{i=1}^{j} \beta_i \Delta L_{t-i}^{QS/MD} + \varepsilon_t \qquad where\ \ \rho = RESDR_t\ and\ RESQR_t$$

Impulse response functions

Impulse responses functions (IRF) quantify the dynamic adjustment of the LOB following a liquidity shock. These functions are a quasi-measure of a LOB's market resiliency. To understand the dynamics of various liquidity variables, we develop a kth order vector autoregression (VAR) model, VAR(k), to capture the linear interdependency between the endogenous liquidity variables and their k lagged values over a given sample period. The VAR(k) process generates a stationary time series, defined by:

$$L_t = \mu_t + \sum_{k=1}^{t-1} A_k L_{t-k}$$

where A_k are $M \times M$ coefficient matrices for the liquidity level variables L_{t-k} at time lag k, whilst μ_t is an M-dimensional vector with expected value zero and positive definite time-invariant covariance matrix.

We extend the framework of Lo (2015) to develop an impulse response function, $R(n; \delta)$, that measures the expected shift in a liquidity level variable from $\mathcal{L}_t$ to $\mathcal{L}_{t+n}$ after a time period of length n and for given shock values δ. The relevant liquidity variable analysed relate to market depth and quoted spreads on the LOB. Liquidity shocks are differentiated based on one of many commonly observed actions, $\mathcal{A}$, performed by traders on the LOB. These actions are all expected to have some impact on liquidity level variables, and include the execution of market orders, and the placement and cancellation of limit orders. The shock matrix, $\delta(\mathcal{L}, \mathcal{A})$, represents the initialised long-run equilibrium market impact, for a given action in set $\mathcal{A}$, on the liquidity variable in $\mathcal{L}$. Positive shock values indicate that the action taken is likely to increase the liquidity level, whilst negative shock values are more common in realistic and efficient markets when market orders are executed, such that liquidity is depleted. Evidently, the impulse response profiles illustrate the dynamic adjustment paths for a liquidity metric when a specific action is executed. The IRF equation bellow quantifies the empirical difference between expected liquidity levels at the end of period n when an initial shock or event δ is executed at period t, given all available information regarding lagged liquidity values:

$$R(n, \delta) = E\left(\mathcal{L}_{t+n} \middle| \mathcal{L}_t + \delta_{\mathcal{L},\mathcal{A}}, \mathcal{L}_{t-1}, \dots\right) - E(\mathcal{L}_{t+n} | \mathcal{L}_t, \mathcal{L}_{t-1}, \dots)$$

Estimation of the impulse response function, R, requires the derivation of a vector moving average (VMA) process to estimate the coefficients of the lagged liquidity variables, by using repeated substitution of lagged liquidity values and a Wold moving averages decomposition of the VAR(k) process:

$$L_t = M_t + \sum_{i=k}^{t-1} A^i \mu_{t-i}$$

where M_t contains both a deterministic trend and the effect of the exogenous liquidity level variables on the initial condition. The impulse response function is generated recursively using the estimated coefficients, A^i, from the VAR equation above, shock matrix, $\delta_{\mathcal{L},\mathcal{A}}$, and response periods n, to derive:

$$\hat{R}(n, \delta_{\mathcal{L},\mathcal{A}}) = \hat{A}^n \delta_{\mathcal{L},\mathcal{A}}$$

Resilience in LOB liquidity is a concern for markets structures with endogenous liquidity provision. IRFs present a method for illustrating this resilience on micro time and event scales, especially useful for UK equity markets with high-frequency LOB dynamics. Furthermore, resilience is an essential component of market quality for the efficient deployment of order-splitting execution algorithms. Resiliency in this context is measured by the speed in which liquidity levels revert back to equilibrium levels after a liquidity shock dissipates by a certain percentage, which we set as the half-life 50%, and significant 90% levels. These resilience metrics are measured in both event and chronological time. For example, the resiliency of the market depth with 90% reversion will measure both the chronological time, in seconds, and the event time, in number of events, following a specific action-induced liquidity shock, such as a bid market order, for the market depth to adjust back to 90% of its historical level. We quantify resiliency by first estimating the VAR(k) high-frequency model of liquidity dynamics, L_t, under an environment where liquidity variables, $\mathcal{L}$, are exposed to regular shocks given specific actions, $\mathcal{A}$. Finally, an impulse response function $\hat{R}(n, \delta_{\mathcal{L},\mathcal{A}})$ is estimated using the results from the VAR equation and the shock matrix $\delta_{\mathcal{L},\mathcal{A}}$.

Impulse response functions, $\hat{R}(n, \delta_{\mathcal{L},\mathcal{A}}) = \hat{A}^n \delta_{\mathcal{L},\mathcal{A}}$, and shock matrixes, $\delta_{\mathcal{L},\mathcal{A}}$, are used to develop two additional features relevant particularly for execution algorithms. Impulse responses for liquidity variables of spread, $IRFS_t^{\mathcal{L},\mathcal{A}}(k,n)$, and market depth, $IRFD_t^{\mathcal{L},\mathcal{A}}(k,n)$, are devised. The liquidity variables are defined as $\mathcal{L} \in (S, D)$ for quoted spread and market depth variables, respectively. Whilst events encompass $\mathcal{A} \in (BBP0, BBP1, BAP0, BAP0)$, which defines additions or deletions of liquidity at the BBP, one tick from BBP, BAP, and one tick from BAP, respectively. Features are computed for both variables at both the 50% and 90% level of liquidity replenishment, with response periods of $n \in 100$ lags of 100 milliseconds, and VAR order $k \in 5$ lags of 100 milliseconds.

2.3.6 Other Strategies

Several additional strategies are acknowledged but not incorporated into the model for various reasons. Statistical arbitrage is a common strategy employed by AT firms that exploit asset pricing differentials between different but correlated financial products. Directional strategies are generally based on non-microstructure data that more directly relates to the fundamental price of an asset, for example, company-specific information. Market manipulation strategies create a misleading representation of the market for a stock to impact the inferences of other traders regarding the fundamental price.

2.4 Feature Engineering

In this thesis, a series of deep learning (DL) methods are deployed to model how traders behave given the state of the limit order book (LOB) and the broader financial market. Feature engineering is performed to identify and generate a rich set of limit order book (LOB) features that one would expect to be integrated into the set of rules, calculations and operations underlying the algorithm of a firm conducting an AT strategy on UK equity markets. Thus, the strategies analysed above in Section 2.3 have each defined a set of raw LOB features relevant to that strategy. We amalgamate these features together to form a combined feature space which serves as an input into the DL algorithms employed in this thesis. The derived features are all numeric, measurable and independently observed characteristics of the current state and dynamics of the consolidated UK LOB in its totality, the algorithmic trader performing an action and various phenomena in financial markets. Appendix A defines the full set of 323 LOB features developed.

The objective of feature engineering is to construct a valid, holistic and optimal feature representation of the data that explains a classifier sufficiently and provides high performance when mapping the relationship between the generated features $\mathbf{x}$ and the output classifier $\mathbf{y}$ for a machine learning problem. Feature generation is the key component of feature engineering that involves the definition of independent and discriminating features that derive from pervasive microstructural phenomena extracted from the internal dynamics of the limit order book (LOB) and are assumed to serve as input into various algorithmic trading strategies. Providing the DL models with a complex feature space allows the model's learning algorithm to devise higher dimensions of abstractions, building more intricate relationships between the features.

Building a holistic feature space that fully captures the input-output relationship is critical for developing a well-specified DL model that is capable of attaining successful training performance and generalisation power (Mohamad, 2015). The dimensionality of the feature representation space also has consequences for the level of abstraction obtained by the model. Furthermore, given the highly complex nature of algorithmic trading, it is unlikely that any single or subset of features would explain a trader's actions completely. Rather, the use of a large feature space that allows the DL to analyse not just single features but also capture abstractions from interactions between multiple features that by themselves may be immaterial to the classifier but joined together, through some operation or function, provide information for the classifier (Bengio, 2013; Hinton, 2006).

The following section explains the feature scaling process used for LOB features. Additionally, feature selection extraction methods utilised in this thesis are defined and explained.

2.4.1 Feature Scaling

Feature scaling requires the transformation of feature values using standardisation, normalisation, winsorisation and other techniques that ensure the transformed input data is both applicable for, and improves the conditioning of, the DL models. Feature scaling is performed on the LOB datasets used in this thesis during the pre-processing phase to standardise and normalise the n raw feature values for each specific trading day and trading date pair. Furthermore, additional processing steps are taken for

features that exhibit high kurtosis or skew. Feature values with novel or extreme distributions require transformations into more relevant Gaussian-based distributions with comparative boundaries that form the input for machine learning algorithms. The primary benefit from feature scaling transformations using standardisation and normalisation is the improvement in precision and speed of convergence for learning algorithms as patterns can be found easier in the representation space (Ioffe, 2015).

Features that evidence high levels of skewness or excess kurtosis, graphically represented by their non-Gaussian asymmetric statistical distribution and fatter tail distributions around its mean, require further feature transformation in the pre-processing phase. Skewness and Kurtosis are dimensionless descriptors of each feature's distribution. Skewness is quantified using the Fisher-Pearson moment coefficient of skewness equal to the third standardised moment, $S_{\mathbf{x}}$, which is the ratio of the third central moment around the mean, $\mu_{\mathbf{x}}^3$, to the third power of the standard deviation, $\sigma_{\mathbf{x}}^3$. Excess kurtosis measures the fourth standardised moment, $K_{\mathbf{x}}$, which is the ratio of the fourth central moment around the mean, $\mu_{\mathbf{x}}^4$, to the fourth power of the standard deviation, $\sigma_{\mathbf{x}}^4$. High positive or negative skewness values indicate that distributions have higher concentrations of values on one side of a mean centred distribution, with significant presence of one-sided outliers. High positive excess kurtosis is common in leptokurtic distributions with a higher quantum of outliers and non-outlier data points centralised around the mean, relative to a normal distribution. These distributions are common for financial metrics with high and concentrated zero or NA values. The feature scaling parameters can be mathematically defined by:

$$S_{\mathbf{x}} = \frac{\mu_{\mathbf{x}}^3}{\sigma_{\mathbf{x}}^3} = \frac{E[(\bar{\mathbf{x}}_N - \mu_{\mathbf{x}})^3]}{E[(\bar{\mathbf{x}}_N - \mu_{\mathbf{x}})^2]^{3/2}} \qquad K_{\mathbf{x}} = \frac{\mu_{\mathbf{x}}^4}{\sigma_{\mathbf{x}}^4} - 3 = \frac{E[(\bar{\mathbf{x}}_N - \mu_{\mathbf{x}})^4]}{E[(\bar{\mathbf{x}}_N - \mu_{\mathbf{x}})^2]^2} - 3$$

Log transformation of features, $\mathbf{x}_N$, with skewness values > X or kurtosis values > Y, are performed to provide more effective distributions of feature inputs from which DL models can build abstract patterns and relationships. The log transformation, x_{LT}, performs a log operation that widens the distribution of small absolute value features, generally around zero, and compresses the distribution of high absolute value features. Log transformation procedures involve:

$$x_{LT} = \epsilon_x \ln(1 + |x|) \qquad where\ \epsilon_x = \begin{cases} 1 & if\ x \geq 0 \\ -1 & otherwise \end{cases}$$

DL models function well when features values have similar mathematical and distributional properties. This ensures that no feature dominates the cost function during learning and each feature has a comparable capability of influencing the DL models' predictor function. Following log transformations of non-normally distributed input features, we account for outliers by performing winsorisation at the five-standard deviation level. Outlier trimming has been suggested as a possible method of outlier treatment for unstructured or contaminated data using the least trimmed squares criterion when fitting a non-linear DL model (Beliakov, 2011). However, the datasets used in this thesis are sufficiently valid, clean and structured to build confidence in winsorisation with wide boundaries as the correct method for outlier treatment. We next perform feature standardisation to transform feature data point value, x, based on its distance from the feature's population mean, $\mu_{\mathbf{x}}$, relative to the feature's population standard deviation, $\sigma_{\mathbf{x}}$, into a standardised data point, x_S. After Gaussian Z-score standardisation, each feature will have a zero-mean and unit-variance, procuring distributional properties in line with normal distributions. Finally, min-max feature scaling normalisation is utilised to rescale feature values into the

range [-1,1] to attain the final feature value, x_N. Rescaling provides robustness for features with high sparsity to maintain the sparseness of the feature data. The normalisation process can be defined as:

$$x_S = \frac{x - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}} \qquad x_N = 2\left(\frac{x_S - \min(x_S)}{\max(x_S) - \min(x_S)}\right) - 1$$

DL models trained with first-derivative backpropagation-based optimisation algorithms have been shown to respond well to pre-processing raw feature vectors to attain features with similar mathematical attributes and scales (Nair, 2010).

2.4.2 Feature Selection & Extraction

Feature selection involves statistical and machine learning techniques that select the optimal subset of a generated feature set and discards irrelevant features, with the objective of improving the accuracy of the machine learning model. Feature selection encompasses the process of selecting the m-dimensional optimal subset feature vector, $\mathbf{x}_{FS} = \{x_{1,} \dots x_{m,}\}$, of the original n-dimensional input feature vector, $\mathbf{x} = \{x_i | i = 1, \dots, n\}$, where $m < n$. This process involves the application of statistical and machine learning methods to eliminate redundant, non-informative and extraneous variables from the original input vector. The objective of feature selection is to reduce the dimensionality of the machine learning problem, improve performance and to reduce the risk of overfitting the training data (Guyon, 2003). By removing irrelevant and redundant features, the selection process improves accuracy and makes the machine learning task more comprehensible and easier to solve (Liu, 2010). Selecting the optimal subset of features has been shown to be a non-deterministic polynomial-time hard problem that requires an exponential search and comparison of all possible subsets, with complexity $\mathcal{O}(2^n)$ for n-dimensional feature vectors (Liu, 2005). The high-dimensional nature of datasets analysed in this thesis renders this process both computationally cumbersome and practically intractable. Thus, to yield a sufficient non-globally optimal feature subset, we focus on metaheuristic feature selection algorithms that utilise feature space exploration techniques to overcome the computational limitations of conducting an exponential search of all possible subsets. Generally, optimisation of the feature set encompasses the procedure of minimising an objective function by iteratively dropping irrelevant features that provide no information regarding the classifier, until a final subset, $\mathbf{x}_{FS}$, is reached.

Feature selection metaheuristic models can be broadly classified into categories of classifier-dependent filter, classifier-independent wrapper and embedded models depending on how the subset feature vector is selected (Kohavi, 1997). Furthermore, these models differ in how they assign relevance to features and deal with issues such as feature interaction and irrelevant features. In particular, redundant features with high correlation to the classifier or irrelevant features that provide no useful information for the classifier, may be considered as such under a univariate filter model, but when considering their interaction with other features using a wrapper search or embedded neural network weight model, they may no longer be considered irrelevant. The performance of the three methods generally can be differentiated along several axes – computational resource requirements, probability of overfitting, dataset sample size and representational space. The following sections introduce relevant filter, wrapper and embedded feature selection models before presenting a high-level explanation of feature extraction.

2.4.2.1 Filter Models

Filter feature selection methods use computationally inexpensive filter techniques, independent of the model and are employed before training the predictor. These methods evaluate the intrinsic characteristics of each feature, based on dependency, distance or consistency measures, and assigns a weighting or ranking to the feature based on a heuristic criterion. Feature subsets are then constructed by selecting high scoring features that meet a specific ranking threshold or percentile (Ahmad, 2008). Filter-based algorithms are generally machine learning model agnostics and can be utilised in various neural network architectures or other models. Filter models generally assume feature independence, thus, do not account for the relationship between interacting features and the classifier, with univariate ranking methods potentially dismissing important information from these interactions. Three primary filter algorithms that are used for feature selection include the univariate Pearson coefficient (Biesiada and Duch, 2007) and mutual information criterion (Estevez, 2009). Generally, the algorithms sequentially compute a feature importance metric for a feature, before deciding whether to include the feature in the optimal subset, $\mathbf{x}_{FS}$, if its importance value meets a certain threshold.

2.4.2.2 Wrapper Models

Wrapper-based feature selection methods 'wrap' the classifier within an iterative feature selection learning algorithm, A, that evaluates the efficacy of the initialised feature subset, $\mathbf{x}_{S^0}$, at training the machine learning model. The method then performs a perturbation update to the feature subset in a way that improves the accuracy of the model (Kohavi, 1997; Guyon, 2003). Wrapper models that incorporate a learning algorithm for feature selection trade-off higher computational expense, which scales with the dimensionality of the feature space, for more effective and accurate feature selection results, when compared to filter methods (Gheyas, 2010). Wrapper algorithms, A, require the definition of an evaluation function, $E(\mathbf{x}_S \ldots)$, a feature evaluation criterion, $\gamma(\ldots)$, a search strategy, $\theta(\ldots)$, and a stopping criterion, δ, to avoid exhaustive searches (Kohavi, 1997), along with the initialised feature subset. These algorithms initialise with the current feature vector, $\mathbf{x}_{S^0}$, and iteratively search for a new feature subset, evaluate that subset based on the cost function, $\mathcal{L}$, until it reaches the stopping criteria δ at which point the process terminates. The aim is to select the subset of features that minimise the objective cost function. This requires the implementation of a search strategy that formulates permutations of input features with the objective of traversing the feature space in a way that optimises feature selection in a timely manner. Several search strategies are commonly employed in wrapper-based feature selection algorithms including exponential search, greedy search and sequential forward selection methods.

Exponential search or brute force are exhaustive search strategies that holistically loop through all available subsets and return the optimal feature subset, though due to its high $\mathcal{O}(2^n)$ complexity it is generally computationally intractable and will require a reduction in the search space by using various heuristics (Liu, 2005). Greedy hill-climbing algorithms are non-optimal heuristic forward selection search methods for high dimensional datasets. These algorithms initialise with one feature, then iteratively update the subset of features, incrementally adding single features to the subset if the classifier evaluation criterion improves (Yang, 1998). These monotonic algorithms may lead to a feature subset stuck in a local minimum, an issue that can be subdued with genetic algorithms that allow mutations and

crossovers between feature subsets as a way of combining or removing features in an iterative learning system. Sequential forward selection (SFS) algorithms are an additional greedy search algorithm that differs from less computationally expensive random hill-climbing algorithms by incorporating an argmax argument into the feature selection strategy that sequentially combines the feature subset with the feature that minimises the objective function (Jain, 1997).

2.4.2.3 Embedded Models

Embedded models of feature selection are inextricably linked to the machine learning model being deployed and its internal parameters. For neural networks, embedded-based feature selection forms part of the backpropagation algorithm that assigns weights. Features with smaller absolute learnt weights propagate their values through the neural units with lower amplitude than those with higher learnt weights. Thus, feature selection is implicitly performed by the networks gradient descent learning algorithm that assigns the weights to the different features (Song, 2013). The benefit of higher accuracy from feature selection is weighed against the loss of generalisability between models, with feature selection specific to the model and its classifiers. Embedded models are most useful when applying feature selection to problems with a high-dimensional feature space. The advantage of embedded models is their ability to perform feature selection simultaneously with classification in an iterative process nestled within the machine learning model.

2.4.2.4 Feature Extraction & Dimensionality Reduction

Feature extraction applies a statistical transformation to the original feature space which is compressed into discriminative components of a smaller dimension. The objective of reducing the feature space is to retain critical information, eliminate redundant features, and reduce variance in the predictors, though at the cost of less interpretability of the feature space. Feature extraction has traditionally been used in high-dimensional machine learning problems through the application of dimensionality reduction to the original feature space, $\mathbf{x}_i \in \mathbb{R}^n$, into a new feature vector over a lower dimensional manifold, $\mathbf{z} = g(\mathbf{x}): \mathbb{R}^n \rightarrow \mathbb{R}^m$, where $m < n$. Dimensionality reduction involves a mathematical transformation of the original features that preserve the Euclidean structure between data points, though with some loss in explanatory power. Extracting 'new' features from the original set, $\mathbf{x}_i$, requires a non-linear mapping function, $g(\mathbf{x})$, that maintains the key information in the original structure.

2.5 Summary

This chapter provides contextual material for the deep learning models employed in forthcoming chapters that seek to analyse the internal dynamics of algorithmic trading strategies and LOBs, the experimental units of analysis for this thesis.

Section 2.1 introduced the concept of market microstructure as a manifestation of the 'trading mechanism' employed by trading venues to modulate the exchange of financial products on electronic LOBs. Market microstructure has ramifications for price and liquidity formation, the price discovery process, and explicit and implicit transaction costs for traded securities. Price formation arises naturally

from the process of traders interacting in markets, generally on LOBs, which is governed by a set of trading rules. We defined the design, structure and rules underlying the operation of UK equity trading venue exchanges and the current legislative and regulatory framework under MiFID in which trading firms and exchanges operate.

Section 2.2 provided an exposition of the design, role and dynamics of LOBs which serve as the primordial mechanism in contemporary UK equity markets for matching buyers and sellers seeking to trade a financial security. LOBs provide a global solution for matching traders by collating limit orders submitted by local traders conducting heterogeneous trading strategies onto a single platform. Market microstructure in conjunction with LOB dynamics provide boundaries between which optimal trading strategies are executed to find willing counterparties to a trade.

Section 2.3 explores these trading strategies with a focus on those conducted by algorithmic traders. We define these algorithmic traders as those conducting profit-driven trading strategies that utilise technologically sophisticated low-latency co-located Automated Trading Systems (ATS). These systems perform statistical, machine learning, fundamental and technical analysis of high-frequency LOB and financial data, which inform objective rules on how orders are submitted, amended and cancelled on an exchanges LOB. Objective rules involve defined quantifiable parameters relating to the trading firms' perception of the state of financial markets. Rules are incorporated into algorithms that are executed automatically and autonomously using the firms ATS. This section also explores the equity market trader ecosystem, which has come to be populated with a multifarious array of heterogeneous algorithmic trading firms bounded by the dynamics of the LOB. The market's trader ecosystem can be viewed externally as an amalgamation of disparate market participants, with each executing optimal order placement algorithmic-based strategies to meet their individual business objectives. We analyse the academic literature to derive five separate algorithmic trading strategies - Automated Market Maker, Execution, Microstructural, Momentum and Technical strategies. These strategies are categorised based on the type of algorithmic inputs or features expected to inform the execution of that strategy.

Section 2.4 utilises a feature engineering framework to extract and amalgamate relevant features from the literature analysis performed in Section 2.3, to inform a holistic representation of the LOB state. Feature generation is first performed to identify and generate a rich set of independent and discriminative LOB features that one would expect to be integrated into the set of rules, calculations and operations underlying the algorithm of a firm conducting an AT strategy on UK equity markets. A scaling procedure is executed next to manipulate features into a form applicable for use in deep learning models employed in the subsequent chapters. Providing deep learning models with a complex feature space allows the model's learning algorithm to devise higher dimensions of abstractions, building more intricate relationships between the features. Utilising these models allows us to better understand algorithmic trading strategies, their impact on UK equity market quality, and the dynamics of LOBs, using models deployed in the forthcoming chapters.

Chapter Three

3 Modelling Algorithmic Trading using Deep Neural Networks

Equity markets have undergone rapid transformation since the beginning of the century, ostensibly driven by an evolution in technology and the market microstructure characterised by the ascendency of automated trading predicated on machine intelligence. Quality financial markets are increasingly organised using a limit order book (LOB) based market structure that attracts algorithmic control of the trading process and rewards sophisticated algorithmic trading strategies capable of dominating these new agile and fast markets. Algorithmic Trading (AT) firms employ systems, infrastructure and complex algorithms based on objective rules capable of inferring trading signals from market microstructural phenomena at the microsecond level that are used to inform strategies to generate profitable trading opportunities.

Regulators with a remit for maintaining effective, competitive and efficient markets are increasingly concerned about fairness in this automated trading environment and the potential risks to the operation of well-functioning quality markets posed by complex algorithmic trading strategies. These regulators also want to understand how these strategies are deployed on the LOB in order to identify negative strategies that reduce market quality and to develop stronger policies that placate the investor communities concern that certain traders have unfair competitive advantages in modern financial markets. To address these concerns, deep learning models such as Deep Neural Networks (DNNs) can be implemented by regulators to ascertain intricate representations in equity market trading data that drive specific algorithmic trading strategies, both those conducive and prohibitive to well-functioning markets, from which policy prescriptions and oversight functions can be developed.

In this chapter, a series of feedforward DNN supervised machine learning models are trained to capture the relationship between market participants interactions with the consolidated UK equity market and a set of LOB-based features that are critically inherent to specific AT strategies and their respective trading algorithms. The high-dimensional non-linear nature of modern LOBs requires complex deep learning architectures, such as DNNs, to model their dynamics and the behaviours of traders in these systems. Once an understanding of the complex representations in this relationship are developed, one can better understand what LOB features drive an AT firms' actions, that is, what phenomenon in the LOB dynamics is manifest in its algorithms from which specific trading strategies can be identified and categorised. This phenomenological analysis within a deep learning construct will result in traders being segmented into a specific trading strategy group in Chapter 4, before the aggregate impact of those strategies on UK market quality is assessed.

Utilising a proprietary broker-identified FTSE 100 dataset that captures and consolidates all messages between traders and the primary UK equity LOBs, the relationship between trader actions and LOB features is modelled using a DNN. Actions within the system are represented by a space of possible order placement options available to the trader at a single point in event time. This includes the decision of

whether to submit or cancel an order, the order type, volume and price point of that order. Features are derived through an analysis of pervading literature on algorithmic trading strategies performed in Chapter 2 which identified a set of 323 features set out in Appendix A that serve as inputs into specific strategies trading algorithms. DNNs are utilised to model the complex non-linear relationship between trader actions and features for a set of five FTSE 100 securities traded over five days in June 2015. The specific actions executed by an individual broker, $\mathcal{B}$, in a trading security ISIN, $\mathcal{S}$, over a trading day, $\mathcal{D}$, represents a single sub- tuple dataset, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, from which that specific broker's trading strategy for that security and date can be modelled using a DNN. A total of 859 separate sub-dataset tuple DNNs are developed and trained before having their internal dynamics assessed in Chapter 4 to ascertain what features or algorithmic inputs were important for each specific strategy before this information is used to cluster firms by their trading strategy and have the impact of that strategy on market quality assessed holistically.

This chapter first outlines the theoretical literature, architecture and elemental components of Deep Neural Networks commonly deployed to solve classification tasks in Section 3.1. Comparative machine learning models are introduced in Section 3.2 that provide a basis against which the benefits from hierarchical feature abstractions pertaining to DNNs can be assessed against more basic models. Section 3.3 introduces the FTSE 100 consolidated dataset, the set of features extracted from the dataset and actions available to traders. Section 3.4 involves a comprehensive ablation analysis that evaluates the multifarious array of different DNN designs, architectures, parameter and hyper-parameter components that can be deployed to model the relationship between algorithmic trader actions and LOB features. The evaluation is conducted to determine the configuration of the optimal DNN that performs best at modelling the non-linear action-feature relationship with this optimal model presented and analysed in Section 3.5. Finally, a comprehensive assessment of the optimised model is performed through a comparative analysis against competing machine learning models in Section 3.6.

3.1 Deep Neural Networks

Deep Neural Networks (DNN) are complex self-learning algorithms that solve optimisation and supervised machine learning classification problems using models of biological neural structures which mimic human intelligence. DNNs manifestly represent the leading-edge of machine learning algorithms for solving non-convex multi-classification problems in various domains including computer vision (Simonyan, 2014; Krizhevsky, 2012), object detection (Szegedy, 2014), speech recognition (Hinton, 2012a) and language analysis (Weston, 2012). The success of DNNs stem from their ability to model relationships inherent in high-dimensional data with vast representation spaces and complex statistical dependencies. Various domains in computational finance provide rich, complex and multi-dimensional datasets which in turn provide a valid base for formulating predictive models using DNNs, a ubiquitous objective for market practitioners and academic researchers in this field. Deep learning and DNNs more generally have been used in finance to model mortgage risk (Sirignano, 2016), make general financial predictions (Heaton, 2017a), optimise trading strategies (Zhu, 2016), and model limit order book (LOB) dynamics (Sirignano, 2016; Dixon, 2017; Ntakaris, 2017). This thesis extends the application of DNNs into the realm of modelling algorithmic trading strategies based on the interrelationship between trader behaviour and LOB features.

DNNs are multi-layered systems of *artificial neurons* characterised by a specific neural network *architecture or topology* that models the interconnections and transmission of data between neurons in different *layers* based on a corresponding *activation function* and *learning algorithm* that minimises some *cost function.* Neural networks provide a machine learning framework for approximating a complex non-linear mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$, that relates input data in the form of a feature vector of neural units, $\mathbf{x}$, through multiple 'hidden' layers of neurons connected to adjacent layers via synapses, represented as weights, $\mathbf{w}$, to an output layer of values or classes, $\mathbf{y}$, based on some *activation function*, f. Learning is conducted by forward propagating training sample input vectors through the network towards the output layer before applying a *learning algorithm* in an iterative and continuous process that trains the neural network to find the optimal weight matrix parameters, $\mathbf{w}^*$, in a way that minimises a defined cost function, $\mathcal{L}$. Essentially, training a DNN with a *feedforward* architecture involves learning hidden layer weights through a sequence of non-linear transformations applied to the input feature vector, that results in an *internal representation* which improves capability of the DNN to more accurately predict outputs given inputs (Bengio, 2012). DNN *learning algorithms* are increasingly capable of comprehending, analysing and modelling the high-dimensional nature and complex non-linear dynamics of trading strategies based on high-frequency LOB datasets.

3.1.1 Neurological Framework

Deep learning models draw their architectural roots and decision-making structures from neurophysiological patterns existent in humans' central nervous system, particularly within the cerebral cortex. Representations modelled by deep machine learning algorithms are gross simplifications of the analogous biological functions performed within the cerebral cortex between the multifarious array of interconnected neurons. A hallmark of biological evolution in humans, the cerebral cortex, is a dense body of neurons that contain primary cortical fields capable of processing qualitative information and adjacent association cortical fields that execute more abstract thought processes to derive associations and patterns giving rise to memory and actions. Cortices are arranged into regions with multiple intercortical layers composed of varied distributions of neuronal cells that organise sensory and cognitive activities. The fundamental unit of both systems is the biological *neuron* which is composed of chemical *synapses* that receive stimuli through one-way connections from other signal-emitting neurons via their *axon.* This electrical signal is received at the pre-synaptic axon side of the synapse, inducing the release of variable portions of *neurotransmitters,* modulated by various chemical processes referred to as the synapses *efficacy*, which influences the degree and type of electrical signal that crosses the synaptic cleft onto the post-synaptic *dendrite* side, before entering the centre of the signal-receiving neural cell, also referred to as the *soma.* The various signals received by connected neurons in the soma are accumulated and transformed into a signal pulse that upon meeting a specific activation threshold ignites, opening the neuron's axon, and transmitting an electric pulse towards the dendrites of other connected neurons, repeating the cycle. Figure 3.1.1 provides an illustration of how electrical signals flow through a biological neuron, which has relevance given the theoretical dependencies of deep learning models on neurological structures.

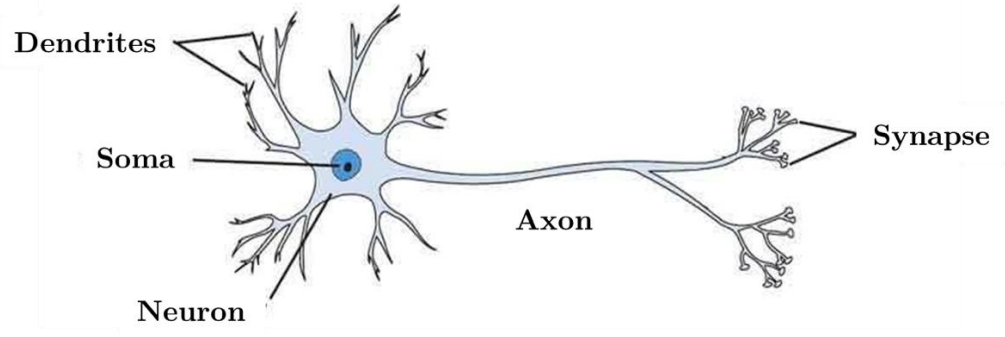

Figure 3.1.1 – Biological neuronal cell structure identifying the dendrite, soma, axon and synapse components. Adapted from various sources.

Some form of singularity or transcendence of machine over human intelligence is currently inconceivable with the human central nervous system and decision-making functions being overtly more sophisticated than current deep learning computational adaptions. In particular, the probability of chemical transfers between neurons and the dynamism of their stochastic transmission of information is near-infinitely complex for any machine to understand or be modelled by any current neural network architecture (Branco, 2009). Whilst DNN models have been developed based on a neurological framework, their objective is *not* to mimic all aspects of biological neurons but rather to *integrate* pervading concepts in their neurological structure into a mathematical machine learning model.

3.1.2 Generalisation & Stability

Integrating neurological elements into DNN models of trader behaviour is predicated on the objective of training stable non-linear classification models capable of generalising well to unseen data. Generalisation refers to a model's ability to make predictions from new data in the same way that the cerebral cortex learns to formulate human actions when new stimuli enter its system. DNN generalisation is assessed by first training the network on a training dataset and tuning hyper-parameters using a validation dataset before exposing the model to a test dataset from which its ability to make predictions can be compared against actual outcomes. The ability of the model to generalise largely depends on the complexity of the model, the size of dataset, N, the number of learnable parameters or degrees of freedom, Z, and the error of classification fraction, e, with the number of training examples necessary satisfying the order of magnitude function $N = O(Z/e)$.

Design of a network is motivated in part by ensuring a stable training process, that is, the condition where the algorithm moves towards convergence during training in a stable manner. Stability of a DNN refers to the smoothness of training or magnitude in diversity during the training process when input features are forward propagated through the network (Belic, 2006). Stable networks are critical given that instability in forward propagation poses questions regarding the model's ability to learn during backpropagation when the process is inverted. Examples of unstable networks are those that have poor convergence properties when small perturbations are applied to input features (Bottou, 2016), or networks that suffer from gradient diffusion problems (Bengio, 1994; Hochreiter, 1998). Furthermore, consideration should be given to the numerical stability of DNNs during training to avoid an under or overflow problem that leads to an unstable and poorly trained network (Wang, 2016). Design methods to improve numerical stability include bounding and normalisation of input and output values to reduce overflow errors, ensuring data is scaled into a suitable range that the DNN can process.

Overfitting is a common denominator that impacts both the DNNs generalisation and stability. This issue occurs during training when the model memorises internal patterns and representations within the training dataset that occur due to noise rather than the underlying distribution of the data, leading to poor generalisation when exposed to unseen data and instability in the learning process. Overfitting is largely prevented by employing early stopping and regularisation techniques addressed in forthcoming sections. Early stopping improves the model's generalisation by stopping the training of the model when the validation error is minimised or after a number of training epochs where the error remains stable. After this point, the model may continue training, though any new patterns learnt will likely be explicit noise conditioned on the training dataset model that is not existent in the underlying distribution of the dataset.

3.1.3 Feedforward Neuronal Architecture

This chapter employs a multi-layer feedforward DNN to approximate the input-output mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, \mathbf{b})$, that models the relationship between trader action, $\mathbf{y}$, and LOB input features, $\mathbf{x}$, by optimising the networks *learnable parameters* represented by the weight, $\mathbf{w}$, and bias, $\mathbf{b}$, matrices. Feedforward DNNs are a type of Multi-Layer Perceptron (MLP) where information flows from an input neuron layer, $\mathbf{x}$, through multiple connected hidden processing layers composed of neurons constrained by connecting weights, $\mathbf{w}$, in a non-cyclical forward direction with no feedback loops until the output layer, $\mathbf{y}$, is reached.

Feedforward MLPs draw their conceptual roots from the single-layer perceptron (SLP), an initial attempt at developing a binary linear predictor function to solve classification optimisation problems (Rosenblatt, 1958). The SLP was composed of artificial neurons first proposed by McCulloch (1943) as Linear Threshold Units (LTU) in which signals represented by an input vector $\mathbf{x}$, are multiplied by weights $\mathbf{w}$, with an added bias $\mathbf{b}$. The output signal of the LTU is modelled by a discontinuous Heaviside Step activation function where the neuron is activated if the function is greater than a threshold value, θ. The SLP develops a linear hyperplane that provides a prediction for the classifier, $\mathbf{y}$, of an input feature vector, $\mathbf{x}$, based on whether output signal meets the LTU threshold θ. SLPs are trained by updating weights using the delta rule (Rosenblatt, 1958). The DNN extends the basic linear input-output SLP by including 'hidden' layers in the neural network and incorporating non-linear activation functions in the design of artificial neurons, both which add non-linearity to the model as a way of approximating increasingly complex mapping functions. Furthermore, the optimisation of DNNs is generally performed using a backpropagation-based gradient descent algorithm (Werbos, 1974; Rumelhart, 1986), which extends the delta rule employed in SLPs.

Feedforward DNNs architectural structures are characterised by the quantum of layers, L, and level of dimensionality, that is, the number of neuron parameters in the network. Deep networks utilise a learning algorithm to optimise the learnable parameters of a mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, \mathbf{b})$, that maps an initial input vector, $\mathbf{x}^0 \in \mathbb{R}^n$, that propagates through the network in a forward direction until the final layer where it returns an output vector, $\mathbf{y}^L$. In a fully-connected neural network, the learning algorithm trains a series of chained functions, $f^{(l)}$, for each layer $l \in L$, composed of neurons connected through the parametrised weight matrix, $\mathbf{w}^{(l)}$. The objective is to attain an approximate mapping function, $f^*(\mathbf{x}, \mathbf{w})$, where f^* meets some predefined accuracy threshold.

Artificial neuron units are a loose adaptation of human neural cells that serve as the building blocks of DNNs. In a fully connected feedforward DNN, each layer is composed of *artificial neurons* that agglomerate input signals $\mathbf{x_0}$ emanating from each of the i connected neural units in the previous contiguous layer through one of multiple synapses. The level of input signals that enter the dendrite of the artificial neuron is dependent on the strength of the synapse, measured by the ratio of synaptic neurotransmitter chemicals received by the artificial neuron, and is mathematically expressed as a weight matrix, $\mathbf{w}$. These weights may be either positive value signals, exciting, or act as signal inhibitors and have a negative value. The decision to propagate data in the form of an output signal from the neural unit to neighbouring neurons in the next layer is dependent on the neurons *propagation function*, which transforms input signals with respect to the connecting weights into a network input value, $\mathbf{z}$. Propagation functions are generally the dot product of the input signal vector, $\mathbf{x_0}$, and the weight matrix of the receiving dendrites, $\mathbf{w}$, with an additional bias neuron, $\mathbf{b}$, integrated into the model. This multiplicative process is performed within the soma of the neural unit. An *activation function* is applied to value received through the propagation function within the soma of the neuron to determine the extent of the neuron's activation, expressed by the output vector, $\mathbf{x_1}$. Once the vector's output activation value is determined after applying the activation function, it travels along the length of the axon, then through synapses connected to the adjacent layer, serving simultaneously as the output of layer zero and the input vector to the subsequent layer one.

Mathematically, the activation output value for the jth neuron in the lth layer of the neural network, $\mathbf{x}_j^l$, is derived by applying the activation function, ϕ, to the activation input value, $\mathbf{z}_j^l$, which is equal to the dot product of previous layers, $(l-1)^{th}$, output values for all i neurons, $\mathbf{x}_i^{l-1}$, and the weights connecting the previous and current layer, $\mathbf{w}_{ij}^l$, with an adjusted bias term added, $\mathbf{b}_j^l$. Furthermore, the output activation value for neuron j, $\hat{\mathbf{y}}_j^L$, is generalised for the final output layer by applying an output activation function, φ, generally a softmax or identity function, to the output layer activation term, $\mathbf{z}_j^L$, giving:

$$\textit{hidden layers}: \quad \mathbf{x}_j^l = \phi(\mathbf{z}_j^l) \qquad \mathbf{z}_j^l = \sum_k \left[\left(\mathbf{w}_{ij}^l \mathbf{x}_i^{l-1}\right) + \mathbf{b}_j^l\right]$$

$$\textit{output layer}: \quad \hat{\mathbf{y}}_j^L = \varphi(\mathbf{z}_j^L) \qquad \mathbf{z}_j^L = \sum_k \left[\left(\mathbf{w}_{ij}^L \mathbf{x}_i^{L-1}\right) + \mathbf{b}_j^L\right]$$

$$\textit{where } \mathbf{x}^l \textit{ is a } [J \times 1] \textit{ matrix}, \mathbf{w}^l \textit{ is a } [J \times I] \textit{ matrix}, \mathbf{x}^{l-1} \textit{ is a } [I \times 1] \textit{ matrix}, \mathbf{b}^l \textit{ is a } [J \times 1] \textit{ matrix}$$

Designing the architectural configuration of a DNN is a critical component for devising an accurate and generalisable machine learning input-output mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$. Within the architecture of the DNN there exists significant flexibility in configuring the model's parameters, design elements and tuning hyper-parameters. A reduced form of the DNN architecture employed in this chapter is presented in Figure 3.1.2.

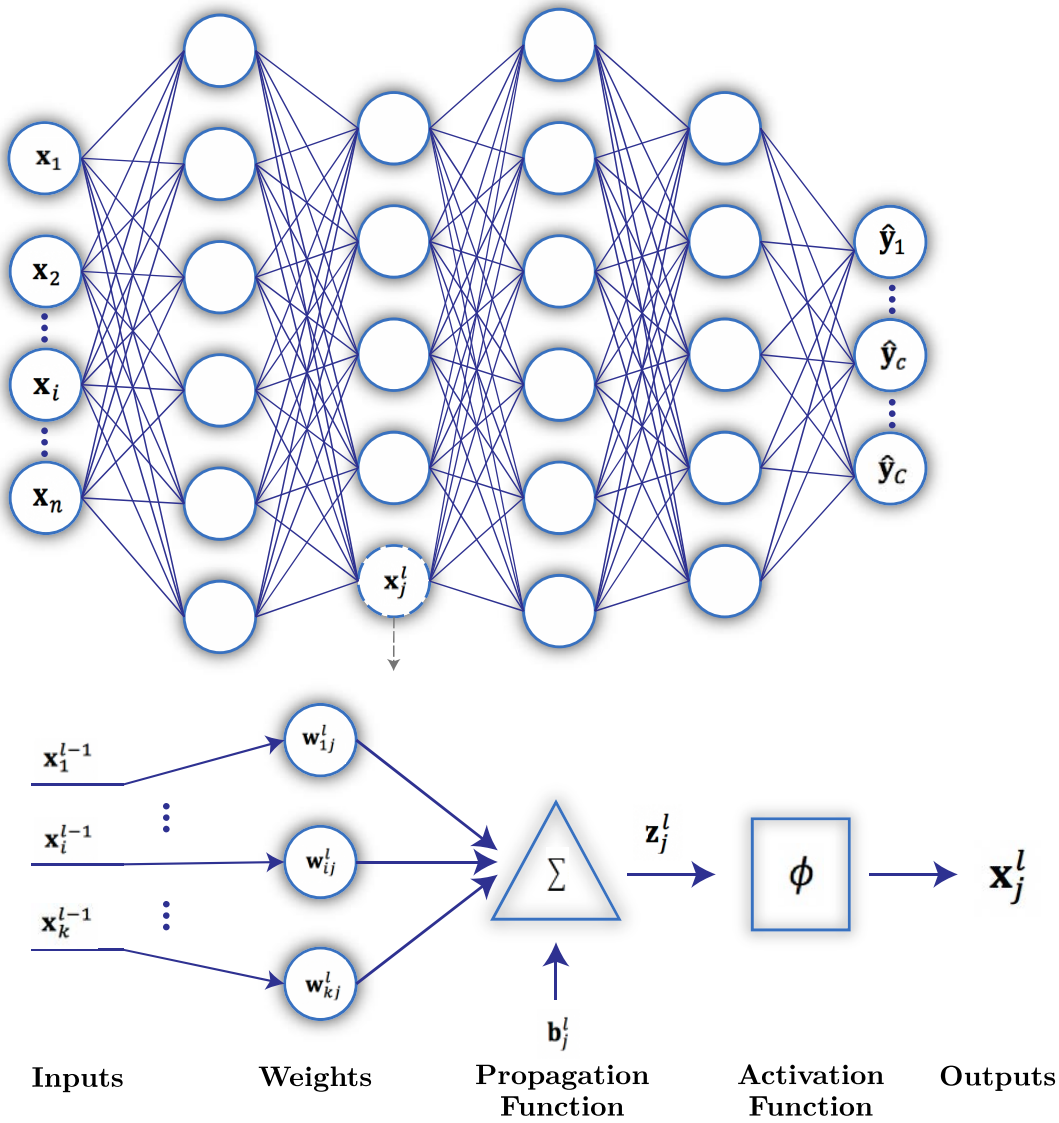

Figure 3.1.2 – The top figure represents a reduced form of the Deep Neural Network architecture utilised in this chapter. The architecture consists of four hidden layers with an n-dimensional input feature vector and C-dimensional output feature vector. The image on the bottom section explains the mathematical operations performed in a single neuron during a pass of the input vector from the previous layer resulting in an output value that feeds into the forward adjacent layer.

The current section has outlined the architectural specification of a standard DNN, including the number of hidden layers, neurons, direction and interconnectedness of the neural network. Sections below outline the decisions required to design, train and deploy a DNN capable of learning a classification function that maps the relationship between trader actions and high-frequency LOB features. First, a theoretical and empirical analysis of how depth in neural networks builds hierarchical feature representations, critical to the success of DNN models, is performed. Second, relevant activation functions that impact the non-linearity and form that output units can take are presented. Third, cost functions that define an error metric for the neural network are formulated, before explaining optimisation algorithms applicable to minimising these cost functions with a focus on first-order gradient-descent based methods. Finally, initialisation, regularisation, unsupervised pre-training and normalisation techniques that can be incorporated into DNNs to improve their performance are introduced and defined.

3.1.4 Hierarchical Feature Representations

Deep learning architectures create a hierarchical feature representation space where raw input features sit at the base of the model and progressively more complex or abstract features arise in deeper hidden layer representations as the cumulative quantum of non-linear mathematical transformations increases layer-by-layer (Bengio, 2009). The capacity and expressiveness of representable functions, f, capable of being identified by a DNN to approximate the mapping relationship $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$, is commonly associated with its level of dimensionality or depth, along with the networks complexity and the dynamism of intra-neuron connections. The expressiveness of a DNNs representation space relates to the number of permissible input configurations, whilst capacity refers to the size of the network which is correlated with network complexity given more parameters induces additional non-linear transformations.

DNNs are theoretically grounded in neurological science where humans' cognitive systems transform and process information through multiple levels of increasingly abstract representations (Serre, 2007). This concept extends to computer science where the level of feature abstractions and permutations correlates with the number of hidden layers and neurons in the network (Montufar, 2014; Bengio, 2013). Thus, deeper networks are capable of expressing more complex relationships than single-layer models (Bengio, 2007; Hinton, 2006). The objective of high-dimensional DNN architectures is to develop a hierarchical feature representation space of abstract features that disentangles and extracts information regarding hidden explanatory factors from raw input features. This allows for the development of a more accurate posterior distribution when predicting the output classifiers. Raw LOB financial features by themselves can rarely predict the actions of algorithmic traders otherwise uniformity in strategy would collapse financial markets. Rather, more abstract higher-level feature representations formed through the amalgamation and non-linear transformation of multiple subsets of features provide a more pertinent construct through which to model algorithmic traders' behaviours and actions.

Theoretical advancements in hierarchical representational learning (Bengio, 2013) and technological improvements in computational performance when training deep networks, particularly through the massively parallel architecture of GPUs, have added to, though not solved the academic debate on the merits of very deep neural network architectures. Deeper networks created by adding additional layers also expend ones limited computational budget during inference, and without regularisation or corrective measures, risks overfitting to noise in the training data. Furthermore, shallow neural networks developed using model compression (Ba, 2014) and networks compressed into thinner and wider structures trained using the Knowledge Distillation approach (Romero, 2015) have been shown to attain comparative permeance with that of deeper architectures. These results align with the Universal Approximation Theory (UAT) which stipulates that a single-layer Perceptron can approximate any non-linear function, given several restrictive assumptions (Cybenko, 1989; Hornik, 1991). The theorem states that a single hidden layer of a feedforward network $f(\mathbf{x}, \mathbf{w})$ can approximate a continuous function $g(\mathbf{x}, \mathbf{w})$ across the closed and bounded subset of hidden layers, $\mathbb{R}^{n-1}$, assuming the function has non-constant output for all input values, is non-linear, is bounded within a range of values $|g(\mathbf{x}, \mathbf{w})| \leq \mathcal{H}$, is monotonically increasing, and is continuous over all input values c of its domain (Hornik, 1991; Sodhi, 2014). Nevertheless, *deeper and wider* architectures remain ubiquitous in contemporary deep learning literature underpinned by significant empirical evidence of their superiority when generalising to unseen data relative to single-layer neural networks (He, 2015; Zagoruyko, 2017; Bengio, 2009).

3.1.5 Activation Functions

Neural Networks generally utilise non-linear sigmoid or partially differentiable rectifier activation functions, ϕ, that transform the activation input value received by each artificial neuron, $\mathbf{z}$, calculated as the dot product of input values and neuron synapse weights, into an output activation value passed into adjacent layers. Thus, these functions modulate the strength and amplitude of signals propagated through the network, a critical component of the overall architecture and capacity of a DNN. Recently, maxout models have been developed that stray from the traditional functional design of the activation function. These functions utilise 'maxout units' to learn a piecewise linear activation function for each individual neuron, rather than applying a single activation function to all neurons in the hidden layers. Activation functions provide non-linearity, a critical universal approximation property (UAP) that neural networks must possess to act as function approximators as per the UAT (Hornik, 1991).

The design of a suitable hidden unit activation function, ϕ, applicable for DNNs is an active research area with no definitive theoretical principles. The flexible nature of the DNN architecture allows for the designer to choose one of multiple pre-defined activation functions or use a heuristic function tailored to a given task. Considerations of the functions design include its impact on (1) the speed of training the model, (2) the stability of the learning process and (3) the ability to correctly and efficiently propagate errors through the network during learning. In relation to computational efficiency, there is evidence that the computational cost of training a neural network is intrinsically linked to the complexity of the activation function which weighs heavily on execution time (Liew, 2016). A well-specified activation function will also improve the structural stability of the learning process during training enabling learnable parameters to update in a consistent and stable manner towards a convergence state. Two inherent specifications of the activation function design considered include its boundary parameters and the form of the functions derivative. Wider boundaries of the activation function allow the network to draw representations from a larger range of possible input values. This aids the DNN when backpropagating errors through the network during optimisation, however, there is a trade-off given the tendency of larger representation spaces to produce less stable outputs during training (Puheim, 2014). The form of the activation function derivative has ramifications for propagation efficiency and structural stability of the learning process. Many activation functions introduced in this section have been successful at solving machine learning problems, such as rectifier functions, despite not being continuously differentiable across their domain, a property of the UAT (Hornik, 1991). However, the nature of gradient descent optimisation methods used to train generalisable DNN models is that only an *approximation* of the non-linear function is required not the exact global *solver*.

For each hidden layer activation function utilised in this chapter, Table 3.1.1 defines the input activation, its relevant mathematical expression $\phi(z)$, the range of output values possible over the real number input domain, the form of the functions derivative $\phi'(z)$, and the seminal paper applying the function. Furthermore, functions are defined along dimensions of whether they allow activations in the negative real number region, whether they are bounded along an asymptote, and whether they suffer from vanishing gradient issues. Additionally, illustrations of activation function expressions are provided in Figure 3.1.3.

Input Activation	Expression $\phi(z)$	Range	Derivative $\phi(z)$	Research	Neg. Act.	Bounded	Van. Grad.
Identity	z	$(-\infty,+\infty)$	1	LeCun, 1998	Yes	No	No
Logistic	$\frac{1}{1+e^{-z}}$	(0,1)	$\phi(z)(1-\phi(z))$	Duch, 1999	No	Yes	Yes
Hyperbolic TanH	$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	(−1,1)	$1-\phi(z)^2$	LeCun, 1998	Yes	Yes	Yes
SoftPlus	$\ln[1+e^{z}]$	$(0,+\infty)$	$\frac{1}{1+e^{-z}}$	Nair, 2010	No	(-) region	No
ReLU	$\max(0,z)$	$[0,+\infty)$	$\begin{cases}1 & if\ z\geq 0\\ 0 & if\ z<0\end{cases}$	Nair, 2010	No	(-) region	No
LReLU	$\begin{cases}z & if\ z>0\\ az & otherwise\end{cases}$	$(-\infty,+\infty)$ $a\in(0,1)$	$\begin{cases}1 & if\ z\geq 0\\ a & if\ z<0\end{cases}$	Maas, 2013	Yes	No	No
PReLU	$\begin{cases}z & if\ z>0\\ az & otherwise\end{cases}$	$(-\infty,+\infty)$ a is learnt	$\begin{cases}1 & if\ z\geq 0\\ a & if\ z<0\end{cases}$	He, 2015	Yes	No	No
Bounded ReLU	$\min(\max(0,z),A)$	$[0,A]$	$\begin{cases}1 & if\ 0<z\leq A\\ 0 & otherwise\end{cases}$	Liew, 2016	No	Yes	No
ELU	$\begin{cases}z & z>0\\ a(e^{z}-1) & z\leq 0\end{cases}$	$(-a,+\infty)$ $a\in(0,1)$	$\begin{cases}1 & z>0\\ \phi_{ELU}(z)+a & z\leq 0\end{cases}$	Clevert, 2015	Yes	(-) region	No
Maxout	$z=\max_{x\in\mathbf{x}}(x)$	$(-\infty,+\infty)$	$\begin{cases}1 & if\ \mathbf{x}_i=\max(x)\\ 0 & otherwise\end{cases}$	Goodfellow, 2013	Yes	No	NA

TABLE 3.1.1 – Information regarding hidden layer activation function, ϕ, employed in deep neural networks to transform input activation value, $\mathbf{z}^l$, into output activation value, $\mathbf{x}^l$, which is then forward propagated through to the subsequent layer. Note that Maxout activations are applied simultaneously to *all* neurons in a layer.

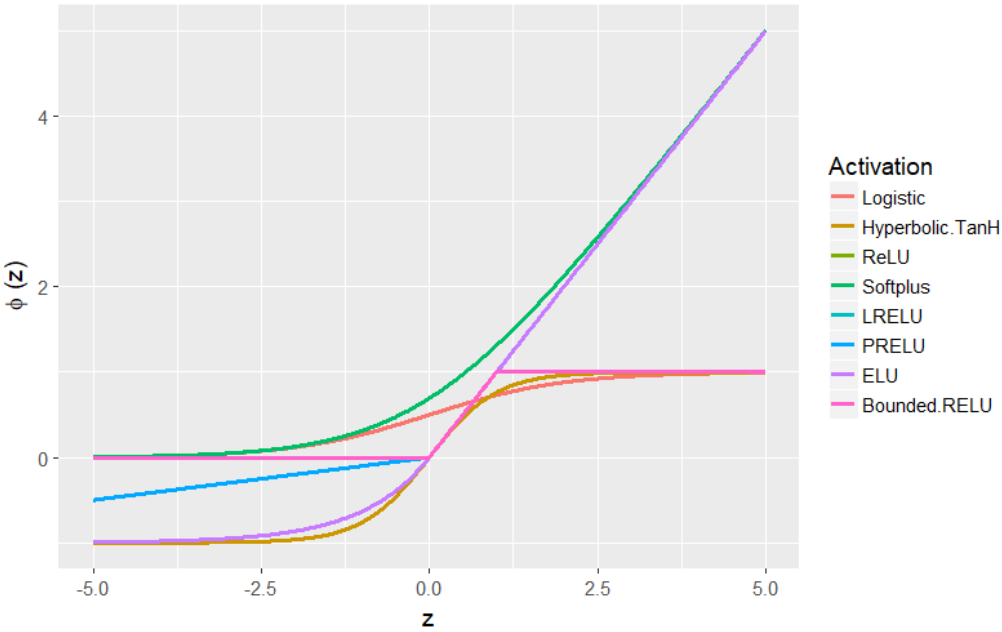

FIGURE 3.1.3 – Expressions for a selection of sigmoidal and rectifier-based hidden layer activation functions, ϕ, over the domain $-5<z<5$.

Output activation functions, φ, provide a final transformation of the activation input value received from the layer adjacent to the output layer resulting in an output prediction value, $\hat{\mathbf{y}}$, for neurons in the output layer. Two considerations when selecting the output activation function is its interrelationship with the cost function, and the desirable form of the output given the machine learning problem being solved. The DNNs employed in this chapter solve the multi-class classification problem of predicting trader actions given a set of LOB features. Thus, the output activation function should be interpretable in a probabilistic sense such that the final neurons, which represent various classes being classified, quantify the DNNs probabilistic prediction for each class. The primary output activation function employed throughout this chapter is the softmax function.

3.1.5.1 Sigmoid Functions

Traditionally, multi-layer neural networks have utilised sigmoidal activation functions for hidden units in the form of logistic, bipolar sigmoid, hyperbolic tangent, or scaled hyperbolic tangent functions. These activation functions are monotonically-increasing bounded functions, differentiable and continuous for all real-valued input activation values and are generally defined by their sigmoid-shaped curve. Sigmoid functions are commonly used to squash artificial neurons' activation values into a range of output values constrained by horizontal asymptotes as absolute activation values approach infinity. These functions are built from principles of statistical mechanics and work in computational neuroscience to model neuronal activity (Cowan, 2016). Logistic sigmoidal functions, ϕ_{LS}, are ubiquitous as an activation in early neural network theoretical work (Rumelhart, 1986; LeCun, 1998), with the function flattening real-valued propagated input values across its domain into an output value with range (0,1). Hyperbolic tangent activation functions, ϕ_{TanH}, are defined as the ratio of hyperbolic sine to cosine functions with respect to the input of the activation function, z. Compared to the logistic function, hyperbolic tangent activations allow for a wider output range (-1,1), and have better symmetric properties about the origin.

The neurological and theoretical foundations of sigmoidal activation functions as good approximations of how information is transferred in biological neurons must be considered against the drawbacks of applying these functions within the mechanics of a neural network. The squashing nature of sigmoid activations that bound output values into a fixed range may have grounds in how biological neurons function, however, the tendency of their gradients to 'saturate' spurs a training condition during learning referred to as the gradient diffusion or 'vanishing gradient problem' (Hochreiter, 1998). Saturation occurs in regions characterised by very small gradients which exist near the asymptotes of sigmoidal activation functions where the input values have high absolute values (Hochreiter, 1998). Vanishing gradient issues arises in DNN models trained with gradient descent type optimisation algorithms which require the backpropagation of cost function errors backwards through the network to calculate the cost function gradient w.r.t a specific neurons weight, which is then used to update that weight. Backpropagation utilises the chain rule to compute a network's gradient which performs a series of multiplicative operations by 'chaining' gradient terms across all adjacent layers towards the output layer when calculating the gradient of a specific neuron. For very deep networks these operations can involve the multiplication of many values. This results in front layer neuron gradients, closer to the input layer, decreasing exponentially as the result of a series of multiplicative operations when applying the chain rule until they ultimately 'vanish' or diffuse.

Ultimately 'vanishing gradients' results in the network learning very slowly which has a negative impact on the convergence properties of DNNs, when sigmoidal activations are employed (LeCun, 1998). Scaled hyperbolic tangent functions are one sigmoidal method of overcoming this problem as they allow for the amplitude, A, and slope, B, variables of the TanH function to be manipulated (LeCun, 1998). By increasing the amplitude of the activation function, the non-saturation region in the middle of the input space increases in size, with the risk of saturation therefore decreasing. Another potential method is to initialise DNN weights into non-saturated regions to ensure gradients propagate well through the network when learning begins. One final technique to overcome vanishing gradients is to employ rectifier activations, which are introduced next, in place of sigmoidal activations.

3.1.5.2 Rectifier Functions

Rectifier activation functions in the form of a ramp function were designed to mitigate the drawbacks of sigmoidal functions including the vanishing gradient problem in deep networks during backpropagation (Glorot, 2011; LeCun, 2015; Ramachandran, 2017). They are based on the principle that weight optimisation is easier when the shape of the activation is piece-wise linear rather than sigmoidal. Rectifier activation functions were first proposed for Restricted Boltzmann Machines (Nair, 2010) based on a rectified linear unit (ReLU) which equals the identity for positive activation value arguments, or zero otherwise. Hence, the rectifier function is piecewise differentiable, with the gradient equal to one when the input is positive, zero when negative, though the function is not differentiable at the origin. Whilst the non-continuously differentiable and anti-symmetric properties of ReLU contradict UAT principles, rectifier functions work well in practice.

Architectures with ReLU functions, ϕ_{ReLU}, have been shown to expedite the rate of convergence for stochastic gradient descent compared to sigmoid functions and improve the accuracy of neural networks classifiers (Krizhevsky, 2012; Maas, 2013), whilst reducing the risk of vanishing gradients given that gradients for positive ReLU arguments are non-contractive. The inherent structure of ReLUs indicates that certain neurons do not always 'fire' because inputs in the negative region are zero which contrasts with sigmoidal activation functions which 'fire' for all input values over the real domain. Thus, ReLU activation functions introduce sparsity into the network, as some nodes become real-zero activations, leading to information flowing in a distributed manner. Contemporary neuroscience research indicates that ReLU behaviour is analogous to that of human neurons which rarely exist in a maximum saturation regime (Lennie, 2003; Douglas, 1995). Furthermore, sparse networks have the advantage of disentangling highly abstract feature combinations deeper in the network to more accurately model the mapping function and reduce overfitting (Bengio, 2012). Small shifts in input feature values will generally have a less amplified impact on the output vector in sparse representations. Sparsity also allows for exponentially more efficient computational training of the neural network at a rate which depends on the number of non-zero features (Glorot, 2011). However, the gains from sparse and distributed networks involve a trade-off against the predictive power of the neural network that gains from more dense representations of the neural structure (Bengio, 2009; Glorot, 2011).

ReLU models with high optimisation learning rates can effectuate instances where a sizable portion of neurons 'die' during training of the neural network (Glorot, 2011). This occurs as gradients saturate at zero when input activation values are negative, resulting in weights during training failing to update, decimating the internal learning dynamics of the neuron. Hence, neurons that 'die' rarely get activated again as they continually output a zero-activation value and gradient. Modifications of the rectifier activation function have been devised to address this problem and extend the advantages of these functions, including the softplus rectifier function, Leaky ReLU (LReLU), Parametric Rectified Linear Unit (PReLU), Exponential linear units (ELU), and Bounded ReLU (BReLU).

Softplus functions, ϕ_{SP}, apply a smooth approximation to the ReLU activation function (Nair, 2010; Dugas, 2001). Softplus activation functions have natural stabilising properties given their continuously differentiable nature and non-zero gradients in negative regions. This allows information in all real input regions of the activation to propagate through the network. These properties contrast with the two inherent structural flaws of ReLU activation functions – they are not continuously differentiable, and

they have a hard-zero output value at the origin. The theoretical benefits derived from employing softplus activation over ReLU require a trade-off of simplicity and sparsity. The smooth S-shape of the softplus activation requires additional computational overheads arising from the more complex operations required in calculating the function and its derivative that slows learning during optimisation (Dugas, 2001). Furthermore, due to its positive activation output across the domain, softplus negates the benefits of sparsity introduced by the zero-activation bound in the negative region of the ReLU activation function. Studies have found that the softplus activation function improves the performance and stabilisation of some speech recognition tasks (Zheng, 2015). However, despite the theoretical foundations of softplus and its generally competitive performance relative to ReLU in many classification tasks (Glorot, 2011), softplus is rarely used in practice (Goodfellow, 2016).

LReLU activations, ϕ_{LReLU}, apply a coefficient of leakage, a, to input values below zero as a way of alleviating the 'dying ReLU' problem by incorporating small negative input values into the learning process (Maas, 2013). This has the effect of the gradient equalling a for input values below zero, and one for input values in the positive regions (Maas, 2013). LReLU activation functions have the desirable property that all neurons activate during the forward propagation of input features through the network, allowing all weights in the network to be updated during the gradient descent learning optimisation process. Essentially, networks with LReLU activations gain from a more robust optimisation process by shifting the mean activation towards zero but lose the benefit of sparsity induced by hard-zero ReLU units.

PReLU are adaptive activation functions, ϕ_{PReLU}, that generalise the LReLU activation by parametrising the coefficient of leakage variable, which means that the variable can be learnt and adapted during the backpropagation process (He, 2015). Studies have found that these activations improve the learning performance over LReLU with minuscule additional computational cost (He, 2015). The coefficient of leakage in PReLU is treated as a learnable parameter, with its own parameter-specific momentum, learning rate hyper-parameters, and is optimised using backpropagation.

ELU are rectifier activation functions, ϕ_{ELU}, that extend ReLUs by allowing for negative output values, similar to LReLU and PReLU but bounded by hyper-parameter value a (Clevert, 2015). ELUs improve learning speed vis-à-vis ReLU's by performing a natural bias shift correction that pushes the mean activation output value towards zero (LeCun, 1998; Clevert, 2015). Furthermore, the ELU activation function encompasses a clear negative saturation plateau at a which bounds the output values in the negative region. The benefit of having a saturation region that is non-existent in the partwise linear models of LReLU and PReLU is that it guarantees a noise-robust deactivation state whereby the gradients are allowed to saturate rather than always be constant in the negative region. This has the impact of making the activation function robust to noisy outliers induced by very large negative values, given that their gradient, and hence the information propagated backwards during training, moves towards zero when there is significant noise. Thus, ELU activation only quantifies the degree of presence of a pattern or phenomena coded in the input data and not the absence of such phenomena (Clevert, 2015). ELUs have been shown to perform with high accuracy in tasks such as image recognition (Clevert, 2015; Shah, 2016) and music classification (Choi, 2017).

Bounded rectifier functions, ϕ_{BReLU}, have their theoretical grounding in the UAT which requires bounded output values as a necessary property for a neural network to be a universal approximator of any

arbitrary continuous function (Liew, 2016). For some image recognition tasks, bounded rectifier functions have been shown to improve classification performance, attaining lower misclassification errors, and reducing training instability of DNN's that utilise non-bounded variants of the rectifier functions (Liew, 2016). Bounded ReLU activation functions address the numerical instability problem (Ma, 2015) and potential resource consumption costs arising from ReLU-type functions that have unbounded outputs for positive argument inputs. This is achieved by bounding the output values by zero when taking negative arguments, and by the upper boundary asymptote determined by the maximum value hyper-parameter A when taking positive arguments.

3.1.5.3 Maxout

Maxout methods modify the conventional implementation of the activation function in deterministic feedforward DNNs by introducing multiple maxout hidden neurons that represent the input activation values as learnable linear functions (Goodfellow, 2013). The maxout network includes a max-pooling layer that envelopes the individual hidden neurons leading to the development of a piecewise linear activation function that can approximate any arbitrary continuous convex activation function on some compact domain according to the Stone-Weierstrass Approximation Theorem (Goodfellow, 2013). Maxout methods scale well for neural networks trained with a regularisation technique called dropout (Srivastava, 2014). The advantages of maxout networks derive from the maxout units' property of rarely having zero gradients, so as to not inhibit training.

Using maxout, any hidden layer l that receives I neuronal input values from the previous layer, $\mathbf{x}^{l-1}$, is transformed into a fully-connected sub-network architecture with K hidden maxout linear extractor neurons for each of the J target output activation neurons in layer l. The maxout hidden layer applicable to the jth output activation neuron is parametrised by K hidden neurons, z_{jk}^{l}, which takes an affine activation transformation of the weighted sum of input activation values, $\mathbf{x}^{l-1}$, weighted by learnable parameter vector, $\mathbf{w}_{jk}^{l}$, and added bias term, $\mathbf{b}_{jk}^{l}$. These two learnable parameters can be represented as three-dimensional tensors where the term $\mathbf{w}_{jk}^{l}$ is taken as the weight vector for the kth neuron in the jth hidden maxout layer, and represents all weights connecting the specific neuron to the input vector $\mathbf{x}^{l-1}$. Evidently, both the number of hidden orthogonal maxout layers, J, or the number of neurons within those layers, K, represent hyper-parameters and may be defined during the design of the maxout neural network architecture depending on the desired complexity. A higher number of maxout units lead to more accurately approximated activation functions, at the cost of additional learnable parameters, whilst the number of maxout units can also differ across different hidden layers. Hence, the value of the hidden maxout layers maxout neuron, z_{jk}^{l}, represents a linear function derived through linear regression parameterised by the new learnable weights and biases of that neuron. Finally, a max-pooling layer is fitted to the network and represented by the maxout activation function, $\phi_{MaxOut}(z_j^l)$, that takes the maximum value across all K hidden maxout neurons for each of the J, which is then propagated as an input into the next hidden layer of the neural network. Maxout activations for each individual neuron can be defined as:

$$\phi_{MaxOut}(z_j^l) = \max_{k\in[1,K]}(z_{jk}^l) \qquad z_{jk}^l = \mathbf{x}^{l-1}\mathbf{w}_{jk}^l + \mathbf{b}_{jk}^l$$

3.1.5.4 Softmax

Softmax output activation functions, $\varphi_{SoftMax}$, are commonly employed in the output layer for multi-class classification problems involving C different classes where each class is represented by a single output neural unit. Softmax exponentiates and normalises the input activation value for neuron of class c, z_c, before calculating the output predictor, $p(y = c|\mathbf{x})$, modelled using a Multinoulli distribution over y conditioned on $\mathbf{x}$ for the valid interval (0,1). This value represents the prediction of the neural network for a specific class over another, given feature vector $\mathbf{x}$. Thus, the set of softmax output values are analogous to a categorical probability distribution which is then inputted into the cost function during learning. Softmax is a continuous and differentiable activation function that represents a generalisation of the sigmoidal logistic function to a discrete variable with C values by fitting a Multinoulli probability distribution where all the output values sum to one. Whilst softmax activation functions in hidden layers are less commonly used, their utility for use in output layers derives from their compatibility with the Cross-Entropy cost function, introduced in the next section. Softmax activations can be defined as:

$$\varphi_{SoftMax} = \frac{e^{z_j}}{\sum_J e^{z_j}}$$

3.1.6 Cost Functions

Defining a cost function, $\mathcal{L}$, which compares the distribution of the prediction values outputted by the DNN model, $\hat{\mathbf{y}}$, with the correct classification, $\mathbf{y}$, is a critical design element of DNN machine learning classification models. The DNN learning algorithms implemented in this chapter update the learnable parameters of the network, $\mathbf{w}$, a way that minimises the cost function of the system. That is, the difference between the models' prediction of an AT firms' trading actions given the set of raw LOB features, $\hat{\mathbf{y}}$, and the actual action executed by the firm, $\mathbf{y}$. This method is built on the assumption that the DNN is 'sufficiently powerful and accurate' (Saerens, 2002), such that one can model the conditional distribution of the learned output vector for discrete class c, $\hat{\mathbf{y}}_c$, as an estimation of the Bayesian a posteriori probability vector representing the true class, $\mathbf{p_w}(\mathbf{y}_c|\mathbf{x})$, conditional on input feature vector $\mathbf{x}$.

Mathematically, cost functions $\mathcal{L}$, are parametric models that quantify the accuracy or error of the neural network for a specific training instance data point n, $\langle \mathbf{w}_n, \mathbf{x}_n, \mathbf{y}_n \rangle$. This requires propagating the input feature vector, $\mathbf{x}_n$, through the network, multiplied by the corresponding weight tensor $\mathbf{w}_n$, to calculate the model's output classifier vector, $\hat{\mathbf{y}}_n$, which is then compared to the correct training instance classifier, $\mathbf{y}_n$. The cost function represents the dimension along which a learning algorithm attempts to optimise during the training of a DNN. Hence, it is generally required that the function is differentiable and definable as a function of the neural network outputs. Training a DNN requires optimisation of the learnable weight parameters, $\mathbf{w}$, in a way that minimises the defined cost function, $\mathcal{L}(\mathbf{w})$, which can be measured as an average over N training examples data points, $\mathcal{L}(\mathbf{w}) = \frac{1}{N}\sum_{n \in N} \mathcal{L}_n(\mathbf{w}_n)$. Henceforth, for clarity purposes the cost function, $\mathcal{L}(\mathbf{w})$, is taken to represent the cost function for a single training instance, mini-batch, or batch, of training sample size N, depending on the context of the discussion.

Non-linear neural networks with multiple hidden layers are generally trained iteratively using some learning algorithm, such as stochastic gradient descent, to update weights in a way that minimises the cost function. However, neural networks learn an approximation of the input-output non-linear mapping

function rather than an exact solution, thus, whilst cost functions must be both differentiable and non-convex, global convergence properties are not a requirement. The modular nature and flexibility in designing the architecture of DNNs allows for a multitude of applicable cost functions, with each having a commensurate impact on the learning dynamics and robustness level of the model. Furthermore, choice of the cost function also has ramifications for the predictive performance, computational learning efficiency, and convergence properties of models under a resource regime with bounded limitations (Janocha, 2017). Principally, the mechanics of the cost function should match the probabilistic assumptions of the DNN model which are dependent on the type of non-linearity applied through the output activation function, φ. Figure 3.1.4 provides a graphic representation of several cost functions employed in this chapter and explained in the following sections.

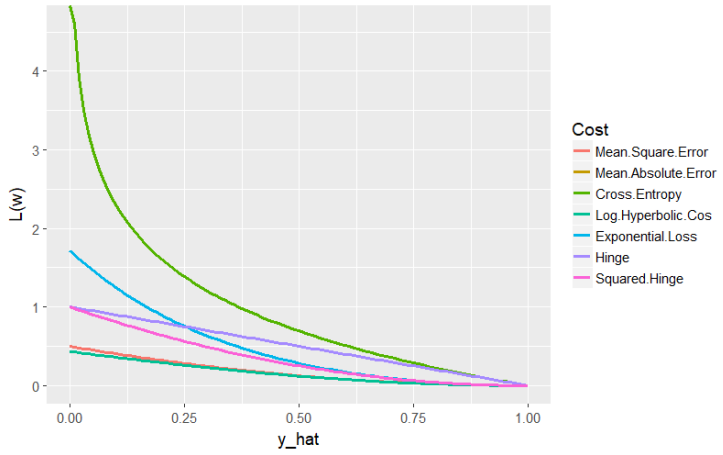

Figure 3.1.4 – Cost function graphs of the loss for model output values of $0 \leq \hat{\mathbf{y}} \leq 1$ for several cost functions employed throughout this thesis.

3.1.6.1 Mean Error

Mean squared error (MSE) is an arbitrary cost function, $\mathcal{L}^{MSE}$, with roots in statistical regression that provides a proxy for the quality of a neural network as a predictor. MSE measures the average of the squared Euclidean distance between the sample output classifier prediction of the DNN, $\hat{\mathbf{y}}_c$, and the correct classifier, $\mathbf{y}_c$, for the class c neuron in the output layer. MSE extends the non-quadratic mean absolute error (MAE) cost function, $\mathcal{L}^{MAE}$, which simply takes the absolute distance between the predicted and correct classifiers as a representation of the error. The advantage of MSE over MAE resides in its more stable form, given the continuous nature of its regression parameters, and singular solution. MAE also requires more complex methods for computing the gradient, such as linear programming. The problem with MSE cost functions is their relative lack of robustness, given their susceptibility to compounding the negative effects of outliers. Both forms of the cost function have practical deficiencies when poorly chosen activation functions and gradient descent-based optimisation techniques are employed, given that they generally lead to very small gradients which vanish when backpropagated through the network. MSE generally performs better using an identity output activation function rather than one with a probabilistic interpretation such as softmax (Goodfellow, 2016). MSE and MAE cost functions are defined as:

$$\mathcal{L}^{MSE} = \sum_{c \in C} \frac{1}{2} |\hat{\mathbf{y}}_c - \mathbf{y}_c|^2 \qquad \mathcal{L}^{MAE} = \sum_{c \in C} |\hat{\mathbf{y}}_c - \mathbf{y}_c|$$

3.1.6.2 Cross-Entropy & Kullback-Leibler

Cross-entropy (CE) cost functions, $\mathcal{L}^{CE}$, also referred to as negative log-likelihood functions, are ubiquitous within DNN academic and practitioner literatures when training classifier DNNs (de Boer, 2005; Janocha, 2017). For probabilistic DNN models that utilise the CE cost function, learning algorithms optimise weights by minimising the negative log-likelihood. The CE cost function can be viewed as a representation of the negative summed cross-entropies between the log-likelihood of the probabilistic models' learned output distribution, $\hat{\mathbf{y}}_c$, and the true output classifier, $\mathbf{y}_c$, for all $c \in C$ class neurons in the output layer. The entropy of any discrete random variable X, $H(X)$, quantifies the level of uncertainty in the probability distribution of that variable, with distributions that bias towards a single event having relatively low uncertainty and thus, minimal entropy. For a DNN, cross-entropy of the cth neuron, $H(\mathbf{y}_c, \hat{\mathbf{y}}_c)$, is a measurement of the degree to which a training data sample output for class c, $\mathbf{y}_c$, provides evidence or support towards the neural networks parametric output value, $\hat{\mathbf{y}}_c$. The CE cost function quantifies the error of the neural network in totality by summing the cross-entropy divergence between the learned output posterior probability distribution of the model, $\hat{\mathbf{y}}$, and the true observed output, $\mathbf{y}$, whose elements equal one for the correct class and zero otherwise. These cost functions are the natural choice for multi-class DNNs with softmax output activations that compress neuron activations into a probabilistic interpretation, utilising a Multinoulli distribution.

Optimisation of DNNs utilising categorical CE cost functions involve perturbing weight parameters in a way that minimises the negative log likelihood between model predictions and the classifier for all $c \in C$:

$$\mathcal{L}^{CE}(\mathbf{w}) = -\sum_{c \in C} \mathbf{y}_c ln(\hat{\mathbf{y}}_c) \qquad \hat{\mathbf{y}}_c \geq 0 \; \forall c \quad \sum_{c \in C} \hat{\mathbf{y}}_c = 1 \quad \mathbf{y}_c = \begin{cases} 1 & \textit{if sample } \mathbf{y}_n \textit{ belongs to class } c \\ 0 & \textit{otherwise} \end{cases}$$

For the special case where the neural networks output predictions conditional probability distribution follows a Gaussian distribution, $\mathcal{N}$, rather than a Multinoulli distribution, such that the outputs of the model, $\hat{\mathbf{y}}_c$, are conditionally i.i.d for a given input vector, $\mathbf{x}$, the cost function reduces to the MSE. Furthermore, CE loss functions can also be equated with Kullback-Leibler divergence cost functions, $\mathcal{L}^{KL} = KL(\mathbf{y}_c \| \hat{\mathbf{y}}_c)$, which measure the divergence between the probability distribution of the true class, $\mathbf{y}_c$, and the expected distribution of the predictor class, $\hat{\mathbf{y}}_c$, by computing both the cross-entropy, $H(\mathbf{y}_c, \hat{\mathbf{y}}_c)$, and entropy of the predictor classifier, $H(\mathbf{y}_c)$, which is assumed constant.

3.1.6.3 Log Hyperbolic Cosine

The log hyperbolic cosine cost function, $\mathcal{L}^{LHC}$, quantifies the distance between the DNN model estimator, $\hat{\mathbf{y}}_c$, and the actual classifier, $\mathbf{y}_c$, by measuring the log of the hyperbolic cosine difference between the two values. The cost function works in a similar manner to the MSE by comparing the difference in values between the actual and model classifier values in a quadratic framework, however, the hyperbolic cosine shifts one unit from the origin and has a flatter shape for small values of $\hat{\mathbf{y}}_c - \mathbf{y}_c$, and a sharper shape for higher values. The resulting impact is that small divergences in predicted values by the DNN from their actual values are punished similar to MSE, whilst outlier predictors that strongly diverge from the correct class are not punished as much as MSE. Log hyperbolic cosine cost functions are defined as:

$$\mathcal{L}^{LHC} = \sum_{c \in C} ln[\cosh(\hat{\mathbf{y}}_c - \mathbf{y}_c)]$$

3.1.6.4 Exponential Loss

Exponential cost functions, $\mathcal{L}^{EXP}$, are flexible parametrised functions that are able to sufficiently emulate the classic MSE and CE cost functions, whilst attaining desirable convergence and robustness properties for specific types of problems, especially those with significant outlier presence. The relatively steep shape of the cost function indicates that predictor outliers outside the region around the correct classifier are punished sharply as the values diverge. Exponential loss cost functions are defined as:

$$\mathcal{L}^{EXP} = \sum_{c \in C} \tau e^{\left[\frac{1}{\tau}(\hat{\mathbf{y}}_c - \mathbf{y}_c)^2\right]}$$

3.1.6.5 Hinge Loss

Hinge loss cost functions, $\mathcal{L}^{HL}$, compute a max function that takes as arguments zero and the difference from one of the multiplier between the actual output, $\mathbf{y}_c$, and the DNN predictor, $\hat{\mathbf{y}}_c$. Squared hinge loss cost functions, $\mathcal{L}^{SHL}$, take the square of the hinge loss and have achieved good results relative to other cost functions in DNNs from the perspective of classification performance, accuracy, and speed of convergence (Tang, 2013; Janocha, 2017). Hinge loss is more commonly used for maximum-margin classification in support vector machines (Rosasco, 2004). The 'margin' for the cost function is defined as one which bounds the maximum error when the predictor classifier of the DNN, $\hat{\mathbf{y}}_c$, and actual output, $\mathbf{y}_c$, have the same sign. However, when the predictor and actual output have opposite signs, the hinge loss increases linearly and the squared hinge loss quadratically. Convex hinge loss functions in the binary case do not have a gradient at zero, which creates difficulties for stochastic gradient descent-based optimisation algorithms that require real-valued gradients across the domain, an issue that is allayed by the use of squared hinge loss cost functions. Theoretically, the hinge loss cost functions also provide a degree of robustness to outliers far from the margin or decision boundary, that is not the case in the MSE and MAE cost functions (Janocha, 2017). Hinge loss and squared hinge loss cost functions are defined as:

$$\mathcal{L}^{HL} = \sum_{c \in C} \max[0, (1 - \mathbf{y}_c \hat{\mathbf{y}}_c)]$$

$$\mathcal{L}^{SHL} = \sum_{c \in C} \max[0, (1 - \mathbf{y}_c \hat{\mathbf{y}}_c)]^2$$

3.1.7 Weight Initialisation

Solving complex non-linear functions with non-convex first-order optimisation algorithms requires the application of weight initialisation strategies that improve the convergence properties of DNNs in a way that minimises the cost function, improves computational efficiency and lowers the generalisation error of the model. Arbitrary random initialisation can compound the issue of exponentially smaller variances in deeper layers leading to minuscule gradients as the absolute value of inputs becomes large. The result of this random weight initialisation is a slowing down of the backpropagation of errors, stunting the learning process and ultimately leading to slower or possible failed convergence (Mishkin, 2015). Initialisation of weights at the beginning of training is a critical design element that impacts the convergence capabilities of a network, requiring a careful balancing act that sets weights within an

equilibrium range. Weights should be set low enough to dampen the risk of exploding output values or saturated gradients that occur when large initial weights lead to high errors and gradients during forward and back propagation, respectively. Furthermore, weights need to be initialised to large enough values to break the symmetry of hidden layer neurons, ensuring neurons activate and learn in initial epochs, allowing them to maintain signals deeper in the network.

Various heuristic (Glorot, 2010; He, 2015), theoretical (Saxe, 2014) and data-driven (Mishkin, 2015) weight initialisation strategies have been devised with their performance and pertinence largely dependent on the type of activation function used within the DNN. These methods have largely built upon previous means of initialising networks with Gaussian noise with zero mean and small fixed standard deviation, a process that is not successful at training very deep networks (Simonyan, 2014). He (2015) and Glorot (2010) heuristic initialisation strategies involve drawing randomly from Gaussian or Uniform distributions, respectively, to break the symmetry of layers by ensuring neuron values diverge, whilst also equalising the statistical distribution properties of all layers. Saxe (2014) develops an initialisation method theoretically rooted in linear networks that applies a scaling gain factor to weights initialised as random orthogonal matrices. Mishkin (2015) uses a data-driven iterative algorithm that initialises weights by combining the orthogonal matrix method with layer-wise Batch Normalisation (Ioffe, 2015). Biases are generally initialised to zero or one (Bengio, 2012).

3.1.7.1 Xavier Initialisation

Xavier weight initialisation methods are designed to maintain the scale of the gradients deep into the network by determining an appropriate variance for the distribution of the weights for a specific neuron being initialised based on the number of connected feed-in or feed-out neurons (Glorot, 2010; Jia, 2014). Xavier initialisation was designed to counter the issue of saturation that occurs in hidden layers when deep networks with sigmoidal non-linear activations are initialised with random or zero weights. The objective is to ensure that propagation occurs near the centre of the activation rather than at saturation points. Xavier methods are built under the assumption of linearity between layers, which is violated in the case of the DNN given that values are propagated through the network based on a series of non-linear chained matrix multiplications, though it serves as a useful heuristic in the non-linear case. The method has been shown to work for sigmoidal activations (Glorot, 2010) but fail in the case of deep networks with a rectifier activation-based architecture (He, 2015; Kumar, 2017; Mishkin, 2015).

Essentially, the Xavier method works from the principle that at initialisation the variance of the input values should be the same as the variance of the output values with mean zero given i.i.d assumptions. Solving mathematically, this requires that the variance of initialised weights should equal $\frac{1}{a}$, where a is the number of connected neurons from the previous layer. Glorot (2010) provides a compromise to ensuring that the variances of both the *forward* input and *backward* output gradients are similar by averaging over the number of forward propagated input neurons and backpropagated output neurons before being scaled to ensure unit variance. The weight connecting neurons in two non-output layers, w_i, is sampled from a uniform distribution, $\mathcal{U}[-1,1]$, then scaled depending on the number of neuronal connections with the previous, α_{in}, and sequential, α_{out}, layers of the neural network, leading to:

$$w_i^{Xavier,\mathcal{U}} \sim \mathcal{U}[-1,1] \cdot \sqrt{6/(a_{in} + a_{out})}$$

3.1.7.2 He Initialisation

The non-linearity and asymmetric distributional properties of scale-invariant non-saturating ReLU type activation functions are accounted for when applying the He weight initialisation for DNN architectures (He, 2015; Kumar, 2017). The fact ReLU activations zero out negative input activation values informs the theoretical foundation of the smaller variance in the He initialisation values which removes approximately half of the variance vis-à-vis the Xavier initialisation technique. This strategy has been shown to improve convergence for extremely deep networks with rectifier activations (He, 2015). He initialised weights are drawn from a Gaussian normal distribution with mean zero and unit variance, $\mathcal{N}(0,1)$, and scaled by a function of the number of input neurons, a_{in}, such that:

$$w_i^{He,\mathcal{N}} \sim \mathcal{N}(0,1) \cdot \sqrt{2/a_{in}}$$

3.1.7.3 Orthogonal Matrix Initialisation

Saxe (2014) provides a theoretical approach for initialising weights using random orthogonal matrices that are scaled by a gain factor, g. An orthogonal real matrix, $M \in \mathbb{R}^{j \times k}$, is composed of a series of j orthonormal column vectors, $\mathbf{m}_1, \dots, \mathbf{m}_k$, with unit norm, $\|\mathbf{m}_i\| = 1$, that are mutually orthogonal, $\mathbf{m}_h \mathbf{m}_i = 0$, where $h \neq i$, and maintains the property of left invertibility such that the Gram matrix, $M^T M$, is equal to the identity matrix, $M^T M = I$. In the case of a linear network, this technique treats network layers as a series of linear chained matrix multiples during initialisation. Applying the initialisation technique to non-linear networks requires the random orthogonal matrices to be scaled by the gain factor, g, which accounts for the non-linearity in connections between DNN layers when setting weights. Choosing this factor balances the requirement of setting it large enough to ensure the input activations signals can propagate forward through the network without decaying to zero, and small enough to ensure gradients do not saturate at asymptote boundaries during backpropagation.

3.1.7.4 Layer-Sequential Uniform Variance Initialisation

Layer-sequential uniform variance (LSUV) is a strategy for DNN weight initialisation that first initialises weights based on the Saxe (2014) random orthogonal matrix technique to de-correlate layer activation values before using an iterative pre-learning algorithm that updates initialised weights after each pass using batch normalisation (Ioffe, 2015) in a way that maintains unit variance for weights in each layer (Mishkin, 2015). Essentially, after the pre-initialisation of weights using the random orthogonal matrix technique, the LSUV algorithm then propagates a mini-batch through the network and searches for the first layer where the variance of the output activation values, B_l, is too large or small, given an acceptable tolerance threshold, Tol_v. This threshold needs to be set large enough to ensure enough iterations are performed and small enough to ensure the loop has a logical stopping criterion. The algorithm then updates the layer weights for each batch trial by normalising them based on the standard deviation of the output activation values for that layer. Normalisation is performed on the actual output values rather than activation input values, similar to batch normalisation. This process is performed iteratively until the activation output variances all converge to within an acceptable threshold close to one, or the T_{max} iteration level is reached, at which point network weights, w_i, are initialised and training can begin.

3.1.8 First-Order Optimisation Algorithms

Training multi-layer non-linear DNN's with differentiable neuron activation functions and a high-dimensional feature space generally involves using a gradient descent-based learning algorithm to optimise internal learnable parameters, the weight vector, $\mathbf{w}$, and the bias vector, $\mathbf{b}$, by perturbing their values a way that minimises a defined nonconvex cost function, $\mathcal{L}$. Machine learning optimisation tasks draw from various strands of computation optimal control theory. However, optimising a DNN through training does not require finding an exact local minimum but rather involves convergence to an acceptable solution where the optimal combination of weights, $\mathbf{w}^*$, minimises the cost function to a specified threshold generalisation error value. The optimal weight matrix provides the most accurate approximation, f^*, of the correct non-linear function that explains the training dataset mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$.

Learning algorithms, also referred to as optimisation algorithms, are almost ubiquitously grounded in gradient descent optimisation theory and can generally be segmented into whether they minimise the cost function by taking either the first or second-order derivative w.r.t the learnable parameters. First-order learning algorithms minimise the cost function by perturbing learnable parameters in the negative downhill direction from the functions gradient, $\nabla_{\mathbf{w}}\mathcal{L}$, noted as the nabla of the cost function parametrised by the weights. The speed of parameter updates is generally regulated by the learning rate hyper-parameter a. In the context of DNNs, the term gradient is manifestly a multi-variable generalisation of all individual partial derivatives of the cost function w.r.t each of the learnable parameters, $\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}$, thus, $\nabla_{\mathbf{w}}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{w}_1} \quad \cdots \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}_n}\right]$. The gradient explains the shape of a particular point on the cost functions error surface, which is stored in the form of a Jacobian Matrix. Conversely, second-order algorithms require additional computational expense to compute the Hessian Matrix which provides more explicit information regarding the curvature of the error surface. These second-order algorithms generally converge in fewer iterations than first-order methods, however, each iteration is generally computationally more expensive, taking significantly longer to process than the first-order methods on average (Martens, 2010; Sutskever, 2013).

This chapter focusses on first-order stochastic gradient descent (SGD) and adaptive SGD optimisation algorithms when building DNNs to minimise differentiable cost functions and update weights. Adaptive SGD algorithms use momentum and learning rate annealing techniques to improve performance and computational efficiency vis-à-vis SGD. Learning annealing is the process of dynamically adapting the learning rate parameter for a specific schedule or training iteration. Adaptive SGD optimisation algorithms that use learning annealing are able to stabilise oscillations in weight updates around a local minimum region by reducing the learning rate parameter over iterations, with weight updates decreasing as the size of the past few gradients increase.

The following sections explain the complex issues to be considered during gradient descent optimisation, including suboptimal local minima and saddle points, before introducing stochastic gradient descent and its various adaptive extensions.

3.1.8.1 Common Issues with Gradient Descent

Myriad DNN learning algorithms have been developed to address limitation's inherent in the learning process, such as sub-optimal local minima and saddle points which relate to the statistical properties of the cost functions error surface, and gradient diffusion issues. Significantly deep neural networks can have an error surface with near infinite *suboptimal local minima* solutions which correlate positively with the dimensionality of the dataset, reducing the probability of cost function convergence to a global minimum. However, given that DNNs are non-identifiable in many of their model parameters, primarily due to architecture and design, many of these local minima compute a near identical cost function value and hence, are a non-problematic form of convexity. Furthermore, network depth increases it becomes exponentially more expensive and unnecessary to recover a global minimum from some local minima that already nearly approximates the lowest cost function, without overfitting (Choromanska, 2015). The tendency of learning algorithms to become trapped in *saddle points* surrounded by high error plateaus is a more omnipresent problem occurring during non-convex cost function optimisation for deep networks with high-dimensional surfaces (Dauphin, 2014). Whilst gradient descent algorithms augment learnable parameters away from saddle points based on the parameter manifolds curvature, there is a risk that the slow acceleration of the gradient along the plateau will stunt learning speed and truncate network performance. This issue becomes prevalent for networks with many layers, given that the ratio of saddle points to local minima increases exponentially with the depth of the network (Dauphin, 2014). Finally, *vanishing or exploding gradients* can occur when using backpropagation to train very deep neural networks that are prone to saturation of gradients at very low or high values (Hochreiter, 1998). Backpropagation trains these weights by multiplying gradients errors back through the network and can lead to exponentially high or low weight updates further along the network, potentially stunting learning when gradients vanish or explode.

3.1.8.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a first-order optimisation algorithm with the objective of minimising the total error of the neural networks differentiable cost function, $\mathcal{L}(\mathbf{w})$. SGD represents the benchmark tool for efficiently optimising neural networks with high-dimensional learnable parameters (Hinton, 2006; Krizhevsky, 2012). The first part of this section explains how the SGD optimisation algorithm selects a batch of n independent random training sample, $\langle \mathbf{w}_1, \mathbf{x}_1, \mathbf{y}_1 \dots \mathbf{w}_n, \mathbf{x}_n, \mathbf{y}_n \rangle$. The next three parts explain the process of forward propagation, backpropagation, and weight updates, which together encompass the SGD algorithm. Input feature vector $\mathbf{x}$ for each sample in a batch are forward propagated through the network to formulate a predictor output, $\hat{\mathbf{y}}$, which is compared to the actual output, $\mathbf{y}$, within the framework of a cost function, $\mathcal{L}$. Backpropagation is then performed by propagating information regarding the gradients of the cost function w.r.t individual weights, $\nabla_{\mathbf{w}}\mathcal{L}$, backwards through the network, using techniques including the chain rule, automatic differentiation and dynamic programming (Werbos, 1974; Rumelhart, 1986; LeCun, 1998). The final component of the SGD optimisation algorithm is the iterative update for each individual synaptic weight, $w_i^{m+1} \leftarrow w_i^m$, after completion of the $m + 1^{th}$ training iteration. Weight updates are a function of their previous iteration weight, w_i^m, the partial derivative of the cost function w.r.t the specified weight, $\frac{\partial \mathcal{L}}{\partial w_i}$, calculated during backpropagation and a learning rate hyper-parameter, α^m. Training is conducted by sweeping the

training dataset through the network multiple times, also referred to as training across multiple epochs, generally done so in a random order to improve convergence (Bengio, 2012).

Updates to the learnable parameters of a network during SGD requires the calculation of an unbiased stochastic approximation of the cost functions average gradient for either a single, mini-batch or whole batch of training samples. Single sample SGD updates weights after each sample is fed through the network, whilst mini-batch and batch gradient descent requires the gradient of multiple or all training samples, respectively, to be calculated and the error averaged before weights are updated. Each sample is drawn i.i.d randomly from the training dataset to not bias the gradient estimate. It is required to find a balance between the lower generalisation error attained when selecting smaller mini-batches and the superior convergence stability of networks with larger batch sizes. Stable convergence occurs when the variances in gradient estimates are dampened, suppressing cost function oscillations, resulting in mild weight fluctuations. However, the advantage from stochasticity in the cost function estimation in smaller batch networks arises from the natural regularising effect that sample specific noise has on the training process (Wilson, 2003). Smaller sample sizes also allow for more exploration in parameter space. Furthermore, mini-batch gradient descent is generally preferred over batch learning due to its relative computational simplicity and beneficial convergence properties, particularly for convex loss functions and for datasets with redundancy (LeCun, 1998). For example, the noise present in mini-batch learning updates has the advantage of shifting weights from a sub-optimal local minimum (Heskes, 1993). The wide adoption of parallel computing infrastructure and alignment of these methods with common network optimisation programming languages has also led to the ascendancy of mini-batch training where sample gradients can be computed in parallel.

Forward propagation involves the computation of the cost function, $\mathcal{L}(\mathbf{w})$, by propagating a mini-batch training sample, represented by the input feature vector $\mathbf{x}$, through the neural network, multiplying values by synaptic weights, $\mathbf{w}$, adding biases, $\mathbf{b}$, and applying relevant hidden, $\boldsymbol{\phi}$, and output, $\boldsymbol{\varphi}$, activation functions to the values moving through the network. Once this process is complete one can calculate the activation value $\mathbf{z}_j^l$, of the jth neuron in layer l, activation output value $\mathbf{x}_j^l$, and the jth neuron in the output layer L, $\hat{\mathbf{y}}_j^L$, as follows:

$$\mathbf{z}_j^l = \sum_i \left[\left(\mathbf{w}_{ij}^l \mathbf{x}_i^{l-1}\right) + \mathbf{b}_j^l\right] \qquad \mathbf{x}_j^l = \boldsymbol{\phi}\left(\mathbf{z}_j^l\right) \quad \hat{\mathbf{y}}_j^L = \boldsymbol{\phi}\left(\mathbf{z}_j^L\right)$$

The objective of SGD optimisation is to minimise the cost function error which is achieved by updating the networks synaptic weights in the opposite direction of the cost functions gradient w.r.t the weight vector, $\nabla_{\mathbf{w}}\mathcal{L}$, composed of all partial derivatives, $\frac{\delta\mathcal{L}}{\delta w_i}$, and calculated using backpropagation. The negative gradient represents the direction of the steepest descent for the cost function which naturally produces a path that leads to a local minimum. For clarity in this section, backpropagation begins in the output layer of the network, with the process moving backwards through the network layer-by-layer until the input layer is reached. Furthermore, the calculation of individual neuron weights' partial derivatives during backpropagation can be completed in parallel for all neurons in a single hidden layer. This is primarily due to the fact that partial derivatives of each neuron do not have any dependence on the weights or input neurons in the current layer or any of the layers towards the input layer of the network.

Backpropagation requires computing the partial derivative of the cost function w.r.t each weight in the network, $\frac{\partial \mathcal{L}}{\partial w_{ij}^{l}}$, where weight w_{ij}^{l} connects the jth artificial neuron in the lth layer to the ith neuron in the previous layer. Applying the chain rule allows one to analytically compute derivatives of the composite functions by decomposing the partial derivative into its component form:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{l}} = \frac{\partial \mathcal{L}}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{ij}^{l}}$$

The first term in the chain, $\partial \mathcal{L} / \partial z_{j}^{l}$, is the partial derivative of the cost function, $\mathcal{L}$, w.r.t the activation value for neuron j in layer l, z_{j}^{l}, which can be represented as the recursive error signal for that neuron, δ_{j}^{l}. Essentially, the error signal is a heuristic measure of the proportion of error in the output that is attributable to an individual neuron. For example, a large absolute error signal signifies that only a minor shift in the activation value, ∂z_{j}^{l}, is required to induce significantly large positive or negative movements in the cost function value, indicating the relatively higher attribution of the neuron to the cost function error.

Following the forward propagation of a mini-batch training sample through the network the error signal or partial derivative, $\boldsymbol{\delta}^{L}$, of the output neuron is computed by comparing the actual output of the neuron, $\mathbf{y}$, to the neural networks prediction, $\hat{\mathbf{y}}^{L}$, through the framework of the cost function. Determining error signals in hidden layers involves backpropagating error signals iteratively layer-by-layer in a backwards direction from the output layer towards the input layer. The theoretical rationale for backpropagation rests on the fact that for a feedforward DNN, changes in activation values of a neuron only drive variations in error signals for neurons layers *closer to the output layer*. For example, fluctuations in neuron j in the lth layer, z_{j}^{l}, lead to values of neurons in later layers being influenced, such as the kth neuron in the $(l+1)^{th}$ layer, z_{k}^{l+1}, but *not* neurons in the previous layers closer to the input layer. This concept allows for the cost function to be expressed as a function of activation value, z_{k}^{l+1}, which itself is a function of the previous layers activation values, z_{j}^{l}, such that, $\mathcal{L} = \sum_{k} \mathcal{L}\left(z_{k}^{l+1}\left(z_{j}^{l}\right)\right)$. Thus, the first partial derivative term of interest in backpropagation, $\partial \mathcal{L} / \partial z_{j}^{l}$, can be defined for hidden layers where $k \neq j$ as:

$$\frac{\partial \mathcal{L}}{\partial z_{j}^{l}} = \delta_{j}^{l} = \sum_{k} \frac{\partial \mathcal{L}}{\partial z_{k}^{l+1}} \frac{\partial z_{k}^{l+1}}{\partial z_{j}^{l}} = \sum_{k} \delta_{k}^{l+1} w_{jk}^{l+1} \phi'\left(z_{j}^{l}\right) \qquad where\ \frac{\partial \mathcal{L}}{\partial z_{k}^{l+1}} = \delta_{k}^{l+1} \quad and \quad \frac{\partial z_{k}^{l+1}}{\partial z_{j}^{l}} = w_{jk}^{l+1} \phi'\left(z_{j}^{l}\right)$$

And for output layers where $k = j$ as:

$$\frac{\partial \mathcal{L}}{\partial z_{j}^{L}} = \delta_{j}^{L} = \frac{\partial \mathcal{L}}{\partial y_{j}^{L}} \varphi'\left(z_{j}^{L}\right)$$

Evidently, the backpropagation algorithms application of automatic differentiation and dynamic programming allows for the error signal, δ_{j}^{l}, to be computed as the dot product of the weights connected to the adjacent layers, w_{jk}^{l+1}, and the recursive error signal in the adjacent layer, δ_{k}^{l+1}, multiplied by the derivative of the activation function w.r.t the activation value in the current layer, $\phi'\left(z_{j}^{l}\right)$. The backpropagation algorithm initialises in the output layer and utilises dynamic programming to recursively solve for gradients in adjacent layers in a continuous backwards process until the input layer is reached, reducing computational cost by caching intermediate partial derivatives.

The chains second term, $\partial z_j^l / \partial w_{ij}^l$, represents the partial derivative of the activation value for the jth neuron, z_j^l, w.r.t the weight connecting that neuron to neuron i in the previous layer, w_{ij}^l. This term is mathematically expressed by a series of summands of weighted inputs, $w_{jn}x_n$, for all n neurons in the network that connect and feed into neuron j, from a *backwards* perspective, which may be a substantially large number of neurons for complex DNNs. However, when calculating the partial derivative all components of this summand are dropped except for the one related to that specific weight, $w_{ij}x_i^{l-1}$, with the partial derivative simply reducing to x_i^{l-1}, for all hidden layers, such that:

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^l}\sum_n w_{jn}x_n = \sum_n w_{jn}\frac{\partial x_n}{\partial w_{ij}} + \frac{\partial w_{jn}}{\partial w_{ij}}x_n = x_i^{l-1} \qquad where\ \frac{\partial x_n}{\partial w_{ij}} = 0 \quad and\ \frac{\partial w_{jn}}{\partial w_{ij}} = 1 \quad if\ n = i$$

Completing the backpropagation process, we compute the partial derivative of the cost function w.r.t the weights, $\frac{\partial \mathcal{L}}{\partial w_{ij}^l}$, by combining the above terms using the chain rule to yield:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \delta_j^l x_i^{l-1} \quad for\ j^{th}\ neuron\ in\ the\ l^{th}\ layer, where \qquad \delta_j^l = \begin{cases} \sum_k \delta_k^{l+1} w_{jk}^{l+1} \phi'(z_j^l) & (hidden\ layers) \\ \frac{\partial \mathcal{L}}{\partial y_j^L}\phi'(z_j^L) & (output\ layers) \end{cases}$$

The SGD optimisation algorithm performs an update to each individual synaptic weight, w_i^{m+1}, by multiplying the partial derivative of the cost function w.r.t that weight, $\frac{\delta \mathcal{L}}{\delta w_i}$, derived using backpropagation, by a learning parameter, α^m, then subtracting this value from the original weight, w_i^m. Optimisation involves an iterative process where weights are updated across multiple mini-batch training samples and epochs until the cost function error reaches some threshold, ε. Accounting for the newly equated partial derivative, the weight update rule for the $m+1$ iteration for weight w_{ij}^l can be defined as:

$$w_{ij}^{l,m+1} \leftarrow w_{ij}^{l,m} - \left(\alpha^m * \frac{\partial \mathcal{L}}{\partial w_{ij}^l}\right) = w_{ij}^{l,m} - \left(\alpha^m \delta_j^l x_i^{l-1}\right)$$

The negative partial derivative represents the proportion and direction for which weights should change to minimise the cost function, hence, improving the accuracy of the neural networks mapping function. Learning rates, α^m, represent the learning rate of the neural network and best practice dictates there should be some reducing factor applied to the rate over each iteration to achieve better SGD convergence properties. Learning rates should be initialised at values large enough to ensure the optimisation algorithm is capable of augmenting weights away from saddle points or sub-optimal minimums and towards convergence in a stable optimal local minimum of the cost function. Furthermore, rates should be set low enough to ensure weight updates are manageable to allow for convergence in an optimal basin on the cost function error surface without the risk of the algorithm diverging significantly or oscillating around a basin without descending towards the minima. Learning rates should have an upper threshold that is double the maximum eigenvalue in the Hessian matrix of the cost function (Bengio, 2012). Recent theoretical advancements in learning annealing using Adaptive SGD optimisation algorithms have altered how the learning rate changes during the optimisation process, with several of these algorithms applied in this chapter and defined in the following sections.

3.1.8.3 Momentum

Momentum SGD (MSGD) is an adaptive technique used to enhance the convergence properties and learning speed of SGD optimisation algorithms through the accumulation of a velocity vector along the direction of the negative gradient (Rumelhart, 1986). Executing vanilla SGD weight updates is dependent solely on the learning rate and the gradient of a training data mini-batch sample. Thus, networks with complex cost functions may have significant stability concerns around the learning process grounded in the risk that SGD may lead to erratic updates and stunted learning if the error surface is difficult to travail. Application of MSGD is particularly relevant for neural networks characterised by these complex error surfaces, particularly those with narrow ravines of varied curvatures along the multi-dimensional surface layer of the cost function. These types of error surfaces predicate high variance oscillations during vanilla SGD learning due to either high curvature faces or very small and consistent gradients. However, they are capable of being traversed by travelling in the relevant downward direction at an accelerated rate through the application of an MSGD velocity vector. MSGD has been shown to improve convergence speed to a local minimum with fewer weight update iterations and softer oscillations than classical SGD (Rumelhart, 1986). MSGD is performed by perturbing the weight, w_i^m, in the direction of the velocity or momentum term, V_i^{m+1}, scaled by the momentum rate, $\gamma \in [0,1)$. The velocity term represents the exponentially decaying moving average of the previous iterations gradient multiplied by the learning rate. Higher momentum values lead to weights being updated more aggressively for dimensions where a series of recent gradients have been large and point in the same direction. Thus, MSGD involves an iterative update to the weight parameter, w_i^{m+1}, as follows:

$$w_i^{m+1} \leftarrow w_i^m + V_i^{m+1}$$

$$V_i^{m+1} \leftarrow \gamma V_i^m - \left(\alpha * \frac{\delta \mathcal{L}}{\delta w_i^m}\right) \qquad \gamma \in [0,1)$$

Tuning the momentum rate hyper-parameter, γ, requires a consideration of its value relative to the learning rate parameter, α. Higher momentum rates impound information from gradients further in the past across many future iterations. Decaying the momentum rate over time is necessary for convergence (Rumelhart, 1986).

3.1.8.4 Nesterov Accelerated Gradient

Nesterov accelerated gradient (NAG) is a first-order optimised SGD learning algorithm closely related to the MSGD technique that evaluates the gradient, $\nabla_w \mathcal{L}(w_i^m + \gamma V_i^m)$, after the scaled velocity, γV_i^m, has been used to update the weight vector, w_i^m (Nesterov, 1983, Sutskever, 2013). Evidently, we see that NAG performs a partial update by adding the velocity vector, γV_i^m, to the weight values, w_i^m, to estimate an initial approximate iterative weight update before evaluating the gradient, whilst the MSGD method computes the gradient from the initial weight position, w_i^m. NAG learning algorithms result in a more stable and tempered weight update than MSGD, particularly for outlier mini-batches, by correcting for excessive velocity values within the gradient computation (Sutskever, 2013). Hence, if the scaled exponentially decayed average velocity term, γV_i^m, perturbs the weights too strongly after an initial update, then the computed gradient, $\nabla_w \mathcal{L}(w_i^m + \gamma V_i^m)$, will augment and decelerate the update back in the opposite direction, limiting the negative effects from inappropriately large velocity updates.

The primary difference between NAG and Momentum methods is that NAG performs a large initial perturbation to the weights before then calculating the gradient, before applying a correction to the weights in its direction, whilst MSGD incurs one large shift to the weights. NAG updates the weights for the $m + 1^{st}$ iteration as follows:

$$w_i^{m+1} \leftarrow w_i^m + V_i^{m+1}$$

$$V_i^{m+1} \leftarrow \gamma V_i^m - \left(\alpha * \frac{\delta \mathcal{L}(w_i^m + \gamma V_i^m)}{\delta w_i^m}\right) \qquad \gamma \in [0,1)$$

3.1.8.5 AdaGrad

Adaptive gradient algorithm (AdaGrad) operates in a similar vein to vanilla SGD, though with iterative adaptations to the base learning parameter, α, on a per-dimension basis for each individual weight parameter, w_i^m, depending on the previous computed gradients for that weight in prior iterations (Duchi, 2011). The learning rate, α, for weight, w_i^{m+1}, is inversely proportional to the root of the summed squares of all gradients, g, of the stochastic cost function w.r.t the relevant weight from all previous iterations, G_{ii}^m, with an added smoothing term, ϵ, to ensure the square root is calculable. This is analogous to computing the L2 norm for all prior iteration gradients. Each element of the diagonal matrix, G_{ii}^m, is a representation of the summed squares across all previous iterations of the gradients with respect to the specific weight element, i, in the matrix. The weight updates using AdaGrad are as follows:

$$w_i^{m+1} \leftarrow w_i^m - \left(\frac{\alpha}{\sqrt{G_{ii}^m + \epsilon}} \cdot \frac{\delta \mathcal{L}}{\delta w_i^m}\right) \quad where \;\; G_{ii}^m = \sum_{j=1}^{m} \left(g_i^j\right)^2 \;\; and \;\; g_i^j = \frac{\delta \mathcal{L}}{\delta w_i^j}$$

Implementing the dynamic AdaGrad learning algorithm provides higher learning rates for sparse parameters with small infrequent updates leading to faster convergence for weight parameters on flatter surfaces. Conversely, AdaGrad results in lower learning rates and hence, smaller updates, for weights with high gradients that are updated regularly. As one would expect, the algorithm trains neural networks with distributed environments quite effectively, with improved convergence properties and stronger robustness relative to SGD, especially in problems that involve datasets with sparse properties (Dean, 2012). Furthermore, the algorithms design predicates a monotonically decreasing and decaying learning rate parameter as the accumulated summed squared gradient term increases with the number of iterations performed, leading to eventual stagnation or stopped learning of knowledge from new training samples.

3.1.8.6 AdaDelta

The AdaDelta algorithm, similar to RMSProp (Tieleman, 2012), provides a dynamic first-order per-dimension adaptive learning rate, developed with the objective of addressing the concerns inherent in AdaGrad of learning parameters decaying aggressively over time, by effectively placing restrictions on the window size for accumulated prior iteration gradients (Zeiler, 2012). Allowing neural network weights to have dynamic learning rates is especially important in deep networks due to the diminishing gradient issue where lower layers update with lower gradients given the series of operations applied to gradients as they backpropagate through the network during learning. AdaDelta scales well for problems requiring

the optimising of nonconvex functions in deeper neural networks through more optimistic searching for locally convex structures, at which point the algorithm slows down discovery, gradients values for the weight accumulate, and the learning rate significantly decreases, allowing the convergence of the network at optimal weight settings. This is performed by reducing the accumulation denominator of the AdaGrad algorithm to a local estimate of the gradient by taking the root of the exponentially decaying weighted moving average of all past iterations accumulated squared gradients, $(V_i^m)^2$, with momentum rate, γ, leading to the AdaDelta weight updates:

$$w_i^{m+1} \leftarrow w_i^m - \left(\frac{\alpha}{\sqrt{\left(V_i^m\right)^2 + \epsilon}} \cdot g_i^m \right)$$

$$V_i^m \leftarrow \gamma\left(V_i^{m-1}\right)^2 + (1-\gamma)(g_i^m)^2 \qquad where\ \ g_i^m = \frac{\delta \mathcal{L}}{\delta w_i^m}$$

3.1.8.7 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) is another first-order adaptive stochastic optimisation algorithm that combines elements of the momentum-based and L2-norm adaptive optimiser algorithms introduced above (Kingma, 2015). Adam extends the AdaDelta algorithm by maintaining the decaying squared gradients learning annealing inversion, V_i^m, whilst augmenting and smoothing the gradient term by incorporating a momentum component, M_i^m, that takes the exponentially decaying mean of past iterations gradients whilst adding a bias correction to account for zero-centred vector initialisation. Both the learning rate annealing denominator and the gradient momentum term vectors are initialised towards zero and hence, it is necessary to counteract the biased moment estimates using bias-corrected estimates, $\hat{V}_i^m$, and, $\hat{M}_i^m$. This involves scaling both annealing and momentum variables by their exponentiate of their respective bias terms, $\beta_1, \beta_2 \in [0,1)$, depending on the number of iterations m, resulting in scale factors of $(\beta_1)^m$, and, $(\beta_1)^m$, respectively. The value of hyper-parameter bias term β_1 should be chosen to provide less relevance to weights further in the past, especially when the vector was initialised, than those calculated in more recent iterations, whilst β_2 needs to be sufficiently large to drive convergence through decreased learning rates when the weights are near the local minimum as the neural network cements specific patterns and the gradients increase in sparsity (Kingma, 2015). Adam weight, momentum and velocity component updates are defined as:

$$w_i^{m+1} \leftarrow w_i^m - \left(\frac{\alpha}{\sqrt{\hat{V}_i^m + \epsilon}} \cdot \hat{M}_i^m \right)$$

$$M_i^m \leftarrow \beta_1 M_i^{m-1} + (1-\beta_1) g_i^m \qquad \hat{M}_i^m \leftarrow \frac{M_i^m}{1-(\beta_1)^m}$$

$$V_i^m \leftarrow \beta_2 V_i^{m-1} + (1-\beta_2)(g_i^m)^2 \qquad \hat{V}_i^m \leftarrow \frac{V_i^m}{1-(\beta_2)^m}$$

3.1.8.8 AdaMax

Learning annealing in the Adam learning algorithm is conducted by inverting the learning rate parameter with the exponentially decayed L2 norm of past iterations' gradients, $\hat{V}_i^m$, a case that can be generalised to the LP norm where $P \to \infty$, using the AdaMax algorithm (Kingma, 2015). Structurally similar to Adam, there is no requirement to correct the momentum term for initialisation bias, and the denominator for the learning parameter is replaced by a max function, U_i^m, initialised to zero, that takes the greater of the recursive value of the previous iteration scaled by β_2, and the absolute value of the gradient with respect to the weights, g_i^m.

$$w_i^{m+1} \leftarrow w_i^m - \left(\frac{\alpha}{1-(\beta_1)^m} \cdot \frac{M_i^m}{U_i^m}\right)$$

$$M_i^m \leftarrow \beta_1 M_i^{m-1} + (1-\beta_1) \qquad \beta_1, \beta_2 \in [0,1)$$

$$g_i^m\, U_i^m = \max\left(\beta_2 U_i^{m-1}, |g_i^m|\right)$$

3.1.8.9 Nesterov Adaptive Moment Estimation

Nesterov Adaptive Moment Estimation (NAdam) adapts components of the Adam algorithm and applies a modification to its momentum term by using the Nesterov accelerate gradient (NAG) momentum technique (Dozat, 2016). The NAG momentum process has been shown to be superior to Momentum SGD (Sutskever, 2013) but requires a modification to its momentum term, M_i^m. NAdam maintains the exponentially decaying average of the first moment past gradients in the moment term, M_i^m, and the second moment of past squared gradients in the learning annealing inversion, V_i^m. The algorithm incorporates a Nesterov component that traditionally first updates weights in the past gradient-driven momentum direction calculated in the prior iteration, M_i^{m-1}, before computing the gradient and correcting weights by perturbing them in the new correct direction (Nesterov, 1983; Sutskever, 2013). However, NAdam incorporates the first step directly in the algorithms iterative momentum term by integrating the current iteration bias-corrected momentum vector, $\hat{M}_i^m$, directly into a new look-ahead momentum update, $\bar{M}_i^m$, which is then used in the weight update formula. Thus, the weight update is utilising Nesterov properties by using the look ahead momentum vector that incorporates the bias-corrected momentum from the current iteration, m, as opposed to the Adam algorithm that only updates weights using $\hat{M}_i^m$ that is the momentum vector from past $(m-1)^{th}$ iterations and doesn't use current iteration computed momentum. The weight updates for NAdam for as follows:

$$w_i^{m+1} \leftarrow w_i^m - \left(\frac{\alpha}{\sqrt{\hat{V}_i^m + \epsilon}} \cdot \bar{M}_i^m\right)$$

$$M_i^m \leftarrow \beta_1 M_i^{m-1} + (1-\beta_1) g_i^m \qquad \hat{M}_i^m \leftarrow \frac{M_i^m}{1-(\beta_1)^m} \qquad \bar{M}_i^m \leftarrow \beta_1 \hat{M}_i^m + (1-\beta_1)\hat{g}_i^m \qquad \hat{g}_i^m \leftarrow \frac{g_i^m}{1-(\beta_1)^m}$$

$$V_i^m \leftarrow \beta_2 V_i^{m-1} + (1-\beta_2)(g_i^m)^2 \qquad \hat{V}_i^m \leftarrow \frac{V_i^m}{1-(\beta_2)^m}$$

3.1.9 Regularisation

Regularisation strategies are deployed with the objective of improving the performance, robustness and generalisation of a neural network by modifying the machine learning algorithm in a way that manages the capacity of the network, reduces the generalisation error of the model, and combats the issue of overfitting. Small network capacities disable a DNNs ability to develop rich representations from the features with high statistical bias whilst large capacities facilitate overfitting of the training dataset that impounds spurious regularities with high statistical variance (Geman, 1992). Regularisation strategies manage this statistical variance-bias trade-off endemic to predictive machine learning models. The process involves adding inhibitory penalties or placing constraints on parameters of the cost function, and the utilisation of ensemble techniques to combine and average parameters across multiple DNNs with the objective of reducing the probability of a single model overfitting the data. Correctly specified and well-fitted machine learning algorithms that are good predictors will be able to generalise the real regularities, inherent to the input-output associations within the training dataset, to future instances.

Overfitting is a frequent problem for DNNs with multiple non-linear hidden layers (Masters, 1993) that results from fitting a poorly generalising model to highly complex associations between inputs and outputs that internalises sampling noise of the training dataset. Models with many free parameters can interpolate the sample training data with great accuracy by memorising associations but may have poor predictive utility when deployed on the test dataset even if drawn from the same distributions of the training dataset (Russell, 2016). The root of this predictive fallacy is the fact that the learnt model has incorporated and captured sampling noise or random irregularities rather than the correct associate input-output relationship.

The Bayesian approach to solving the overfitting problem is to regularise the model by weighting the predictions of a DNNs for all available settings by the posterior probability of that model given the training dataset used. Whilst this best practice approach improves the generalisation of the model, it is computationally expensive and difficult to apply to large datasets such as our FTSE100 LOB dataset. Regularisation of the DNNs objective function is a more realistic solution to improve model performance by reducing overfitting risk. This is generally applied by adding a weight parameter norm penalty to the original cost function using *L1 or L2 Regularisation* to penalise large weights during training (Ng, 2004). Another common method for regularising the data using ensembles or combinations of neural network architectures that is computationally feasible is the *Dropout* technique (Hinton, 2012b; Srivastava, 2014). Early stopping is an implicit regularisation technique based on heuristics of when to cull the training process.

3.1.9.1 Weight Decay L2 Regularisation

L2 Regularisation (Tikhonov, 1943), or weight decay, is a commonly used technique to control the capacity of a DNN model by adding a regularisation term composed of the sum of the squared weights scaled by the regularisation parameter, λ, to the original training criterion cost function, $\mathcal{L}_0$, for given sample size, N:

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}_0 + \frac{\lambda}{2N}\sum_{w} w^2$$

The regularisation technique punishes weights that are too large even if these weights were reducing the original cost function. Effectively, the cost function is augmented by a magnitude of the sum of the DNNs squared weights. The regularisation parameter, λ, provides an equilibrium trade-off between minimising the original cost function whilst ensuring weights do not grow too large. For example, when the parameter is set low, there is minimal effort to protect against overfitting whilst allowing the complexity of the model to increase. However, when the parameter is set high then weight vectors with higher magnitudes are penalised with the result that more input features are considered in the model, rather than several features with very large weights. Regularisation parameters can be set using heuristics or through a grid search that selects the parameter which minimises the cost function error of the validation dataset (Goutte, 1998).

The addition of the regularisation term impacts the backpropagation algorithm during stochastic gradient descent by adding a term to the partial derivative of the cost function with respect to the weights, $\delta\mathcal{L}/\delta w_i$. Expressing the gradient of the cost function w.r.t the weights, $\nabla_{\mathbf{w}}\mathcal{L}$, and biases, $\nabla_{\mathbf{b}}\mathcal{L}$, as a composition of all partial derivatives, the learning algorithm update for iteration m can be derived:

$$\mathbf{w}^{m+1} \leftarrow (1-\alpha\lambda)\mathbf{w}^{m} - (\alpha\nabla_{\mathbf{w}}\mathcal{L})$$

$$\mathbf{b}^{m+1} \leftarrow \mathbf{b}^{m} - (\alpha * \nabla_{\mathbf{b}}\mathcal{L})$$

Hence, the original weight term is first scaled by the learning rate and regularisation parameter in a process generally referred to as weight decay, with the effect of lower weight updates than the original SGD algorithm. Evidently, the bias is not constrained by the regularisation term. The logic behind L2 regularisation is that it forces the DNN to find a slightly simpler solution that fits the underlying pattern in the training data rather than seeking to learn the perfect fit that may lead to memorisation and overfitting to noise in the training dataset. Thus, L2 regularisation equilibrates the trade-off between reducing overfitting whilst still maintaining relative model complexity, resulting in a model that generalises well by effectively suppressing the effect of static noise inherent in samples (Krogh, 1992). Research has found that L2 methods of regularisation may improve DNNs classification performance in some instances (Krizhevsky, 2012).

3.1.9.2 LASSO L1 Regularisation

L1 Least Absolute Shrinkage and Section Operator (LASSO) regularisation techniques apply an L1 norm penalty that follows a similar intuition to L2 regularisation but takes only the absolute value of the weights, rather than its square, when adding the regularisation term to the cost function (Tibshirani, 1996). Therefore, the gradient of the regularisation term no longer scales linearly with the weight parameters but rather reduces simply to a constant with its sign dependent on the sign of the weight. L1 regularisation punishes large weights relatively less than L2 given the extrication of the squared exponent from the penalty term. Whilst L2 regularisation generally precipitates weight vectors with many features having very small weights, L1 Regularisation results in a weight vector with fewer non-zero feature connections given higher importance. Thus, L1 regularisation has a higher probability of pruning synapse weight connections between neurons given their tendency to push weights down towards zero and inject sparsity into the network (Ng, 2004). Sparse representations allow DNNs to disentangle the underlying features that formulate the networks overall representation (Goodfellow, 2009), improving

the ability of the model to generalise. Due to this sparsity property where input features are more spatially localised (Bengio, 2012), there are benefits of using L1 regularisation as a feature selection mechanism to determine features relevant for a certain model (Ng, 2004). The relevant learning algorithm update using L1 regularisation for the weights and biases, as a function of the regularisation parameter λ, can be defined as:

$$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^{m} - (sgn(\mathbf{w}^{m}) * \alpha\lambda) - (\alpha\nabla_{\mathbf{w}}\mathcal{L})$$

$$\mathbf{b}^{m+1} \leftarrow \mathbf{b}^{m} - (\alpha * \nabla_{\mathbf{b}}\mathcal{L})$$

3.1.9.3 Dropout

Dropout is a stochastic regularisation technique for reducing overfitting by learning ensemble representations of the training dataset through the random deactivation of a certain portion of neurons during learning optimisation (Hinton, 2012b; Srivastava, 2014). Masking neurons during training prevents overfitting and improves the robustness of the DNN by constraining the networks ability to co-adapt to certain neurons. Complex co-adaptations between neurons in DNNs are developed when parameter weights are updated during learning to minimise the cost function in a way that fixes errors relating to *other* co-adapted neurons rather than errors of the *individual* parameter. These co-adaptations lead to the model overfitting the training data and poor generalisation to the test data where these co-adaptations between neurons may not exist. Dropout regularisation prevents co-adaptation and overfitting by making the presence of co-adapted neurons unreliable or dependent on a probability, hence, the neurons must learn in situations where their co-adapted neurons are not part of the network (Srivastava, 2014).

Dropout regularisation develops and trains an ensemble of m sub-networks derived by dropping a proportion of random non-output neurons from the base neural network (Srivastava, 2014). Networks with dropout approximately average these m sub-networks and have shown to improve classification performance and minimise errors when optimising DNNs (Srivastava, 2014; Krizhevsky, 2012). Dropout layers at each iteration fit a binary mask vector, $\boldsymbol{\mu} \in \{0,1\}^{n}$, to the output vector of a hidden layer of shape n, with elements of $\boldsymbol{\mu}$ randomly and independently sampled from a Bernoulli distribution with fixed hyper-parameter probability, h_{μ}^{l}. This is defined as the dropout rate of the layer, the probability that a neuron in sub-network k and layer l remains active. Generally, the dropout intensity for layers closer to the input vector is lower, increasing as the network deepens. These probability hyper-parameters should be set to maintain capacity in the DNN with enough neurons in the network capable of forming abstract representations during training, whilst high enough to break neuron co-adaption and prevent overfitting. For neurons in the lth layer of the kth DNN iteration, the dropout technique fits the mask $\boldsymbol{\mu}^{l,k}$ to the activation output vector, $\mathbf{x}^{l,k}$, applying an element-wise multiplication using the Hadamard product to derive the dropout output vector, $\mathbf{x}^{l,k(d)}$, which is fed into the next layer, such that:

$$\mathbf{x}^{l,k(d)} = \left(\mathbf{x}^{l,k}\boldsymbol{\mu}^{l,k}\right) \cdot \frac{1}{1 - h_{\mu}^{l,k}}$$

Evidently, each neuron in layer l of the sub-network will remain active with probability h_{μ}^{l}. Each sub-network with removed neurons is trained individually using the employed learning algorithm. During

validation and testing inference the dropout layer is disabled, with the DNNs trained weights scaled by $1/(1-h_{\mu}^{l,k})$, to maintain expectations of the output during inference.

The dropout process is analogous to learning with bootstrap aggregation, or bagging, as m 'thinned' DNN models of equal size are constructed by sampling with replacement. These 'weaker classifier' models are capable of learning various aspects and errors of the data (Breiman, 2001), before being averaged during inference to form a strong classifier which reduces overfitting and improves predictions if each model m produces independent and uncorrelated errors. The primary difference with bagging is that dropout regularisation develops a series of sub-networks that inherit a subset of parameters from their 'parent' DNN network which are shared across all the sub-network models, whereas bagging involves training multiple independent models (Hinton, 2012b).

A typical architecture of a DNN employing dropout layers is presented in Figure 3.1.5. Neurons with dashed boundary lines represent those masked by the dropout layer and extricated from the manifold during a particular learning iteration. The input and output neurons are never dropped, whilst the dense layer additionally has neurons fixed throughout training. Dashed synaptic weight connections between neuron infers that no information is forward or backpropagated through the connections during learning. Mask vectors are applied only temporarily for a single epoch iteration, with a new mask formulated and fit over the dropout layer during the next training iteration. Note that dropout layers only apply mask vectors during training, not during inference.

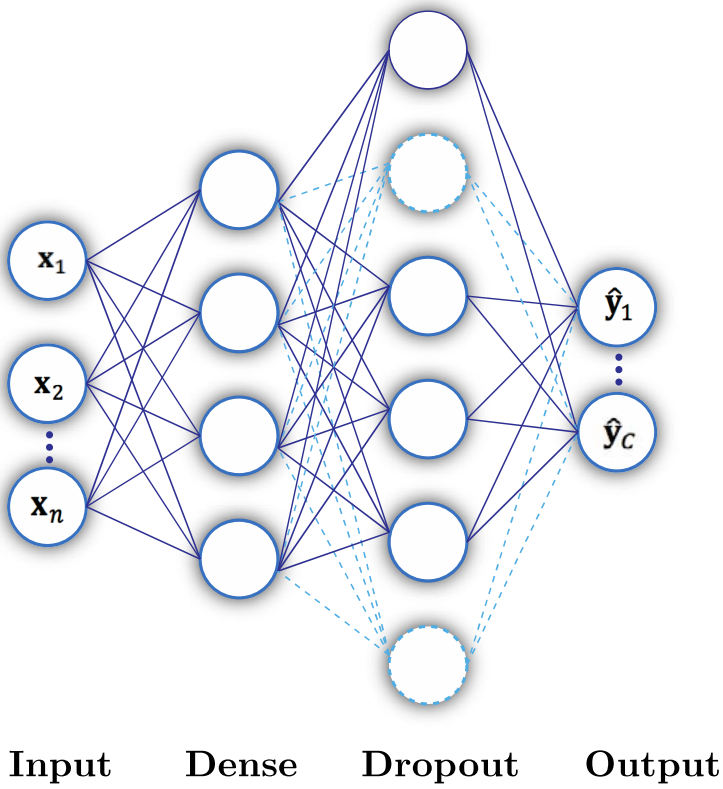

Figure 3.1.5 – Example of a DNN model that employs a dropout layer. The DNN is composed of an input, dense, dropout and output layer. Note that dashed neurons represent those dropped from the data manifold during a single training iteration, whilst dashed weight connections indicate that no information flows through them during learning.

3.1.9.4 Early Stopping

Regularisation of the model is implicit in cases where early stopping strategies are deployed (Wang, 1994; Yao, 2007; Zhang, 2017). Early stopping involves the artificial termination of the training process for a DNN at the point where the model sufficiently explains the underlying associative input-output relationship without overfitting to statistical noise inherent in the training dataset. Statistical noise in this instance refers to the variation in output values that is unpredictable given the input feature vector, regardless of the model, with higher levels of noise increasing the risk of overfitting. In practice, the technique requires a comparison during the learning process between the training accuracy of the model versus the generalisation error when fitting the model to a validation dataset that is representative of the underlying data distribution. An example in Figure 3.1.6 demonstrates how training is stopped when the generalisation error is at a minimum such that any additional training detracts from the ability of the model to generalise. Cross-validation techniques are commonly employed (Sarle, 1995) to monitor the generalisation error in real-time during training, with training stopping when the error begins to increase over several epochs. Parallels between L2 regularisation and early stopping exist in that both perform the same function of limiting the learning of weight parameters to some extent (Collobert, 2004). Whilst L2 regularisation attains this result using an explicit modification to the cost function, early stopping implicitly limits the growth of weight values by pruning them at the point where training is terminated. Early stopping techniques may potentially improve generalisation performance (Zhang, 2017), however, there remains no agreed criteria of when to stop training but rather it is done on an ad hoc basis, generally according to a user-defined heuristic.

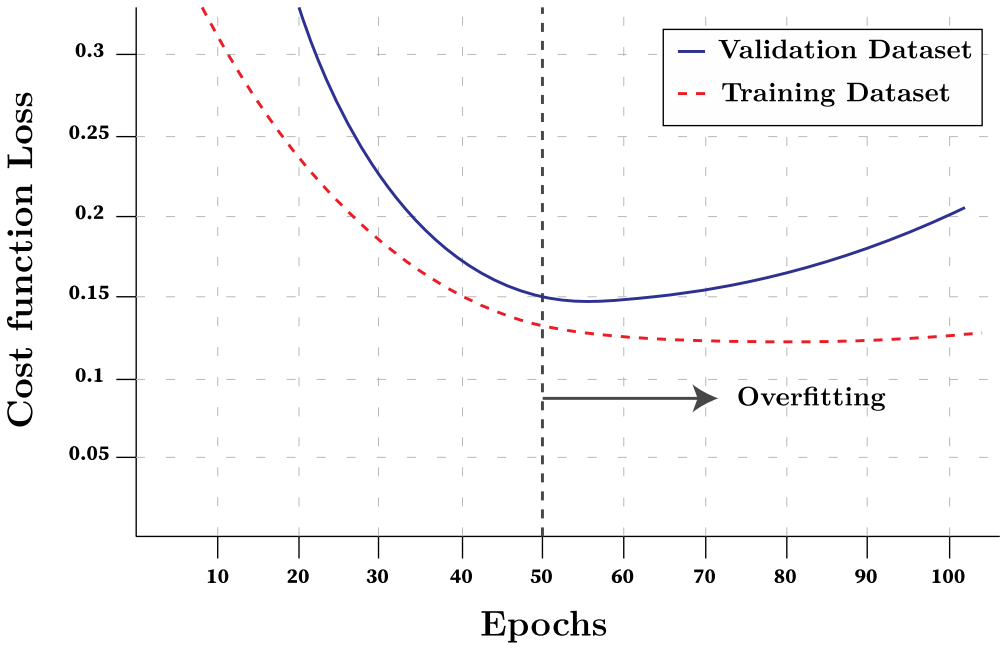

Figure 3.1.6 – Example of a training and validation curve that represent the training dataset cost function loss and validation dataset cost function loss computed by the model after each epoch of training is completed. The common issue with DNN models is that they continue to learn patterns in the data, increasing training accuracy, though at a loss of real-world generalisability. This is illustrated by the validation loss beginning to increase and diverge from the training loss after epoch 50. Training after this point potentially causing the model to overfit to the training dataset, at which point early stopping techniques should be deployed to cull the training process.

3.1.10 Unsupervised Pre-Training with Denoising Auto-Encoders

It has been suggested that pre-training neural networks using greedy layer-wise unsupervised representation learning is an agreeable prior step before applying supervised learning during training of the holistic model (Hinton, 2006; Bengio, 2007). Unsupervised pre-training has been shown to improve generalisation in some instances over simply training a randomly initialised network with generic supervised learning (Bengio, 2012). De-noising stacked auto-encoders (DAE) are an applicable unsupervised learning algorithm that first takes as input the raw feature vector, $\mathbf{x}$, which is mapped to representation $h = f(\mathbf{x})$, based on a non-linear *encoder* function f, before being mapped back into the input feature representation space using a *decoder* function g (Vincent, 2010). These algorithms operate similarly to a feedforward DNN trained using backpropagation, though both the shape of the input and output vector will be identical given that the unlabelled output layer is an attempt at reconstructing the input vector. DAEs extend the initial auto-encoder concept through the integration of a stochastic corruption component by either randomly masking input values or adding Gaussian noise that forces the algorithm to learn by recovering or 'de-noising' the original artificially uncorrupted input. DAEs generally result in feature representations that are more robust to the corrupted data and the discovery of statistical dependencies between input features (Vincent, 2010). However, in the current deep learning domain the use of DAE for pre-training is less common given the plethora of theoretically sound initialisation strategies and the implicit initialisation performed by *deep* networks that employ batch normalisation layers.

The objective of the DAE algorithm is to minimise the reconstruction error, that compares actual input values, $\mathbf{x}$, to those predicted by the auto-encoder reconstruction function, $r(.) = g(f(.))$, generally using mean-squared error (Alain, 2014) or the cross-entropy of both variable distributions (Creswell, 2017). By using a greedy stacked auto-encoder algorithm to train the weights of each layer's reconstruction function, iteratively layer to layer, one has a basis for then applying a supervised gradient-descent based learning algorithm from a more favourable region in the input feature space than if weights were randomly initialised. This region is delineated by the boundary of the supervised learning cost function, with the algorithm biasing the learning dynamics to initialise weights in a 'local basin of attraction' with low generalisation error (Erhan, 2010). Essentially, the unsupervised pre-training procedure restricts the region in parameter space to areas that best capture the input feature vectors distributional structure, $p(\mathbf{x})$. Thus, at each hidden layer the statistically reliable learned features of $\mathbf{x}$, those important in helping to predict the shape of the input vectors distribution $p(\mathbf{x})$, are leveraged to improve the prediction of the DNN output given the input features, $p(\mathbf{y}|\mathbf{x})$.

Unsupervised pre-training as a heuristic method does require an additional learning step, beyond the normal DNN training process, with real computational costs and additional meta-parameters that need to be learnt. However, extracting new information from raw feature signals may make it easier to train a model to learn the conditional probability of the classifier, as opposed to learning this function immediately by merely training a DNN on the training dataset. DAEs additionally have an advantageous and implicit regularising effect on the DNN given that it biases configurations of the input parameter space and reduces variance (Vincent, 2010). This allows the cost function to initialise in an error region with topological features that aid in the gradient descent optimisation process, such as sharp peaks and troughs.

3.1.11 Batch Normalisation

Batch Normalisation (BN) is a modular layer component incorporated into the feedforward layers of the DNN architecture with the objective of improving optimisation performance by stabilising activation value distributions throughout the network (Ioffe, 2015). Stability is achieved by dynamically normalising activation values using adaptive reparameterisation. BN corrects for the phenomenon of internal covariate shift (Ioffe, 2015), which arises from divergent distributions of neuron input values across hidden layers during training. This issue is a natural manifestation of the architectural design of feedforward DNNs as parameter values in lower layers of the networks have a modulating effect on neuron input values in higher hidden layers during forward propagation. For example, forward propagation of input vector, $\mathbf{x}$, involves a series of compounded non-linear operations as feature values flow layer-by-layer through the network, resulting in distributions of input values and weights in layers potentially being vastly different. This effect amplified in deeper networks. The phenomenon of internal covariate shift can result in vanishing gradients during backpropagation as neurons in lower layers saturate relatively easier, stunting network convergence and slowing training (Ioffe, 2015).

The fitting of BN layers in DNNs has also remedied several other learning and convergence issues inherent in the architecture of these models. Previously, smaller learning rates were set to maintain network stability by preventing small shifts in activation inputs amplifying gradients into non-linear saturation regions, resulting in vanishing gradients and the learning algorithm converging into sub-optimal local minima. BN placates this issue by normalising the activation values for each layer which allows the network to deploy higher learning rates and improve convergence speed (Ioffe, 2015). Furthermore, BN allows for less reliance on weight initialisation and reduces the need for dropout or other regularisation techniques. BN has also been shown to have an implicit regularising effect on DNNs given that it performs operations, explained below, on *all* samples in a training batch during each learning epoch (van Laarhoven, 2017). Thus, noise in individual samples is regularised to an extent given that BN uses batch, rather than mini-batch, averages.

BN is applied on a per-layer and per-neuron basis using a method of adaptive reparameterisation by whitening scalar input vectors in a way that corrects for divergent input distributions between layers, improving the training and convergence properties of the network (Ioffe, 2015). Internal covariate shift is countered by fitting a BN layer on top of a networks dense hidden layer that whitens all neurons input activation values for a mini-batch sample by normalising these values to unit Gaussian with zero mean and unit variance. BN is generally applied to the activation value vector, $\mathbf{z}^l$, that is, after the affine transformation but before the non-linearity transformation, ϕ, rather than the input feature vector, $\mathbf{x}^{l-1}$. The rationale proposed by Ioffe (2015) derives from the fact that $\mathbf{x}^{l-1}$ has already undergone a non-linear transformation given its values flow as the output of the previous contiguous layer. So, its distribution is likely to change during training whilst the activation value vector is likely to follow a stable symmetric Gaussian distribution (Hyvarinen, 2000; Ioffe, 2015).

When training an L-layer DNN with mini-batch dataset, $\mathcal{B}$, comprising n training samples, one can normalise the input activation value of neuron j in layer l, z_j^l, by subtracting the mean over the entire mini-batch for that neuron, $\mu_{\mathcal{B},j}$, and standardized by the root of the variance for the mini-batch, $\sqrt{\sigma_{\mathcal{B},j}^2}$, plus a small stability term, ϵ. Note that the bias term of the network neuron is omitted from BN given

its redundancy after reparameterisation where it is annulled when subtracting the mean. This leads to the normalised input activation value for training sample m, $\hat{z}_{j,m}^{l}$, such that:

$$\mu_{\mathcal{B},j} \leftarrow \frac{1}{n}\sum_{m=1}^{n} z_{j,m}^{l}$$

$$\sigma_{\mathcal{B},j}^{2} \leftarrow \frac{1}{n}\sum_{m=1}^{n} \left(z_{j,m}^{l} - \mu_{\mathcal{B},j}\right)^{2}$$

$$\hat{z}_{j,m}^{l} \leftarrow \frac{z_{j,m}^{l} - \mu_{\mathcal{B},j}}{\sqrt{\sigma_{\mathcal{B},j}^{2} + \epsilon}} \qquad where\ \ z_{j}^{l} = \sum_{k}\left[\left(w_{ij}^{l} x_{i}^{l-1}\right)\right]$$

Normalising input activation values before the activation function is applied, whether sigmoidal or rectifier, can constrain these values to a linear regime. Ioffe (2015) recommends applying an identity transformation to scale and shift the normalised activation input values by learnable parameters, γ_{j}^{l}, and, β_{j}^{l}, respectively, for neuron j in layer l, to restore the neural networks expressive power. Hence, correcting for internal covariate shift and normalising neuron inputs requires the next step of executing the differentiable BN transformation, $BN_{\gamma,\beta} z \rightarrow v$, for each input activation value of the mini-batch, $z_{j,m}^{l}$, to derive their linear transformation, $v_{j,m}^{l}$, which is inputted into the activation function ϕ. Learnable BN parameters are identical for each mini-batch $\mathcal{B}$:

$$v_{j,m}^{l} \leftarrow \gamma^{l,j} \hat{z}_{j,m}^{l} + \beta^{l,j} \equiv BN_{\gamma,\beta}\left(z_{j,m}^{l}\right)$$

$$\phi\left(v_{i}^{l,j}\right) \leftarrow \phi\left(z_{i}^{l,j}\right)$$

The additional parameters, $\gamma^{l,j}$, and, $\beta^{l,j}$, are learnt for each neuron of the neural network and updated during each mini-batch training iteration. The increase of only two extra parameters in the network should not prohibit the networks ability to draw representations from the data or its convergence stability properties (Ioffe, 2015). Backpropagation is performed in the same way as in standard feedforward architectures, explained in Section 3.1.8, by computing the cost function gradient w.r.t neuron weights, $\partial\mathcal{L}/\partial w_{ij}^{l}$, with the additional step of updating and optimising the mini-batch sub-network shift, $\beta^{l,j}$, and scale, $\gamma^{l,j}$, until convergence, such that:

$$\frac{\partial\mathcal{L}}{\partial w_{ij}^{l}} = \frac{\partial\mathcal{L}}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{ij}^{l}}$$

$$where\ \ \frac{\partial\mathcal{L}}{\partial\beta^{l,j}} = \sum_{m=1}^{n} \frac{\partial\mathcal{L}}{\partial v_{j,m}^{l}} \qquad and\ \ \frac{\partial\mathcal{L}}{\partial\gamma^{l,j}} = \sum_{m=1}^{n} \frac{\partial\mathcal{L}}{\partial v_{j,m}^{l}} \cdot \hat{z}_{j,m}^{l}$$

In the inference stage, once the DNN has been fully trained and all BN parameters have been learnt, neuron j is normalised using the population mean, μ_{j}, and unbiased variance, σ_{j}^{2}, rather than the batch mean, $\mu_{\mathcal{B},j}$, and variance, $\sigma_{\mathcal{B},j}^{2}$. Each neuron will have its own mean and variance. All other learnt BN parameters remain the same during inference.

3.2 Comparative Machine Learning Models

Deep learning architectures in the form of feedforward Deep Neural Networks (DNNs) represent one technique among a plethora of supervised machine learning algorithms capable of solving the non-linear mapping function, $\mathbf{y} = f(\mathbf{w}, \mathbf{x})$, for the dataset. DNNs are unique given their ability to disentangle abstract feature representations and relationships in data. Comparative non-parametric classification learning models include Generalised Linear Models, Classification and Regression Trees, Deep Random Forests, Support Vector Machines, and Naïve Bayes Models. This section introduces each comparative model separately.

The primary advantage of DNNs over other machine learning models, including Support Vector Machines (SVM) and Random Forests (RF), is their ability to generalise by learning any arbitrary high-dimensional non-linear function. SVM techniques scale poorly to high-dimensional feature spaces and their shallow architecture preclude the formation of abstract features that limits the representable function space to a smaller domain vis-a-vis DNNs (Bengio, 2007). Similarly, despite the success of RFs in various domains including financial time series (Khaidem, 2016) and bioinformatics (Diaz-Uriarte, 2006) classification tasks, these algorithms are only capable of building patterns from raw input features using decision trees and ignore the more abstract higher-level feature representations in data that can be captured by DNNs. Particularly for financial time series-based datasets, RFs may not produce an invariant feature space, finding no productive split in the features, when there is sparsity in the data such that an irrelevant random subset of features is selected. For example, a subset of financial time series or LOB features may work particularly well at explaining trader behaviour, with the relationship captured by a DNN, but ignored by RFs when these subset features are not selected together.

3.2.1 Generalised Linear Models

Regression models employed to solve machine learning classification problems construct trained parametric algorithms, or probability functions, which model the estimated association between training set data and their output classifiers through the predictor function, $\mathbf{y} = f(\mathbf{w}, \mathbf{x})$. Given certain assumptions, including normally distributed response variables, basic linear regression models build a simplistic representation of the general linear relationship, $\mathbf{y}_i = \mathbf{w}_i\mathbf{x}_i + \mathbf{b}_i + \epsilon_i$. Where the real-valued predicted outputs, $\mathbf{y}_i$, are a function of input features, $\mathbf{x}_i$, scaled by coefficient weight variables, $\mathbf{w}_i$, and bias coefficient, $\mathbf{b}_i$. The weights represent relative strength of association between the input features and the output variable and are trained to most accurately map the relationship. Errors of the model, ϵ_i, are assumed to be i.i.d with expected mean zero, fixed variance and normally distributed, $\epsilon_i \sim N(0, \sigma^2)$.

Generalised Linear Models (GLM) are a family of statistical models parameterised by a 'link' function that extends and generalises the normally distributed general linear-response regression model explained above. The advantage of GLMs is that they account for situations where the response variables may have restricted ranges, are not continuous, or may not be normally distributed (McCullagh, 1984). This is commonly the case for financial variables with non-normally distributed properties and restricted boundaries, such as quoted spreads that are bounded by zero. Thus, applying a link function to these arbitrarily distributed financial variables transforms their distributional structure into a format with more agreeable properties for regression.

The structure of GLMs is comprised of three components. Firstly, the *response distribution* specifies the conditional probability distribution of the independent response variable, $\mathbf{y}$, at a certain covariate space point. Response variables are generally drawn from a member of the exponential family of distributions, including multinomial and Gaussian, as such they can be specified completely in terms of their mean and variance. Secondly, the *structural component* requires the specification of a linear predictor, $\mathbf{b}_i + \mathbf{x}_i\mathbf{w}_i$, represented by a linear combination of the explanatory variables and their weight coefficients. Thirdly, an invertible linearising, differentiable and monotonic *link function*, f, is specified in a way that relates the mean of the conditional response distribution at a certain point in covariate space, μ_i, to the systematic components linear predictor, y_i, such that $f(\mu_i) = \mathbf{y}_i$. The link function is designed to transform the response variable, $\mathbf{y}_i$, such that it is linearly related to the explanatory variables, $\mathbf{x}$.

3.2.1.1 Multinomial Logistic Regression

Multinomial logistic regression model the relationship, interaction and association between a multi-dimensional input feature vector linear in parameters, $\mathbf{x}_i$, and a categorical discrete classifier, $\mathbf{y}_i$, which extends the binary dichotomous classifiers used in logistic regression. The assumption of statistically independent classes with low levels of multicollinearity is required to allow for an assessment of the impact from individual input features. Multinomial logistic regression are GLMs that construct a logit predictor function to estimate the probability of observation i pertaining to certain class, $P(y_i = c|\mathbf{x}; \mathbf{w})$, in a set of classes C, based on the 'score' of a specific data sample i for that class, $e^{\mathbf{w}_c\mathbf{x}_i}$. These scores depend on the characteristics of the sample feature vector $\mathbf{x}_i$ and class c's corresponding coefficient weights $\mathbf{w}_c$. Scores are calculated using exponential functions to scale differences between values. Multinomial logit models estimate the probabilities for individual classes using the softmax function that divides the score by the partition function of the distribution which sums all scores:

$$P(\mathbf{y}_i = c|\mathbf{x}; \mathbf{w}) = \frac{e^{\mathbf{w}_c\mathbf{x}_i}}{\sum_{k=1}^{C} e^{\mathbf{w}_k\mathbf{x}_i}}$$

The softmax function compresses the output of the linear function into a range between zero and one to allow for a probabilistic interpretation of the models' default class. For the multinomial case, maximum a posteriori (MAP) estimation can be used to select the classifier with the highest conditional probability P. Training a logistic function algorithm involves optimizing the weights, $\mathbf{w}$, of the independent variable input vector, $\mathbf{x}_i$, in a way that minimises the models' errors, using some cost function, $\mathcal{L}(\mathbf{w})$. The chapter utilises a cross-entropy cost function to compare the distributions of the model's predictions and the actual class from the training dataset to compute the error of the cost function. Iterative stochastic gradient descent (SGD) optimisation is performed to minimise the cross-entropy cost function, $\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{C} -\mathbf{y}_i ln\hat{\mathbf{y}}_i$, where $\mathbf{y}_i \in \{0,1\}$, given that there is no closed-form solution. Furthermore, regularisation techniques, specifically L2 and L1 weight penalties as defined in Section 3.1.9, are applied to the multinomial logistic regression to decrease the risk of the model overfitting to noise in the training data.

Multinomial logistic regression provides an intuitive probabilistic interpretation for training a classifier model, is capable of fast learning using SGD, and avoids severe overfitting through the application of regularisation techniques. The primary drawback of this machine learning algorithm is that it generally performs poorly for datasets with non-linear decision boundaries as it is not able to capture complex non-linear representations in the data that other models, such as DNNs or SVMs, are capable of.

3.2.2 Classification and Regression Trees

Classification and Regression Trees (CART) utilise optimal hierarchical discriminant analysis to model both categorical and continuous classifiers based on conjunctions of features at multiple splitting nodes. These models form the core of random forest (RF) and C5.0 (C50) machine learning algorithms. The objective of CART is to maximise the predictive accuracy of classifier, $\mathbf{y}_i$, by partitioning input feature vector, **x,** into a multi-dimensional space from which observations are split at successive nodes, represented by individual features, in a recursive procedure that culminates in a classification at the terminal node (Breiman, 1984).

Decision trees are highly expressive models capable of learning complex representations of the data from which they can predict classifiers with minimal bias. Furthermore, decision trees are relatively robust to irrelevant or erroneous features which can be discarded at the base of the tree and as such are invariant to transformations or scaling of the input feature vector (Hastie, 2009). However, decision trees are susceptible to overfitting noise in the training dataset which is impounded in the tree, leading to misclassifications of unseen test data drawn from the same distribution. Pruning mechanisms using cross-validation data are implemented to minimise the risk of overfitting and improve accuracy (Strobl, 2009).

Decision tree algorithms, such as C50, develop decision tree models iteratively by selecting features in a sequential recursive process without replacement at each node in a top-down greedy search process based on a specific metric, or impurity, that best splits the data (Rokach, 2005). C50 algorithm uses the information gain (IG) metric, $IG(D,\mathbf{x}_i)$, to split features, a technique found to yield similar results to the Gini impurity metric commonly used in CART algorithms (Raileanu, 2004). IG measures the ability of an individual feature to discriminate between classes by calculating the difference between the entropy of training data sample subset D, $H(D)$, and the weighted conditional entropy, $H(D|\mathbf{x}_i)$. Training data entropy is equal to the summed entropy for all classes $c \in C$, $H(D) = \sum_{c\in C} p(D_c)\ln\big(p(D_c)\big)$. Computing the conditional entropy requires the definition of all possible discrete values v in the training dataset sample D for the feature $\mathbf{x}_i$, $v \in Val(\mathbf{x}_i)$, where D_v encompasses the set of samples where the feature $\mathbf{x}_i$ equals the value v, such that $D_v = \{\mathbf{x}_i \in D|Val(\mathbf{x}_i) = v\}$. After segmenting the feature vector into its composite values, conditional entropy of the dataset D is measured as, $H(D|\mathbf{x}_i) = \sum_{v\in Val(\mathbf{x}_i)} \frac{|D_v|}{|D|} H(D_v)$, with $\frac{|D_v|}{|D|}$ representing the fraction of the dataset where the feature equals v and $H(D_v)$ is the entropy of the subset of the training dataset where the feature is also equal to v. Entropy measures the uncertainty in the distribution of the random variable being measured, such that if all classes equal the same value, a uniform distribution, entropy equals one. IG generally performs poorly for datasets where features are continuous, such as the FTSE 100 dataset employed in this chapter, causing the model to build too many branches leading to irregularities being impounded into the model and overfitting occurring. Information gain ratio, $IGR(D,\mathbf{x}_i)$, corrects for these concerns by penalising features not capable of distinctly classifying and splitting the data in a broad and uniform manner. This involves scaling information gain, $IG(D,\mathbf{x}_i)$, by the inverse of the intrinsic value measure, $IV(D,\mathbf{x}_i)$, leading to:

$$IGR(D,\mathbf{x}_i) = \frac{IG(D,\mathbf{x}_i)}{IV(D,\mathbf{x}_i)} = \frac{\sum_{c\in C} p(D_c)\ln\big(p(D_c)\big) - \sum_{v\in Val(\mathbf{x}_i)} \frac{|D_v|}{|D|} H(D_v)}{\sum_{v\in Val(\mathbf{x}_i)} \frac{|D_v|}{|D|} \ln\left(\frac{|D_v|}{|D|}\right)}$$

Thus, at each sequential node, features with the highest $IGR(D,\mathbf{x}_i)$, are used to split the training dataset.

3.2.3 Random Forests

Random Forests (RF) employ the *ensemble learning* techniques of *bootstrap aggregation* and the *random subspace method* to construct an improved lower variance classifier by amalgamating a multitude of complex low bias decision trees (Breiman, 2001; Ho, 1995). Forests are built using a randomised tree-building algorithm that grows in parallel M independent, diverse and uncorrelated decision tree models composed of a randomised bootstrap subset of N samples selected with replacement from the training dataset, with each tree node splitting the sample using a random subset of K features. Predictors are developed by aggregating and averaging the ensemble of M orthogonal decision trees, then computing the forests majority vote of the classifier given the input data. Decision trees tend to overfit by learning irregular patterns in the training dataset, resulting in a predictor with high variance, poor generalisation and high sensitivity to minor fluctuations in feature values (Hastie, 2009). RF's reduce the high variance manifest in individual decision trees and their risk of overfitting the data by building an ensemble of multiple trees trained with different data and feature subsets.

Recent theoretical advancements in RF architectures (Kontschieder, 2015; Zhou, 2017; Miller, 2017) have enabled learning techniques from DNNs to be integrated into RF's. However, there exists minimal theoretical evidence that these models are capable of developing hierarchical feature abstractions in the same vein as DNN or other deep learning models.

Determining the structure of the RF and number of decision tree models requires a trade-off between improving performance accuracy at the cost of computational resources. Machine learning problems involving many classifiers will require enough trees capable of modelling the inherent representations in the data and causal relationship between input features and the classifier in a way that improves classification performance. Oshiro (2012) suggests a range of 64 to 128 decision trees as a way of managing the trade-off between error rates and computational power, with less trees grown if feature dimensionality is small. We determine the optimal number of trees by using cross-validation and a heuristic rule that searches for an optimal M by comparing the RF's out-of-bag error rates, an estimate of the model's generalisation ability.

3.2.3.1 Ensemble Methods

Ensemble methods are a metaheuristic form of supervised learning that integrates multiple learner models to formulate a more accurate predictive algorithm than would be possible if only one of the iterative models were considered. Ensemble methods do not presuppose a globally optimal solution for classification problems, but rather implement a combinatorial optimisation function that averages over a set of learner models. A necessary precondition for ensemble methods to outperform its component models requires that they are sufficiently diverse with uncorrelated errors and accurate classifiers (Kuncheva, 2003).

Several constraints of single classifier models, such as CART decision trees, motivate the deployment of RF ensemble methods. First, ensembles control the trade-off between a single complex model that potentially overfits the data and has high variance in the classifier, and a single simple model with high bias that may underfit the data. Second, the 'true' classification function may not exist within the hypothesis space of this single model, thus by expanding the number of models to reach outside this

hypothesis space, optimal model parameters may be solved. However, injecting diversity into the learning process for its own sake to develop heterogeneous RF's with potentially irrelevant attributes, does not perform as good at classification as RF's with stronger learning algorithms and relevant features (Gashler, 2008).

3.2.3.2 Random Subspace Method

Random subspace ensemble methods, or 'feature bagging', were developed to overcome overfitting problems inherent in decision trees by training an ensemble of learner models that sample a random subset, or random subspace, of features from the training dataset with replacement (Ho, 1995; Ho, 1998). Applying this technique reduces the correlation between predictor classifiers in the ensemble if some features are very strong predictors such that they dominate the learning process of single decision trees. Hastie (2009) provides the heuristic that for datasets trained with m separate input features, one should select a random subspace of $\sqrt{m}$ features, with a lower bound of five features, when splitting each individual tree node.

3.2.3.3 Bootstrap Aggregation

RF's utilise bootstrap aggregation, or bagging, to decompose training dataset, $D(\mathbf{x}, \mathbf{y})$, into M bootstrap replicate training subsets composed of N data points of the same cardinality which are drawn uniformly with replacement and randomly from the training dataset (Breiman, 2001). Bagging's intuitive parallel ensemble meta-algorithm allows an RF to grow M independent and diverse base learner trees the training data subsets, with base learners selecting a subset of features using either the random subspace method (Ho, 1995), or randomised node optimisation (Dietterich, 2000). The decision tree learner algorithm is applied to each sample, split along K features, in a recursive process until the classified, $f_m(\mathbf{x}, \mathbf{y})$, is defined at the terminal node. Individual classifiers are averaged over the forest to compute the RF classifier, f^{Opt}.

$$f^{Opt} = \frac{1}{M} \sum_{m \in M} f_m(\mathbf{x}_K, \mathbf{y})$$

Bagging ensemble methods that amalgamate votes of weaker classifier trees in the forest have been found to improve on the stability of a simple classification model, such as a single CART, reducing the variance in the predictor without a corresponding increase in bias (Breiman, 1996). Once an RF has been built, model accuracy and generalisation are computed using the out-of-bag (OOB) error. OOB measures the mean classification error of all trees when applied to a validation dataset composed of all training samples that were *not* a component of the bootstrap sample used to build the tree. OOB error is a function of the level of correlation between decision trees in the forest and the bias of each individual tree (Breiman, 2001).

3.2.4 Support Vector Machines

Support Vector Machine (SVM) algorithms are binary non-probabilistic classifier models, which after learning an optimal class separation hyperplane, is capable of automatically executing the classification process based on the feature space $\boldsymbol{R^n}$ rather than some stochastic element (Vapnik, 1963; Boser, 1992). Whilst these models derive naturally as a binary class separator, it becomes relatively straightforward to extend this technique to a multi-class problem by training multiple binary one vs one (OVO) with complexity $\mathcal{O}(C^2)$ or one vs all (OVA) SVMs with complexity $\mathcal{O}(C)$, where C is the number of classes, then using a voting system to classify sample data points.

For training dataset with data point tuples $(\mathbf{x}, \mathbf{y})$, defined by a real vector of features, $\mathbf{x}$, and classifiers, $\mathbf{y}$, there exists a hyperplane, $\mathbf{w}$, offset from the origin by bias b, which achieves optimal separation by maximizing its distance from the nearest training data points for each classifier, known as the support vectors. Vapnik (1963) first proposed the SVM model for linearly separable classifiers in which the optimal maximum margin hyperplane (MMH) is a normal vector function, $||\mathbf{w}||$, that provides the maximum functional margin between the MMH and training set classifier hyperplanes. Thus, support vectors represent points most difficult to classify as they lie close to the MMH, with the number of support vectors positively correlated with the ability of the SVM to generalise to new data (Boser, 1992).

3.2.4.1 Soft-Margin SVM

Soft-margin SVMs relax the assumption that classifiers must be linearly separable by allowing minor deviations of data points from the MMH (Cortes, 1995). These SVMs apply separation to high-dimensional systems with non-linearly separable data by utilising a hinge loss cost function and solving for the optimal hyperplane $\mathbf{w}$ that minimises the separation boundary $||\mathbf{w}||$ given the constraints:

$$\min_{w} ||\mathbf{w}|| \quad and \quad \mathbf{y}_n(\mathbf{w}.\mathbf{x}_n + b) \geq 1$$

The optimisation problem requires consideration of both constraints which can be solved using the Dual Lagrangian multiplier method that intuitively seeks to find the intersection where both constraints are satisfied. Solving for the SVMs optimal soft-margin hyperplane requires the maximisation of the Dual Lagrangian, $L(\mathbf{x}, \mathbf{a})$, with respect to the introduced dual 'slack' variable, $\mathbf{a}$, subject to certain conditions, with the optimisation problem hence defined as:

$$\max_{C} L(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{n} \mathbf{a}_i - \frac{1}{2}\sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_j \mathbf{y}_i \mathbf{y}_j (\mathbf{x}_i . \mathbf{x}_j) \qquad s.t. \sum_{i=1}^{n} \mathbf{a}_i \mathbf{y}_i = 0 \quad \& \quad 0 \leq \mathbf{a}_i \leq C$$

The slack variables can be solved for by using this method. This first requires defining the concurrent cost of constraint hyper-parameter C which controls for how 'soft' the margin is for misclassified data. This modulates the trade-off between data points being on the correct side of the hyperplane whilst maintaining a maximum functional margin that effectively separates classes. In this way, misclassified outliers impose a cost which is controlled by a slack regularisation term to ensure the model does not overfit to outliers in the training data. Higher penalty values of $\mathbf{a}$ impose higher costs on incorrect classification and result in an SVM that has a lower probability of misclassifying data, that is, harder margins, more rigidity in the boundaries and rougher decision surfaces. Standard hyper-parameter grid

searches can be used to find the optimal penalty **a**. Once value **a** has been calculated, the optimal soft-margin linear hyperplane that splits the classifiers is determined to be:

$$\mathbf{w} = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{y}_i \mathbf{x}_i$$

3.2.4.2 Kernel Functions & Non-Linear SVM

Boser (1992) extended the soft-margin SVM to apply to arbitrary non-linear classifiers using kernel functions. When the feature space includes non-linearly separable overlapping support vectors, it results in a potentially non-linear optimal separation hyperplane. The kernel 'trick' is a feature mapping function, $\vartheta: \boldsymbol{R^n} \rightarrow \boldsymbol{S}$, which transforms data points $\mathbf{x}_n \rightarrow \vartheta(\mathbf{x}_n)$ into a higher dimensional feature space $\boldsymbol{S}$ that allows classifiers to be spatially separated. Instead of transforming the feature vectors it is possible to define a kernel function, $\boldsymbol{K}$**,** a symmetric positive semi-definite function which can be expressed as a dot product between support vectors, $\vartheta(\mathbf{x}_i)$ and $\vartheta(\mathbf{x}_j)$ in multi-dimensional feature space $\boldsymbol{S}$ such that:

$$\boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_j) = \langle \vartheta(\mathbf{x}_i) \cdot \vartheta(\mathbf{x}_j) \rangle$$

Intuitively, the inner product serves as a measure of similarity of distance, equalling one when parallel with complete similarity and zero when perpendicular implying no similarity (Vert, 2004). Using the kernel function, the support vectors do not need to be transformed into in *multi-dimensional space* $\boldsymbol{S}$ to calculate the dot product, but rather this can be replaced by computing the dot product between the two support vectors in *current feature space* $\mathbf{x}_i, \mathbf{x}_j \in \boldsymbol{R^n}$. Thus, applying the kernel function $\boldsymbol{K}$ renders it unnecessary to compute all extended basis vectors in the transformed feature space $\boldsymbol{S}$, allowing for a less computationally expensive optimisation problem for non-linear SVMs that reduces to:

$$\max_{\mathbf{c}} L(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{n} \mathbf{a}_i - \frac{1}{2} \sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_j \mathbf{y}_i \mathbf{y}_j \boldsymbol{K}(\mathbf{x}_i . \mathbf{x}_j) \qquad s.t. \sum_{i=1}^{n} \mathbf{a}_i \mathbf{y}_i = 0 \quad \& \quad 0 \leq \mathbf{a}_i \leq C$$

Radial basis function (RBF) kernels, $\boldsymbol{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{\gamma \cdot |\mathbf{x}_i - \mathbf{x}_j|^2}$, are applied to SVMs employed in this chapter given their common use in literature (Chang, 2010). RBF kernels provide a distance metric bounded between zero and one, with higher values for feature vectors that are similar and lower values for feature vectors that are dissimilar. The introduction of the gamma smoothness parameter γ provides a proxy for the impact that single training data points have on the SVM classification, acting as an inverse radius of influence for support vector samples. Thus, gamma variables are inversely related to the standard deviation of the RBF Gaussian function around each data point that influences the level of similarity between the point in which it is being compared. For example, larger gamma values indicate that the influence of a support vector $\mathbf{x}_i$ on classifying a data point $\mathbf{x}_j$ is relatively small even if input features are similar. Gamma values set too high will therefore result in the SVM having high variance in the predictor and low bias, given that the influence radius of support vectors is very small, potentially creating pointed hyperplanes that overfit the training data. Alternatively, smaller gammas allow support vectors to have a higher influence on the class as Gaussians have higher standard deviations and wider bell-curves. This could lead to under-fitted SVMs incapable of modelling complex non-linear hyperplanes that separate the data, resulting in a low variance and high bias model. Hyper-parameters are generally tuned using k-fold cross-validation to improve generalisation and attain unbiased estimates of classifiers.

3.2.5 Naïve Bayes Models

Naïve Bayes classification techniques are a probabilistic machine learning method that utilises Bayes theorem and generative conditional probability models to define classifiers, achieving competitive results in some classification problems (Rish, 2001). Naïve Bayes models are based on the naïve assumption that each individual feature in the dataset is independent of one another with correlations and interaction effects between two or multiple features ignored. For a feature vector $\mathbf{x}$ of length n and classifier $c \in C$, classification is dependent on the conditional probability, $p(\mathbf{y}_c|\mathbf{x})$, of class c occurring given prior information regarding the feature vector, $\mathbf{x}$, such that:

$$p(\mathbf{y}_c|\mathbf{x}) = \gamma . p(\mathbf{y}_c) \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{y}_c)$$

These posterior probability values are a function of the prior class probability, $p(\mathbf{y}_c)$, the likelihood of the data feature given the class, $p(\mathbf{x}_i|\mathbf{y}_c)$, which is calculated using an event model based on feature distributions, and the scalar factor γ. Bayesian inference estimates the posterior probability distribution for class c given the sample feature values. Bayesian decision theory and maximum a posteriori estimation approaches to solving machine learning problems require training models to compute the posterior distribution of class probabilities based on the imputed dataset, then selecting the classifier with the highest probability across the hypothesis space. Multinomial Naïve Bayes models are based on the frequencies of discrete feature values observed, $\mathbf{x}$, based on a specific class, $\mathbf{y}_c$, which calculate the posterior probability as:

$$\log\big(p(\mathbf{y}_c|\mathbf{x})\big) = \log(p(\mathbf{y}_c) + \sum_{i=1}^{n} \mathbf{x}_i . \log(p_{c,i})$$

Additive smoothing regularisation is applied to probabilities of feature-class pairs with probability zero by modifying their values with Laplace smoothing to improve stability in learning and convergence properties. Classification using multinomial Naïve Bayes models and maximum-likelihood training can be performed using iterative approximation or by solving a closed-form expression. Once a model is trained, predictions of the classifier can be made by selecting the class with the maximum a posteriori hypothesis:

$$MAP(\mathbf{y}_c) = \max(p(\mathbf{x}|\mathbf{y}_c) * p(\mathbf{y}_c)$$

3.3 Data

The deep learning optimisation algorithms and machine learning classification models employed in this chapter utilise raw UK FTSE100 equity order-book data and other external financial data sources to learn representations in the relationship between trader actions and the features of the limit order book (LOB). We utilise two primary datasets to view, model and analyse trader behaviour – a unique consolidated limit order book of FTSE 100 UK equities (FTSE100) developed from trading exchange reporting data provided by the Financial Conduct Authority (FCA), and the Thomson Reuters Tick History (TRTH) database. Securities Industry Research Centre of Asia-Pacific (SIRCA) has provided access to TRTH which contains microsecond-timestamped intraday trade and order data for a range of equity securities across financial markets.

The unique FTSE100 CLOB data provided by the FCA captures all messages between market participants and four primary UK exchange lit and three dark-pools – London Stock Exchange (XLON), BATS Europe Lit BXE (BATE), BATS Europe CXE (CHIX), Turquoise Integrated Lit Order Book (TRQX), Bats Europe BXE Dark order Book (BATD), Bats Europe CXE Dark Order Book (CHID) & Turquoise Plato Dark Order Book (TRQM) – introduced in Section 2.1. The dataset covers the period from January 2015 to December 2017 for all FTSE 100 securities identified by their unique ISIN. Raw unfiltered message data and transaction reports for each of the order books undergo a cleaning and filtering process, before being amalgamated into a single FTSE100 CLOB. Orders placed on each exchange occur in a sequential order from the beginning of trading until all orders are cleared off the LOB at the end of the trading day. Participants are identified across markets by their Broker ID which is a function of their unique Bank Identification Code (BIC). The dataset includes intraday orders submitted, cancelled, amended, all transactions, and other residual messages at all levels of the LOB and is timestamped to the millisecond. Additional meta-data is provided in the form of the order type, for example limit or iceberg order, the initiator and aggressor of the trade, being either the bid or ask broker, and the period that the trade is submitted, during continuous trading or auctions.

A primary facet of the FCA CLOB FTSE100 dataset is the consolidated nature of the LOB which enables us to perform a panoptic inter-market analysis across the primary UK equity trading venues that represent 97% of trading activity over the period 2015-2017 for FTSE100 sample securities.[20] A key tenet of this thesis is that analysing the complex interdependencies and phenomena of algorithmic trading requires a holistic consideration of their actions and state space across all relevant trading venues rather than any unitary market such as the LSE. At any point in time, before a trader executes an action, also referred to as an event, one can develop a representation of the state of the CLOB that a market participant would be faced with. From this reconstructed state, a Deep Neural Network (DNN) can be fitted to model how market participants respond to microstructural phenomena, particularly changes in market dynamics for a specific security, given their current order distribution. This chapter truncates the spatial dimensions of the LOB by defining an event as any action or interaction between a market participant and the LOB that modifies their order distribution within five ticks of the BBO.

[20] Estimates were calculated for the period 1/1/15 to 31/12/17 for the FTSE 100 Index using data from fragmentation.fidessa.com.

The FTSE100 CLOB dataset is constrained by several limitations. First, meta-data is reported by trading venues at the millisecond level that does not allow a deeper analysis of sub-millisecond strategies commonly deployed by algorithmic traders. Second, the chapter does not consider internally matched trades or interactions between traders and dark pools, potentially limiting the ability of the model to capture all strategies being executed. Finally, data is presented at the Broker level which provides a challenge to differentiate between different investors or underlying strategies that may be conducted at the desk-level of a firm. Each firm may have several Broker IDs through which they trade, with each considered to be separate during this analysis.

Due to the computational resources required to train deep learning algorithms, we limit the period of analysis to five days over the first week of June 2015 (1/6/2015-5/6/2015), and five randomly sampled FTSE100 securities that trade across all four trading venues. The period of analysis, stocks and sample size have been chosen to maintain a level of robustness and accuracy in the analysis, whilst working with the constraints of available computational resources. The dataset totals several billion messages exchanged between traders and venues and is cumulatively over one terabyte in size. Over the sample dataset, none of the securities undergoes a trading halt or suspension that impairs the continuous trading of that stock.

Descriptive statistics for the FTSE100 CLOB dataset utilised are reported in Table 3.3.1 as an average for each of the five securities analysed – VOD, BAT, BP, GSK, HSBA – over the five trading dates. Market capitalisation on average across the sample trading firms was £80.96bn, with HSBA having the highest market valuation of £121.27bn for its shares. UK FTSE100 securities are among the most liquid and actively traded financial products in the world with the composite securities seeing an average exchange of £171 million over a single trading day. The immense computational resources required in developing DNN models to explain the dataset is evident in the number of relevant messages pertaining to orders within five ticks of the BBO for all stocks being over 1.1 million messages on average per stock over a trading day. The domination of modern markets by algorithmic trading strategies has required a reassessment of how academic enquiry is conducted in the realm of financial market microstructure. Analysis of only trade data is justified given that transactions represent the real exchange of financial assets, whilst orders merely represent a traders' willingness to transact at a specific price for a given quantity. However, from the table one can see that trades represent only 2.2% of applicable messages on UK equity trading venues. The implications are that trader interactions with the LOB at the limit-order level are a prerequisite when modelling algorithmic trading strategies based on traders' behaviour.

The high-frequency nature of UK equity LOB dynamics is evident in Table 3.3.1 by the fact that traders send an average of 34.7 messages per second to the four primary trading venues, of which the primary market LSE accounts for 39.9% of those messages, on average. It should be noted that the range of primary market activity for different stocks oscillates between 36.8% and 43.9%, on average, indicating that activity is distributed differently between the venues depending on the specific stock traded. Limit orders clearly dominate over more complex order type on lit venues, representing 88.7% of all messages on average. Interestingly, the average order survival time of 35.7 seconds appears to be quite high and an execution ratio of 2.38% quite low for hyper-liquid stocks with a significant presence of high-frequency trading activity. However, these results are based on orders submitted within five ticks of the BBO, hence, messages further from the BBO may remain on the LOB longer without any commensurate impact on LOB dynamics. Finally, average market quality metrics of effective spreads, quoted spreads,

mid-quote volatility and BBO market depth, are reported for the composite dataset stocks. The average quoted spread of 1.82 ticks over the sample stocks is in line with expectations.

Metric	Units	VOD	BAT	BP	GSK	HSBA	Average
Market Cap	*£ billions*	66.52	65.79	81.53	69.70	121.27	**80.96**
Messages	*Count*	1249300	1166180	1548897	660806	1305537	**1186144**
Number of Trades	*Count*	34306	19360	29455	22207	27823	**26630**
Volume Traded	*Shares millions*	98.7	3.4	30.8	11.1	30.7	**34.9**
Value Traded	*£ millions*	246.7	120.8	137.5	159.6	190.9	**171.1**
Order Size	*£ thousands*	13.6	10.9	7.7	11.4	9.9	**10.7**
Messages per Second	*Count*	36.53	34.10	45.29	19.32	38.17	**34.68**
Execution Ratio	*Ratio*	2.67%	1.68%	1.94%	3.48%	2.15%	**2.38%**
Cancellation ratio	*Ratio*	47.32%	47.83%	47.90%	46.06%	47.67%	**47.35%**
LSE Market Share	*% Messages*	37.42%	43.85%	39.52%	41.79%	36.84%	**39.88%**
Limit Order Type	*% Orders Submitted*	86.64%	90.93%	92.74%	86.29%	87.70%	**88.86%**
Traders	*Count*	93	85	92	92	89	**90**
Order Survival Time	*Seconds*	35.4	19.4	23.1	64.4	36.3	**35.7**
Effective Spread	*Bps*	0.025	0.021	0.019	0.033	0.019	**0.023**
Quoted Spread	*Ticks*	1.62	2.03	2.45	1.28	1.73	**1.82**
Mid-Quote Volatility	*SD*	0.0071	0.0065	0.0072	0.0068	0.0061	**0.0067**
Market Depth	*£ thousands*	104.6	48.3	28.7	173.7	79.7	**87.2**

Table 3.3.1 – Descriptive statistics for the FTSE100 CLOB dataset comprising five FTSE 100 composite securities traded over the period 1/6/2015-5/6/2015 on the four primary UK equity order books – LSE, Bats, Chi-X and Turquoise.

The average distribution of order depth across all stocks and trading days is also studied and presented in Figure 3.3.1 for a total of 12 price points at and within five ticks of the BBO on both the bid and ask side of the LOB. Note that the best ask quantity (BAQ) and best bid quantity (BBQ) at five ticks from the BBO includes orders placed further outside of these boundaries. Evidently, the shape of the average order distribution is symmetrical about the BBO. Lower depth is registered at the BBO compared to the other price points, with the BAQ averaging 4.78% of liquidity and BBQ averaging 4.4% when analysed over the 12 price points closest to the BBO. Market depth increases as price points move towards three ticks from the BBO on both sides, before reducing once more.

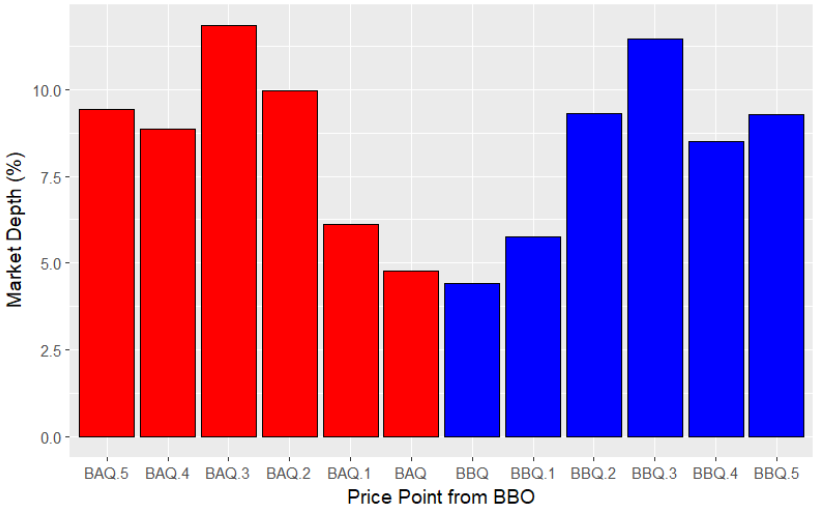

Figure 3.3.1 – Order distribution on consolidated LOB for sample FTSE 100 securities. The horizontal axis refers to the price point from the relevant BBO. The vertical axis is measured as the time-weighted average market depth at price points at and within five ticks of the BBO.

3.3.1 Limit Order Book Features & Actions

Algorithmic trading (AT) strategies deployed on UK equity markets are identified, grouped and analysed using a deep neural network (DNN) that builds a prediction framework of AT firms *actions* and behaviour, **y**, based on their relationship to a set of market microstructural *features*, **x**. Features and actions are derived from the FTSE100 CLOB dataset described in the previous section.

Feature engineering is performed to identify and generate a rich set of limit order book (LOB) features that one would expect to be integrated into the set of rules, calculations and operations underlying the algorithm of a firm conducting an AT strategy on UK equity markets. The derived features are numeric, measurable and independently observed characteristics of the current state and dynamics of the consolidated UK LOB in its totality, the algorithmic trader performing an action, and various phenomenon in financial markets. The process of feature generation and selection is explained in Section 2.4 based on a comprehensive study of relevant feature inputs in AT algorithms conducted in Section 2.3. A holistic set of 323 features are generated granularly by analysing each individual AT strategy and extracting relevant phenomenon in the LOB that may be critical inputs into each specific AT strategies algorithms. These features form the input for our DNN to predict the actions of traders executing a specific trading strategy. LOB features are generated from the consolidated FTSE100 dataset. Primary strategies investigated include Automated Market Making, Execution Strategies, Microstructural Signal-based Strategies, Momentum Strategies and Technical Strategies. The full set of 323 LOB features **x** utilised by DNN models in this chapter are holistically defined in Appendix A, with a summary of the features relevant for specific AT strategies provided below:

- ***Automated Market Making*** – traders actual, volatility normalised, and aggressiveness level normalised quote offset from the BBO, traders net inventory ratio and relative inventory position, effective spreads, probability of informed trading measures and investor risk aversion factors.
- ***Execution Strategies*** – percentage of volume (POV) imbalance, Volume-Weighted Average Price (VWAP), Implementation Shortfall (IS), liquidity and spread charges, timing consequence, price impact metrics, and impulse response functions.
- ***Microstructural Strategies*** – various order flow, aggressiveness level, market depth, limit order imbalance, quoted spreads and volatility signals.
- ***Momentum Strategies*** – various momentum and acceleration LOB signals.
- ***Technical Strategies*** – Bollinger bands, Moving Average Convergence-Divergence Index, Relative Strength Index, Stochastic Oscillators, and Chaikin Volatility.

The action executed by an algorithmic trade, **y**, represents the response of that trader to the state of the LOB, **x**, composed of a set of complex and abstract LOB features that explains the current state of the environment, as explained above. Actions represent one of 26 discrete responses available to algorithmic traders at event time t defined through the action space $\mathcal{A}$. These actions are viewed externally as events that impact the state of the LOB, transforming input variable values for other participants' algorithms, that in turn induce other participants to execute actions. LOB interactions over the course of the trading day provide a holistic depiction of various trading strategies that together make a market and perform price discovery for a specific security. This chapter is interested in how actions executed by traders can be explained given current state of the financial environment as viewed by that trader, with these complex dependencies captured within the DNN model.

The DNN encapsulates 26 primary actions AT firms execute on the LOB. These actions can be segmented by market, $MO_t^{S,V}$, limit submission, $LO_t^{S,\tau}$, iceberg submission, $ICE_t^{S,\tau}$, and immediate-or-cancel submission, $IOC_t^{S,\tau}$, orders that are sent by the AT firm to one of the four trading venue LOBs captured in the dataset. Orders are categorised by the side of the LOB that it is placed, S, defined as either a bid order, B, or ask order, A. Furthermore, for order submissions, orders are segmented by the distance of the order from the LOB, τ, defined for order placed at or within the best bid-ask spread (BBO), 0, one tick outside the BBO, 1, two ticks outside the BBO, 2, or between three and five ticks outside the BBO, 3. For the purpose of model simplicity, both hidden and iceberg orders are amalgamated into a single category, $ICE_t^{S,\tau}$, given that the placement of pure hidden orders is relatively rare in the dataset and the fact that both orders have very similar properties as a portion of their total orders is not visible on the LOB. Additionally, fill-or-kill (FOK) orders and immediate-or-cancel (IOC) orders are also clustered into the same action category, $IOC_t^{S,\tau}$, as both types of orders have similar attributes. Furthermore, pegged and stop orders, which represent less than 0.01% of all orders, are not included in the analysis. For classification purposes in the DNN model, actions are treated as categorical classes and defined in Table 3.3.2.

$\mathcal{A}$	Action type	Action side	Action definition
MO_t^B	Market order	Bid	Price >= BAP & OT=MO
MO_t^A	Market order	Ask	Price <= BBP & OT=MO
$LO_t^{B,0}$	Limit order	Bid	Price >= BBP & OT=LO
$LO_t^{B,1}$	Limit order	Bid	Price = BBP – 1 tick & OT=LO
$LO_t^{B,2}$	Limit order	Bid	Price = BBP – 2 ticks & OT=LO
$LO_t^{B,3}$	Limit order	Bid	Price = BBP – 3 ticks & OT=LO
$LO_t^{A,0}$	Limit order	Ask	Price <= BAP & OT=LO
$LO_t^{A,1}$	Limit order	Ask	Price = BAP + 1 tick & OT=LO
$LO_t^{A,2}$	Limit order	Ask	Price = BAP + 2 ticks & OT=LO
$LO_t^{A,3}$	Limit order	Ask	Price = BAP + 3 ticks & OT=LO
$IOC_t^{B,0}$	Immediate-or-Cancel order	Bid	Price >= BBP & OT=IOC
$IOC_t^{B,1}$	Immediate-or-Cancel order	Bid	Price = BBP – 1 tick & OT=IOC
$IOC_t^{B,2}$	Immediate-or-Cancel order	Bid	Price = BBP – 2 ticks & OT=IOC
$IOC_t^{B,3}$	Immediate-or-Cancel order	Bid	Price = BBP – 3 ticks & OT=IOC
$IOC_t^{A,0}$	Immediate-or-Cancel order	Ask	Price <= BAP & OT=IOC
$IOC_t^{A,1}$	Immediate-or-Cancel order	Ask	Price = BAP + 1 tick & OT=IOC
$IOC_t^{A,2}$	Immediate-or-Cancel order	Ask	Price = BAP + 2 ticks & OT=IOC
$IOC_t^{A,3}$	Immediate-or-Cancel order	Ask	Price = BAP + 3 ticks & OT=IOC
$ICE_t^{B,0}$	Iceberg order	Bid	Price >= BBP & OT=ICE
$ICE_t^{B,1}$	Iceberg order	Bid	Price = BBP – 1 tick & OT=ICE
$ICE_t^{B,2}$	Iceberg order	Bid	Price = BBP – 2 ticks & OT=ICE
$ICE_t^{B,3}$	Iceberg order	Bid	Price = BBP – 3 ticks & OT=ICE
$ICE_t^{A,0}$	Iceberg order	Ask	Price <= BAP & OT=ICE
$ICE_t^{A,1}$	Iceberg order	Ask	Price = BAP + 1 tick & OT=ICE
$ICE_t^{A,2}$	Iceberg order	Ask	Price = BAP + 2 ticks & OT=ICE
$ICE_t^{A,3}$	Iceberg order	Ask	Price = BAP + 3 ticks & OT=ICE

Table 3.3.2 – List of LOB actions available to algorithmic traders that are modelled as output values by the DNN. Action definition refers to the price an order is placed at, which is a function of the best bid price (BBP), best ask price (BAP), and the order type (OT). Ticks refer to parameter τ, or the number of ticks from the best price on the relevant side of the LOB.

3.3.2 ISIN-Date-Broker Tuple Datasets

Modelling the relationship between a specific algorithmic traders' actions, $\mathbf{y}$, and the state of their environment represented by the set LOB *features*, $\mathbf{x}$, is performed by fitting a DNN model to the dataset for that specific trader. Thus, once the LOB features and actions have been extracted from the consolidation FTSE100 dataset, we then split the dataset into a series of sub-datasets filtered by whether the action on the LOB was performed by broker $\mathcal{B}$ in security $\mathcal{S}$ on trading date $\mathcal{D}$. The sub-datasets are represented by the tuple $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$. DNNs can be trained to each individual tuple dataset to model the non-linear relationship, $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, \mathbf{b})$, between the actions performed by the specific algorithmic trader on the LOB and the n-dimensional LOB input feature vector, by optimising the learnable weight and bias parameters. The internal dynamics of the trained DNN can then be analysed to understand what LOB features were important to that strategy which offers information on the type of strategy they are conducting. This analysis and complementary process of segmenting firms by their strategy, then quantifying the impact of the strategy on UK market quality, is performed in the next chapter.

Two caveats are introduced. First, whilst features are developed over the whole trading day, the DNN models only focus on learning from the continuous trading period of 9 am to 4p m. Primarily, this is to extricate strategies conducted during volatile periods in the market, before 9 am, and during pre-auction periods, after 4 pm, which offer minimal utility for comprehending the AT strategies of interest defined in Section 2.3. Second, we consider only trader tuples where a minimum threshold of 1,000 actions are conducted over the trading period. DNNs are highly expressive models though they require a large quantum of data to learn useful representative functions of the relationship between variables. In total 859 DNN tuples $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$ are developed and trained for each valid ISIN-Date-Broker dataset that meet the 1,000 action threshold.

3.4 Baseline Model Experimental Evaluation

The objective of this chapter is to develop an optimised DNN that most accurately models the relationship between LOB features and trader actions in the FTSE100 CLOB dataset. The inherent modular flexibility of DNN design components in conjunction with a multitude of hyper-parameter tuning options available when constructing a network allows for a diagnostic evaluation of what components are most suitable for building a high-quality DNN that fits the FTSE100 CLOB dataset well. The DNN design evaluation involves an Ablation Analysis (Fawcett, 2016) procedure that initialises with an expert-defined 'source' DNN configuration before analysing the impact on the DNNs performance from iteratively modifying parameter settings that lead to a 'target' model configuration. Critical to the experimental nature of the DNN design evaluation is the development of a 'source' DNN which is referred to in this chapter as the 'baseline' model. The baseline model defines the configuration, individual components and hyper-parameters of a neural network, providing a basis from which single components can be augmented to test for their individual impact on performance. These results will inform the optimal choice for the individual components and hyper-parameter values that best optimise the DNN. Evaluation techniques of dichotomised competing architectures inform potential improvements that can be made to the 'baseline' DNN model, given that no specific DNN design is capable of generalising to all machine learning problems (Wolpert, 1997).

This section develops a 'baseline' DNN model using best practices available in the deep learning literature (Bengio, 2012; Kingma, 2015; Clevert, 2015; Ioffe, 2015), before evaluating each element of the network including its configuration, components, hyper-parameters and network architecture. The evaluation is designed to inform improvements to the DNNs design that result in a more accurate predictive model of trader actions give the state of the LOB. The *baseline model* developed in this chapter has a feedforward deep neural network architecture with four-layers comprising 500, 300, 300 and 200 neurons, respectively. The model activates neurons in the hidden layers using Exponential Linear Units bounded in the negative region, whilst output layers use softmax activations. The network is optimised using an Adam adaptive optimisation algorithm with learning rate α of 0.001, β_1 of 0.9, β_2 of 0.95, schedule decay of zero. The baseline DNN is trained with mini-batches of 128 samples, within a cross-entropy cost function framework. Additionally, batch normalisation and dropout layers are attached to each hidden layer with dropout rates of 0.5, 0.4, 0.4, and 0.3. Networks are initialised using orthogonal matrices with gain factor g of one. Finally, the data is split 70/10/20 between training, validation and test datasets, with early stopping techniques employed.

DNN models are trained oo ISIN-Date-Security tuple datasets, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, that include all actions performed by an individual Broker over a single trading day in a single security. This allows for the relationship between individual brokers and the LOB features to be modelled. For the purposes of DNN design evaluation conducted through the Ablation Analysis in this section, and given computational resources, the design evaluation experiments are performed on 59 ISIN-Date-Broker tuple datasets, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, that contain over 10,000 actions executed by brokers. Thus, each dataset has over 10,000 samples to split into a *training dataset* which the DNN is trained on, a *validation dataset* and a *test dataset* which the model is evaluated on for its ability to generalise. This chapter employs a data generator methodology that prescribes a 70/10/20 training/validation/testing ratio for splitting the data using random sampling without replacement. The following sections generally use an average of the DNN performance across the 59 models to allow for a more comprehensive evaluation of the optimal DNN design components when modelling the FTSE100 CLOB dataset.

Experimental evaluation of the DNN configuration is performed by perturbing single components of the model before testing the impact of the elemental augmentation on accuracy and generalisation performance. Results of this analysis help derive recommendations for how the baseline DNN design can be improved and optimised. For example, we evaluate the performance of using ELU activation functions by comparing the performance of baseline models with ELU against the exact same baseline model except that the activation function is changed to alternative functions, such as ReLU or hyperbolic tangent.

Design *components* and *configurations* evaluated include the choice of activation function, bias terms, weight initialisation strategies, cost functions, optimisation algorithms, connectivity patterns, dimensionality and depth of the network, batch size, regularisation and normalisation techniques. For each of these components, a holistic experimental evaluation is performed to determine the best element and hyper-parameter settings used to build the optimal DNN that maximises performance. In relation to *hyper-parameter* evaluation, it is sufficient to perform a 'grid search' by manually defining the hyper-parameter search space given the high quality and richness in depth of the FTSE100 dataset, and the abstract representations inherent in the high-dimensional LOB feature space (LeCun, 1998; Maclaurin, 2015). The series of design evaluation experiments are performed in the following sections.

3.4.1 Rectifier vs Sigmoidal Activated Neurons

Baseline DNN models employ differentiable Exponential Linear Unit (ELU) activation functions to apply non-linearity to each hidden layers' input values whilst correcting for bias shift commonly associated with rectifier activation functions. This section performs an ablation analysis by perturbing the hidden activation function configuration and the hyper-parameter values, to evaluate the utility of different activation functions when predicting trader actions given the state of the LOB. The primary differentiator between activation functions is how they apply mathematical operations to transform inputs as values feed forward through layers in the network, the range that these transformed values take, and the derivatives of these functions which has importance when performing backpropagation. We perform an experiment to assess non-linearity utility by evaluating the accuracy, speed and convergence properties of ELU in our baseline DNN compared against network architecture configurations that utilise various linear and non-linear activation functions common in deep learning academic literature. The set of non-linear activation function formulas fit within either the sigmoidal or rectifier function families, in addition to the basic linear identity function. The first family includes the sigmoid (Rumelhart, 1986), TanH (Rumelhart, 1986), and softplus (Zheng, 2015), whilst the second family includes ReLU (Glorot, 2011), Leaky ReLU (Maas, 2013), Parametric ReLU (He, 2015), ELU (Clevert, 2015) and a Bounded ReLU (Liew, 2016). Output layers maintain the softmax activation function in all the DNNs trained in this section. All activation functions, their formulas, definitions, derivatives and additional information are provided in Table 3.1.1 in the previous section.

Results for DNNs trained with ELU and eight comparative DNNs trained with various identity, sigmoid and rectifier activation functions for the given hyper-parameters are reported in Table 3.4.1.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Activation	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
Identity	59	81.38%	0.572	0.062	24.46%	0.796	0.100	777	8.4	92
Logistic Sigmoid	59	92.82%	0.214	0.034	4.41%	0.159	0.035	744	8.6	86
Hyperbolic TanH	59	95.43%	0.144	0.027	3.87%	0.142	0.031	677	8.6	79
SoftPlus	59	93.27%	0.199	0.033	4.40%	0.169	0.037	747	9.1	82
ReLU	59	96.74%	0.108	0.022	2.91%	0.118	0.021	617	8.7	71
LReLU (a=0.1)	59	96.70%	0.111	0.023	2.97%	0.119	0.023	615	9.0	68
PReLU (γ=0.1)	59	96.83%	0.106	0.022	2.81%	0.114	0.022	696	10.0	70
BReLU (T=0.1)	59	96.48%	0.118	0.023	3.03%	0.120	0.023	593	8.9	67
ELU (a=0.1)	59	96.84%	0.106	0.023	2.97%	0.119	0.023	656	9.2	71

Table 3.4.1 – Tabular activation function ablation evaluation of baseline DNN ELU activation functions against eight comparative identity, sigmoidal and rectifier functions. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

Evidently, ELU performs effectively as a non-linear activation during the training of the DNN models, with an average model accuracy of 96.84% and generalisation error of 2.97 %, converging on average after 71 epochs and 656 seconds. ELU activation-driven DNNs tend to outperform all other models on average given their high training accuracy and low generalisation error, with PReLU activations initialised with zero γ being the most competitive the closest with training accuracy of 96.83% though taking an additional 5.7% longer to converge than ELU models. In addition to ELU and PReLU, both

LReLU and ReLU activation functions have average generalisation errors less than 3%. Clearly, the application of rectifier vis-à-vis sigmoidal non-linearity activations in the DNN models for the FTSE100 data results in both superior model accuracy, convergence speed and ability to generalise outside the training dataset. This is in line with findings from research in acoustic speech recognition models (Maas, 2013; Zeiler, 2013) and image recognition (Glorot, 2011; Krizhevsky, 2012).

From all activations tested, only identity functions lead the model to converge in a sub-optimal local minimum, with a high generalisation error of 24.46% and standard deviation of 0.1. Of the sigmoidal family, logistic sigmoid activations perform poorly during both training, with a training loss of 0.214, and inference, with the highest average generalisation error of 4.41%. Conversely, the hyperbolic TanH activation attains the lowest average generalisation error of 3.87% and highest training accuracy of the sigmoidal family, converging to a local minimum with training accuracy of 95.43% and training loss of 0.144. DNNs with hyperbolic TanH activations also take an average 70 seconds and three epochs faster to convergence than zero-bounded softplus activations. The improved accuracy performance from employing the symmetric about the origin hyperbolic TanH versus the non-symmetric sigmoid activation function in the DNN can be theoretically explained by two factors. First, sigmoid activations introduce systematic bias into the hidden layers of the DNN due to the nature of our dataset, where input feature values are scaled into the range [-1,1]. This scaling process results in features coalescing more effectively during training for DNNs configured with hyperbolic Tanh activations. Second, less 'squashing' of the input activation value is performed by hyperbolic TanH activations vis-à-vis logistic functions, allowing more gradual gradients to propagate through the network, decreasing the risk of the algorithm becoming caught in a non-optimal local minimum basin (LeCun, 1998).

The results for TanH and softplus are competitive with the basic ReLU rectifier function which has a training accuracy of 96.74%, only 1.31% higher than hyperbolic TanH configured networks. However, whilst ReLU's average speed per epoch of 8.7 seconds is slightly slower than the hyperbolic TanH function, we find that ReLUs converge in an average of only 71 epochs, whilst TanH takes 79 and softplus 82 epochs. One explanation for the respectable results of sigmoidal activations is the large sample size of the FTSE100 dataset that has shown to improve convergence and generalisation abilities for DNN models with sigmoidal-activated neurons (Daqi, 2003). The accelerated rate of convergence using ReLU activations is in line with the findings of Krizhevsky (2012), Zeiler (2013), and Maas (2013), given its ability to ameliorate the vanishing gradient issue due to the functions non-saturating regime (Hochreiter, 1998). Furthermore, ReLU activations superior training and speed performance over s-shaped sigmoidal functions is grounded in the lower level of complexity required in the mathematical operations performed to calculate forward propagated values, given their basic zero threshold max function, and their derivatives, due to its binary nature and piecewise linear form.

A comparison of the training accuracy and generalisation error versus the number of epochs for linear, sigmoidal and rectifier activation functions is presented using training and generalisation curves in Figure 3.4.1. In line with the results described above, all rectifier functions converge to high degrees of accuracy, though from the graph it can be noted that not only do sigmoidal activations model the inherent DNN input-output relationship less successfully, but it takes longer for these models to find a basin on the error surface with relatively low cost function error values. Sigmoid activation models do not attain a training accuracy over 80% until 18 epochs have been completed on average, whilst SoftPlus activation models take 11 epochs. In comparison, ELU DNNs take an average of 4 epochs to reach this level of

training accuracy. The primary defect endemic to sigmoidal activations resides in preponderant nature of exploding or vanishing gradients when learning during backpropagation, an inherent architectural design flaw that causes saturated gradients, slow and stunted learning (Hochreiter, 1998; Krizhevsky, 2012). We attribute the slower convergence and lower accuracy of our sigmoidal activations to the tendency of sigmoidal activations to saturate at their functions asymptotes. This is especially the case given the deep nature of the baseline models architecture which has a compounding effect when multiplying gradients through the network during learning, prohibiting signals from propagating backwards and slowing the convergence process which risks resting in a suboptimal local minimum.

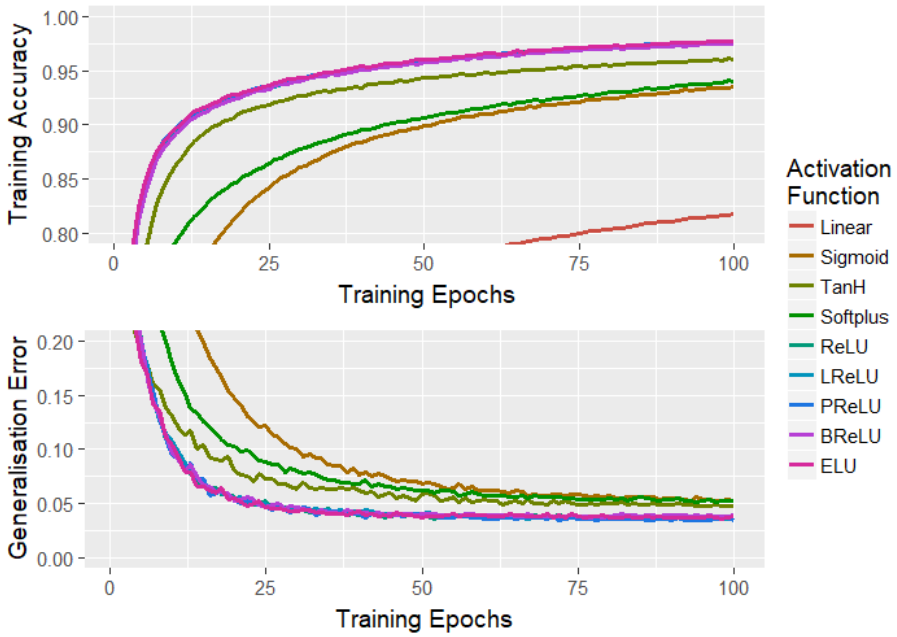

Figure 3.4.1 – Graphical comparison of accuracy and generalisation error for baseline ELU-activated DNN versus eight DNNs trained with augmented activation functions.

Different variants of rectifier functions all attain similar optimisation performance and representation quality, as evidenced by the graphical comparison of the models provided in the above figure. Interestingly, vanilla ReLU performs relatively well compared to other rectifier functions, attaining accuracy of 96.74% after an average of 71 epochs. This performance is on par with other rectifier functions that allow small negative slopes defined by slope coefficient hyper-parameters, including parametric (PReLU), leaky (LReLU), and exponential (ELU) ReLU. One would expect vanilla ReLU activations bounded at hard zero with identity values for positive arguments and zero value for negative arguments to improve on sigmoidal activations given their derivative for positive values is non-contractive, thus alleviating the concern of gradients exploding or vanishing, and due to their ability to introduce sparsity into the network (Hochreiter, 2001; Glorot, 2011). However, neurons with ReLU activations risk 'dying' when they receive negative inputs during training and as such never update weights during learning, with the neurons having zero gradients the whole time during training such that they are extricated from the data manifold. Sparse representations developed in architectures with ReLU activation are both biologically plausible and allow for the disentangling of abstract features or 'hidden information' in dense models. For deep networks there exists a trade-off in ReLU activations where sparsity injected into the network from zero-activation neurons come at the cost of these neurons preventing backpropagated gradient signals flowing backwards through these neurons.

The implementation of rectifier activations that allow negative values, such as LReLU, PReLU, and ELU, was designed to shift mean activation values towards zero and allow for small non-zero gradients in the negative region capable of propagating errors back through the network, whilst trading off the hard-zero sparsity property of vanilla ReLU. However, as identified in Table 3.4.1 PReLU, LReLU and ELU attained very similar results during training optimisation, suggesting that introducing small non-zero gradients to correct mean biased activation values greater than zero may not necessarily improve training optimisation. This aligns with research comparing the activation functions in acoustic models (Maas, 2013). PReLU-based models take the longest of all rectifier-type DNNs to converge and train for a single epoch, requiring an average of 10 seconds per epoch training time and 696 seconds for the model. The rationale for this result is based on the additional training time required for optimising the extra learnable coefficient slope parameter, a, during backpropagation.

3.4.1.1 Activation Function Hyper-Parameter Search

Hyper-parameter choice for rectifier activation functions that control the coefficient of leakage and boundary constraints are critical for optimisation performance and assessed in this section. We conduct a grid search of different rectifier hyper-parameters to evaluate them against the baseline models' choice of ELU with α of one. Leaky ReLU coefficient of leakage, a, defines the slope for negative input values which we tune to $a \in 0.01, 0.1, 0.3, 0.5$, where an alpha between 0.01 and 0.3 is commonly recommended in evaluation studies (Maas, 2013; Agostinelli, 2015; Xu, 2015). Parametric ReLU optimises the coefficient of leakage during backpropagation, though it requires an initialisation argument for gamma γ which we set at $\gamma \in 0, 1$, for evaluation purposes (He, 2015). Bounded ReLU activations are ReLU functions with a positive region bounded at value T, which we test for at values $T \in 1, 10$. Finally, ELU activations bound the negative region with constant alpha, a, which is evaluated for values of $a \in 0.01, 0.1, 0.5, 1$. The comparative analysis of all DNN models tested are presented in Table 3.4.2.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Activation	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
ReLU	59	96.74%	0.108	0.022	2.91%	0.118	0.021	617	8.7	71
LReLU ($a = 0.1$)	59	96.70%	0.111	0.023	2.97%	0.119	0.023	615	9.0	68
LReLU ($a = 0.05$)	59	96.56%	0.115	0.022	2.92%	0.117	0.022	617	9.0	69
PReLU ($\gamma = 0$)	59	96.83%	0.106	0.022	2.81%	0.114	0.022	696	10.0	70
PReLU ($\gamma = 1$)	59	95.08%	0.151	0.029	3.27%	0.134	0.028	835	10.0	83
BReLU ($T = 1$)	59	94.43%	0.175	0.031	3.96%	0.151	0.027	586	8.9	66
BReLU ($T = 10$)	59	96.48%	0.118	0.023	3.03%	0.120	0.023	593	8.9	67
ELU ($a = 0.01$)	59	96.63%	0.112	0.022	2.88%	0.118	0.022	626	9.2	68
ELU ($a = 0.05$)	59	96.77%	0.108	0.023	2.95%	0.120	0.022	642	9.2	69
ELU ($a = 0.1$)	59	96.84%	0.106	0.023	2.97%	0.119	0.023	656	9.2	71
ELU ($a = 0.5$)	59	95.95%	0.134	0.023	3.30%	0.129	0.025	590	9.2	64
ELU ($a = 1$)	59	95.44%	0.148	0.028	3.68%	0.144	0.030	606	8.8	68

Table 3.4.2 – Tabular activation function hyper-parameter ablation evaluation of rectifier functions with varied hyper-parameter values. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

LReLU activations allow an expert-defined small non-zero gradient slope equal to the coefficient of leakage, a, when a neuron activates in the negative region, whilst the PReLU activation learns this parameter value during backpropagation. The LReLU results for setting the coefficient is mixed with an a of 0.01 attaining a higher training accuracy but worse generalisation error than the model trained with the higher 0.05 alpha, with both models taking a similar time to converge. Additionally, the results mirror those of the ReLU trained model, indicating that the LReLU's objective of minimising the risk of saturation and shifting the mean activation towards zero does not significantly improve network accuracy even with steep negative slopes. An interesting result when studying the PReLU activation function is the impact that initialising the slope coefficient a has on the ability of the DNN to optimise into a good parameter space. Initialising the hyper-parameter to zero improves training accuracy by 1.75% points and reduces generalisation error by 0.56% points on average over models initialised with a unit slope. This indicates that DNNs are more conducive to learning the optimal slope from the basis of a vanilla ReLU rather than a rectified function with a steep slope in the negative region. Results for the BReLU are in line with expectations. Bounding the ReLU function with a value of one in the positive region does not improve training accuracy or decrease generalisation error over the vanilla ReLU. Even a higher boundary of 10, such that input activation values less than 10 follow an identify function whilst higher values are bounded at 10 with zero gradient, leads to an average 0.01-point increase in the training datasets categorical cross-entropy error and a 0.12% increase in generalisation error over models trained with ELU.

The baseline model utilises ELU as an activation function which according to Table 3.4.2 performs well across all a slope coefficients tested in the hyper-parameter search region. Evident from the table, training accuracy tends to follow an inverse U-Shape, increasing up to the alpha 0.1, before reducing slightly. Generalisation error on the other hand increases as the coefficient begins to rise. This indicates that there is a good region of alpha values around 0.1 which train DNN models with good predictive performance. Thus, when optimising the DNN models design, it would be more pertinent to lower the ELU alpha levels closer to 0.1, limiting the negative asymptote of the function and shifting the function closer to a vanilla ReLU. The benefits of performing this modification, such as an increase in sparse representations, are evident in the next section which explores these agreeable properties of ReLU functions when training DNNs.

3.4.2 Sparse Representations & Bias Shift Correction

This section studies the internal dynamics of the trained DNNs to evaluate the ability of rectifier activations to stimulate sparse representations in hidden layers of the model and ELU activation functions to correct for bias shifts common in vanilla RELU activated networks. Neurons in the network activate when they pass information through the network during forward propagation of the input feature vector, $\mathbf{x}$. After training, validation data is passed through the trained DNN to derive the activation values of each neuron by computing the value that the neuron outputs into the adjacent hidden layer.

3.4.2.1 Bias Shift Correction Using Exponential Linear Unit Activations

DNNs with ReLU activation functions have an inherent architectural bias of activating neurons with distributions centred in the positive region. This characteristic of ReLU networks is caused by the hard-zero nature of the function, which does not align with the properties of the UAT (Hornik, 1991), and can be corrected for by utilising ELU in place of ReLU activations. This section analyses the internal state of the DNN to understand how neurons fire and to what degree. This allows for an evaluation of the impact of ELU, ReLU and sigmoid functions on the activation values of the network, with summary statistics for networks trained with each function shown in Table 3.4.3.

ELU activations reduce the mean output activation value of all neurons in the first hidden layer from 0.445 for RELU activations to 0.054 for ELU activations, augmenting the mean closer towards zero, confirming the bias shift correction hypothesis of Clevert (2015). This property of ELUs corrects for the bias shift inherent in non-zero mean rectifier activation networks, thus, leading to a more accurate alignment of the natural and standard gradient of the DNN, culminating in improved learning speeds and more reflective weight values (LeCun, 1998; Clevert, 2015).

	Neuron Activation Values			
Activation	**Mean**	**SD**	**Min**	**Max**
ELU	0.054	1.66	-1	6.02
ReLU	0.445	1.19	0	5.88
Logistic Sigmoid	0.179	0.31	0	1.00

Table 3.4.3 – Neuron activation values in the first hidden layer of the trained DNNs. Results are computed over the 59 DNN evaluation Broker dataset tuples and winsorised at the 99% level. Note that logistic sigmoid have minimum and maximum *asymptotes* of zero and one, respectively, whilst ELU has an asymptote in the negative region of minus one, represented by the alpha hyper-parameter.

The shift in mean activation can be graphically shown in Figure 3.4.2, which compares the logarithmic density distributions for models trained with the three tested activation functions. ELU functions in essence shift the negative bound by one unit relative to ReLU activations, allowing gradients to propagate through the network even when neurons have a negative value. The figure confirms the result and theoretical expectations that ELUs have activation distributions closer to mean zero vis-a-vis RELU, which as an architectural trait skew into the positive region given their hard zero output. The smooth increase in ELU density distributions as values approach the negative bounded region can be contrasted with the sharp hard-zero cut-off for ReLU activations, as evident in the figure. By allowing for the function to smoothly saturate as activation values approach the negative asymptote, ELU activations are able to attain a noise-robust deactivation state (Clevert, 2015). This arises from the fact that saturated regions allow small gradients and information about input concepts to flow back through the network during learning, even if the quantum of this data is minimal. This property is evident in ELU activations that have a saturation region near the asymptote. Similarly, ReLU DNNs retain the property of deactivation states given the image of their negative region is zero, serving as a quasi-saturation region (Clevert, 2015; Unterthiner, 2015). Saturation regions place a bound on both activation and gradient values in the negative region, thus, they stop noise permeating from very high negative activation values propagating through the network resulting in a noise-robust activation state. In contrast, other non-bounded negative slope rectifier functions such as LReLU and PReLU do not attain this state given the non-saturating property of the activation function in their negative domain. This allows noise to be injected into the model as output values can take any value over the real-number domain. The primary

advantage of activations with noise-robust deactivation states is their ability to reduce variation in propagated values, by bounding output activation values to an extent, allowing the dependencies between ELU neurons to be more effectively modelled with a more robust representation space.

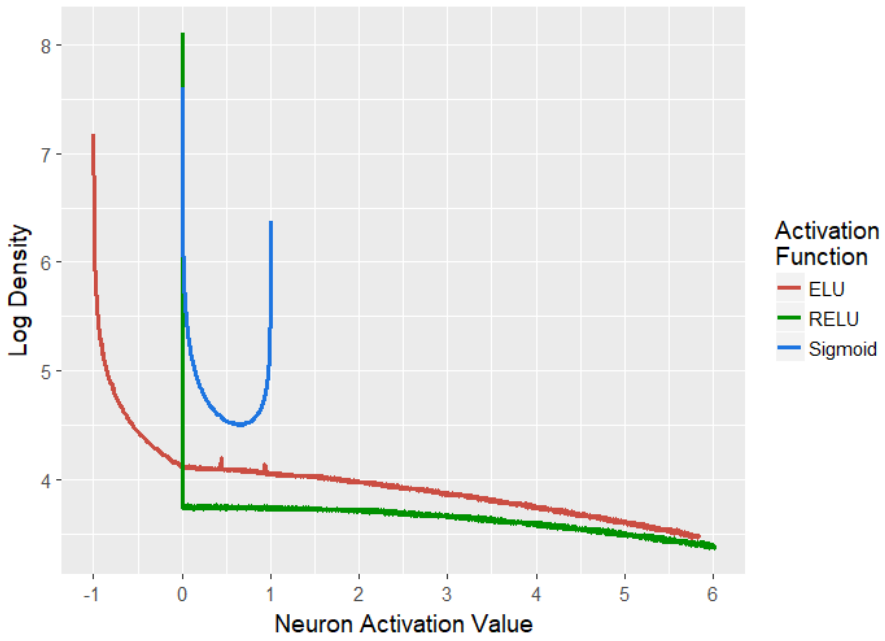

Figure 3.4.2 – Neuron activation value log density distributions for ELU, ReLU and Sigmoid activation functions. Values are winsorised at 99% and represent the activation output for all neurons in the first layer across the sample datasets.

Figure 3.4.2 also provides an exhibit of the semi-symmetrical distribution properties of sigmoidal activated neurons which centre on mean 0.179 with a standard deviation of 0.31. Furthermore, one can note the explicit preponderance of sigmoidal functions to saturate at their asymptotes, given the sharp boundary U-shape distribution evident in the figure. However, one would expect these saturation issues to be most prevalent in the first layer of a deep network, which is the layer studied in this experiment, as errors propagate *backwards* such that the number of multiplicative operations performed to compute the gradients increases as one counts layers backwards from the output layer. Nevertheless, the graph also confirms previous concerns that sigmoid activations do not have agreeable distribution properties that fit the underlying FTSE100 dataset used to train the DNNs. The FTSE100 dataset is composed of financial LOB metrics scaled into a range (-1,1) with varied distributions, which does not align with the outputted range of the logistic sigmoid function, (0,1), and may be a cause of the saturation evident in Figure 3.4.2 and the lower accuracy attained by sigmoidal DNNs compared to rectifier functions as presented in the previous section.

Further examination of the bias shift correction properties of ELUs requires a comparison of the mean neuron activation values during training to verify that the functions provide a stable optimisation process that induces mean activations closer to zero. DNN's are trained for each of the 59 ISIN-Date-Brokers evaluation datasets with the internal state of each model studied at the culmination of a learning epoch. This allows for the neuron activation values of the first layer to be computed by feeding the test dataset into the trained model with the output being the 500 neurons in the first layer. Figure 3.4.3 depicts the evolution of mean neuron activation values, with attached 90% confidence intervals, as training epochs increase. Evidently, ELU activated DNNs initialise close to zero mean activation, decreasing during the early stages, before slowly rising until it plateaus at mean activation around 0.5 after approximately 50

epochs. DNNs trained with ReLU functions initialise with a low mean activation of 0.25 which stabilises over the first few epochs before rising in line with ELU. As a comparison, Sigmoid activations induce a more stable mean neuron activation state, which is most generally caused by the saturation of neuron values at the asymptotes.

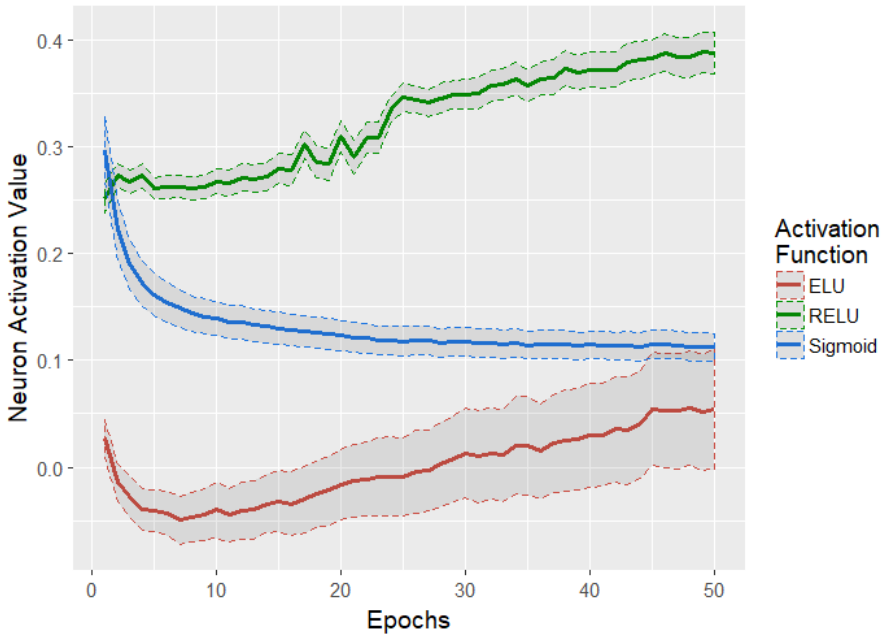

Figure 3.4.3 – Mean neuron activation values during iterative training epochs for the 59 DNN evaluation models trained. Mean activations are computed by analysing the internal dynamics of the DNNs after each training epoch. Confidence intervals represent 90% sample bands.

After evaluating the internal dynamics of the DNNs in this section, one can conclude that the ELU activations succeed at performing a bias-shift correction for vanilla ReLU activated DNNs by centring mean neuron activation values closer to zero. Furthermore, ELUs allow for a noise-robust deactivation state, with these theoretical performance-inducing characteristics of these activation functions aligned with the superior accuracy and generalisation results attained in the previous section for DNNs with ELU activated neurons.

3.4.2.2 Rectifier Functions Induce Sparse Representations

Rectifier functions inject sparsity-inducing components into the optimisation learning objectives of neural networks, aiding in the performance of these networks in modelling the input-output non-linear mapping function (Nair, 2010). Hard-zero bounded rectifier linear units in the negative region were designed to introduce discontinuities in neuronal activity, such that neurons would not *always* 'fire', resulting in DNN models capable of learning more sparse representations. These techniques succeeded at aligning machine learning with the sparse and distributional properties of biological neurons (Glorot, 2011; Attwell, 2001) which analogously have an 'off' switch where they remain dormant for periods. Sparse networks have the advantage of being able to build highly abstract feature combinations deeper in the network to more accurately model the mapping function and reduce overfitting (Glorot, 2011).

This section evaluates the sparsity-inducing capabilities of different rectifier and sigmoidal activation functions, by using an approximate measure for each neurons probability of activation. The measure is empirically computed as the percentage of neurons in the first hidden layer that *activate* into a region

with *useful* information coded into the gradients, as opposed to a saturation region. Neurons that activate pass information through the network, conversely, those that do not activate or 'saturate' create discontinuities in the network that stops information flowing through the neuron during both forward and backpropagation. The neurons probability of activation serves as a proxy for the level of sparsity injected into the network by the relevant activation function. RELU neurons are taken to activate when their activation values are positive, $\boldsymbol{\phi}(\boldsymbol{z}) > 0$, ELU neurons are assumed to activate when $\boldsymbol{\phi}(\boldsymbol{z}) > -0.75$, whilst sigmoidal functions activate when activation values are within a conservative non-saturation region away from the asymptotes, $0.9 > \boldsymbol{\phi}(\boldsymbol{z}) > 0.1$. Additionally, 'saturation' probabilities are estimated as the complement to the activation probability. The activation values of the evaluation dataset are winsorised at the 95% level to account for spurious outlier activations.

Comparison between the activation functions probability of activation and saturation are presented in Table 3.4.4.

	Neuron Activation Probabilities			
Activation	**Mean**	**SD**	**Min**	**Max**
ELU	41.9%	0.059	34.9%	71.5%
ReLU	20.4%	0.033	7.6%	36.5%
Logistic Sigmoid	22.2%	0.047	8.8%	40.8%

Table 3.4.4 – Neuron activation probabilities for neurons in the first hidden layer of the trained DNNs. Results are computed over the 59 DNN evaluation Broker dataset tuples and winsorised at the 95% level. Saturation probabilities are computed as: 1 - Activation probability.

RELU activation functions are shown to induce sparse representations given that only 20.4% of ReLU neuronal units are activated on average. In comparison, whilst ELU networks activate at all real values, only 41.9% of neurons on average allow signals to flow through the network during learning without significantly dampening the quantum of information being backpropagated. These neurons are assumed to activate given their activation values are greater than 0.76. The hard-zero bound of the ReLU activation prohibits the optimisation algorithm passing *any* gradient information through non-firing neurons that activate in the negative region. Whilst ELU activations have a quasi-hard-zero property given the asymptote for activation values in the negative region, all neurons in an ELU-activated DNN allow some quantity of information to flow through them during learning, though this comes at the cost of less activation function-induced sparsity.

In contrast to ReLUs and similar to ELUs, sigmoid activations activate at all real values. Theoretically, these functions induce a steady state mean activation of 0.5 units, a property of sigmoid DNNs that both stunts learning and is biologically implausible (Bengio, 2012). However, results in Table 3.4.4 indicate that DNN's trained on the broker FTSE100 datasets perturb weights in a way that neuron activation distributions are denser in regions closer to zero with a mean neuron activation 0.179 across the evaluation datasets. This may be caused by the sparse nature of the input feature space due to the fact that certain LOB features may have minuscule or uniform values. Financial time series data is strongly auto-correlated with some subsets of features maintaining similar values for certain event periods of the dataset. This inherent trait of the dataset may result in neurons activated with sigmoid function taking on lower values as the input feature vector propagates forward through the network, explaining the mean activation being closer to zero than 0.5. A common complaint regarding sigmoid activations in deep networks is their prevalence of suffering from the vanishing gradient problem (Hochreiter, 1998), a phenomenon existent within the sigmoid-activated DNNs trained with the

FTSE100 dataset. This is confirmed by the high saturation probability of these models, 77.8%, and very high density of activation values near the asymptotes of the function where gradients approach zero. Vanishing gradients impact the ability of the network to optimise quickly using gradient-descent and may lead to the algorithm converging into a sub-optimal minimum (LeCun, 1998), with theoretical limitations of sigmoid functions aligning with the relatively poorer accuracy reported in the previous section.

The level of neuron dispersion in a DNN is another useful proxy for networks ability to produce sparse representations. Dispersion is a quantitative measure of neurons ability to respond uniquely to different stimuli in the input feature values, such that the probability of each neuron in the network of would fit within a tight range. The measure can be mathematically computed as the level of dispersion or standard deviation between all neuron activation values in a network hidden layer (Willmore, 2000; Maas, 2013). Figure 3.4.4 graphically depicts the level of dispersion between neuron activation probabilities for the first hidden layer of DNNs configured with ELU, ReLU and sigmoid activation functions and trained over the evaluation datasets. For DNNs trained with a specific activation function, the probability of activation for each neuron in networks first layer is computed. The probability values are then ranked from highest to lowest, with neurons then placed in 1000 equally sized 'bins' ranked in descending order by their probability of activation. An analysis of the figure can be performed in conjunction with the activation probability standard deviation values computed in Table 3.4.4. Evidently, ReLUs produce sparser representations than ELU and sigmoid activated DNNs given the flatter activation probability distribution curve and lower standard deviation of 0.033. The figure provides graphic evidence of the ReLUs sparsity inducing properties given that a more uniform distribution implies that information coded in signals propagated through the network are more dispersed across the neurons, as opposed to signals being concentrated in fewer neurons.

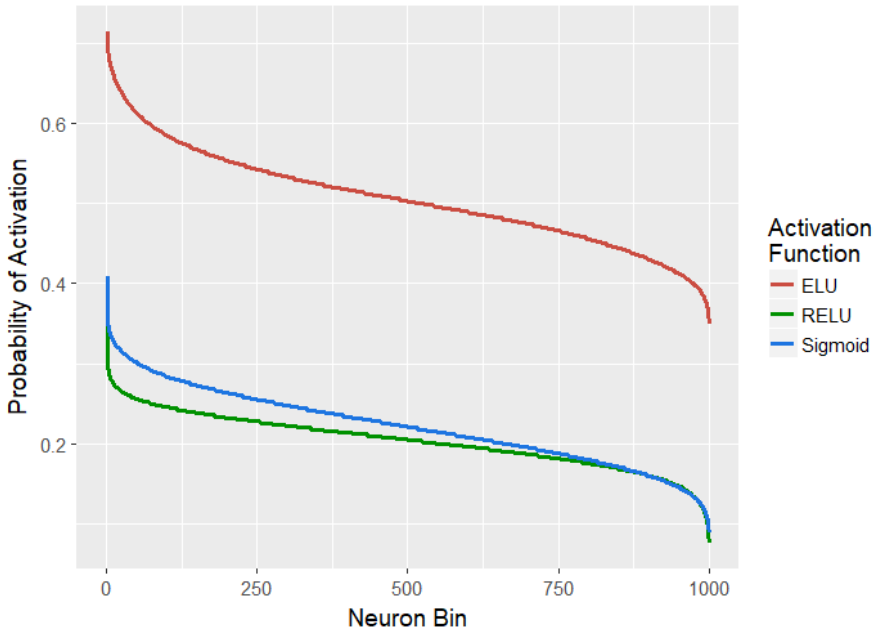

Figure 3.4.4 – Comparison of probability of neuron activation dispersion levels for DNNs configured with ELU, ReLU and Sigmoid activation functions. Each neuron in the trained network is subscribed to a bin based on their position in descending order of all neuron activation probabilities. The mean activation probability for the bin is computed and represents the data point in the graph.

From this analysis we can see that ELU and ReLU hold both theoretical and empirical advantages over DNNs configured with sigmoidal activated neurons, aligning with the performance results computed in

the previous Section 3.4.1. Results have also shown that DNNs trained with ELU activation functions have attained superior performance over ReLU when trained on the FTSE100 dataset. Furthermore, ELUs have strong theoretical foundations given their ability to perform a bias-shift correction for mean neuron activation values, induce a noise-robust deactivation state, whilst also injecting a degree of sparsity into the network. This section concludes that DNNs configured with ELU activation functions, that tune the alpha a hyper-parameter to 0.1, are more agreeable to the datasets used in this study, thus, are deployed in the optimised DNN model.

3.4.3 Bias Node Irrelevance

The high-dimensional feature space extracted from the FTSE100 dataset appears to constrain any flexibility about the learned function gained from including bias nodes in the hidden layers of the DNN. Bias terms allow the transfer function, which performs an affine transformation taking the dot product of the input vector and connecting synaptic weights, to be shifted by bias term b before undergoing non-linearity when passed through the activation function. This provides the DNN with a trainable constant whilst maintaining the shape of the transfer function. Table 3.4.5 compares the average performance of the baseline DNN model configured with and without bias terms. Evidently, the model with no bias terms has a 0.24% higher training accuracy, though slightly higher generalisation error and converges on average 3 seconds slower than the model that includes bias terms.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Bias Nodes	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
With Bias	59	95.22%	0.155	0.028	3.66%	0.140	0.031	600	8.9	67
Without Bias	59	95.46%	0.146	0.026	3.74%	0.144	0.031	603	8.8	69

Table 3.4.5 – Ablation evaluation of baseline DNN without bias term compared against DNN models trained with a bias term. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

The purpose of this chapter is to model trader's strategies by understanding how their algorithms respond, in the form of actions, when the state of the LOB changes, defined through a series of LOB features. This process requires an understanding of what features are important to each individual traders' algorithm utilising technique to analyse feature importance metrics introduced in the next chapter. Thus, the addition of a bias term introduces a parameter that provides no utility to this analysis given that the term is likely to bias the synaptic weights connecting neurons which inform those feature importance values. As an additional note, the inclusion of batch normalisation layers in our network placate the need for including a bias term given the inherent normalising nature of these layers that implicitly applies bias adjustments based on the covariance of the neurons input, rendering bias terms less necessary (Ioffe, 2015). Finally, the addition of non-zero bias terms in DNN models may not be optimal for learning in sparse networks where the network is segmented into small regions where the bias term may play an undue role in shifting ReLU activation values away from low error regions (Konda, 2014). The results from the FTSE100 evaluation does not align with this finding, though it should be noted that the baseline models are trained with ELU activations that largely counter the level of network sparsity infused from ReLU activations as fewer neurons activate at zero value. In summary, there is no evidence that including bias terms in an optimal DNN is necessary given the nature of the machine learning problem this chapter seeks to solve and the strong performance of non-bias node DNNs.

3.4.4 Initialisation Strategies

Common kernel weight initialisation strategies are evaluated against the baseline models' orthogonal matrix initialisation with unit gain factor for their impact on training optimisation performance for complex non-linear DNN architectures, their ability to generalise well outside the training data, and their convergence properties. Interestingly, of the five comparative initialisation strategies studied, orthogonal matrix initialisation (Saxe, 2014) with gain factor g of 0.5 attains the highest performance, though only marginally, with an average training accuracy of 95.46%, training loss of 0.147 and generalisation error of 3.82%, converging in 620 seconds on average. We compare these results against DNN models initialised with constant zero values, heuristic models that draw from uniform (Glorot, 2010) and normal (He, 2015) distributions, and random initialisation methods that draw from a truncated normal distribution with mean zero and 0.05 standard deviations, with results reported in Table 3.4.6 and generalisation curves shown in Figure 3.4.5.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Initialisation strategies	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
Constant Zero	59	26.15%	1.941	0.083	73.34%	1.941	0.102	813	8.9	91
Glorot Uniform	59	95.28%	0.155	0.028	3.78%	0.149	0.032	598	8.9	67
He Normal	59	95.41%	0.148	0.025	3.71%	0.139	0.030	621	8.9	70
Orthogonal ($g = 0.5$)	59	95.61%	0.142	0.025	3.75%	0.142	0.030	635	9.0	71
Orthogonal ($g = 1.0$)	59	95.46%	0.147	0.026	3.82%	0.147	0.032	620	8.9	69
Orthogonal ($g = 1.5$)	59	95.15%	0.157	0.029	3.76%	0.145	0.032	597	9.0	67
Trun. Normal (σ=0.05)	59	95.26%	0.152	0.029	3.82%	0.147	0.030	595	9.0	66

Table 3.4.6 – Ablation evaluation of baseline DNN initialised with orthogonal matrices and gain factor g of one compared against DNN models trained with different initialisation strategies. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

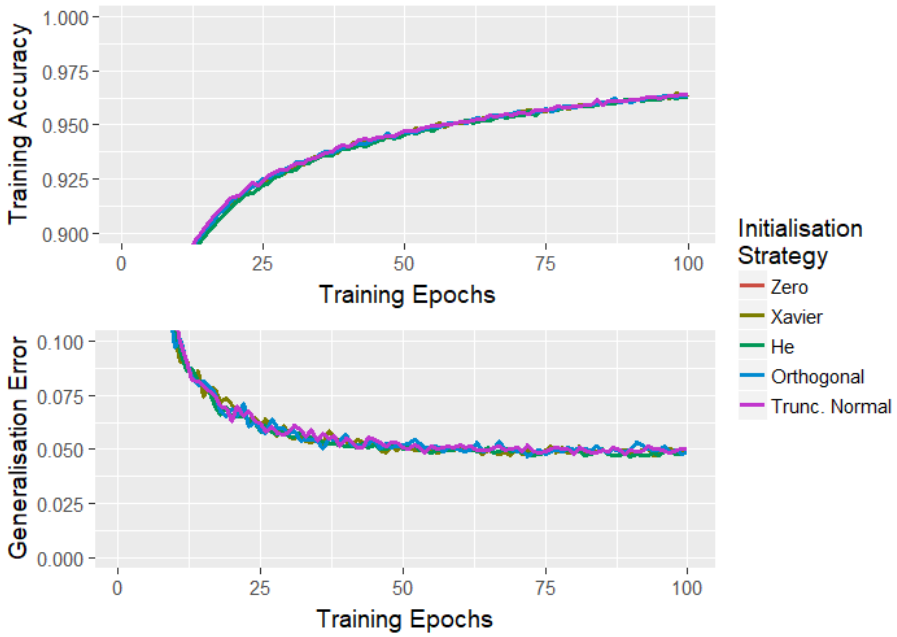

Figure 3.4.5 – Graphical comparison of accuracy and generalisation error for baseline orthogonal matrix initialised DNN versus DNNs trained with augmented initialised strategy configurations.

The heuristic methods developed by He (2015) and Glorot (2010) perform competitively against the orthogonal method, attaining generalisation errors of 3.71% and 3.78%, respectively, as they successfully break the symmetry of initialised values between different hidden layers whilst ensuring that neuron values diverge significantly enough to allow gradients to backpropagate through the network. There is no evidence of whether the He or Glorot model improves accuracy in our model vis-à-vis the other model. However, He (2015) argues that the main divergence in stability and convergence properties between the two is transparent in *very* deep networks, whilst our baseline model is only four layers deep, He uses a 30-layer model. DNN models initialised with weights zero completely fail to converge with their learnt functions providing minimal predictive ability, in line with the DNN literature (Bengio, 2012).

Applying an initialisation strategy that draws from a normal distribution, truncated by two standard deviations, with zero mean and standard deviation of 0.05, performs surprisingly well with generalisation error of only 3.82% and relatively high training accuracy only 0.2% less than the baseline DNN model initialised with random orthogonal matrixes and gain factor of 1.0. However, there is wider dispersion in the training accuracy results with DNNs initialised with truncated normal distributions trained with a standard deviation of 0.029, higher than the orthogonal and heuristic models. The competitive results of the truncated normal distribution align with findings of Hinton (2012c) when training Restricted Boltzmann Machine (RBM) networks. A possible explanation for good but not superior result of truncated normal initialisation is that arbitrary random drawing of weights from a Gaussian distribution during initialisation increases the risk of dampening gradients in deep networks. This issue is amplified as the network increases in depth leading to weaker convergence properties (Simonyan, 2014) and slower learning, particularly if absolute feature input values are large (Mishkin, 2015). Orthogonal matrices overcome this issue through their desirable property of dynamical isometry (Saxe, 2014) proven for deep linear networks, achieved by initialising weights using random orthogonal matrices with eigenvalues of absolute value one. These strategies minimise the risk of vanishing gradients during matrix multiplication across hidden layers, allowing learning to occur as signals with useful information faithfully backpropagate through the network. Random Gaussian initialisation and the heuristic methods of He (2015) and Glorot (2010) do not meet this property.

Another set of DNN models are trained using the same comparative initialisation schemes, however, these models are configured without batch normalisation layers in their DNN architecture. Average generalisation errors for these models are reported in Table 3.4.7. A key result is that batch normalisation evidently has a very important role in training networks, significantly increasing their ability to learn representations in the data and improve their predictive capabilities. Another takeaway is the fact that Orthogonal matrix initialisation with unit gain attains the lowest generalisation error of 11.8%. With the advent of batch normalisation by Ioffe (2015) and its ubiquitous deployment in DNN architectures, initialisation strategies have become less relevant when designing a DNN model. However, the results confirm Saxe's (2014) theory regarding the benefits of dynamic isometry ingrained in the orthogonal matrix initialisation strategy, providing a rationale for adopting the strategy in our baseline DNN models.

The evaluation indicates that orthogonal matrix initialisation is a successful strategy, though accuracy can be improved for the optimal DNN by setting the gain factor lower to 0.5.

Initialisation Strategies	Firms	Gen. Error
Constant Zero	59	72.12%
Glorot Uniform	59	12.29%
He Normal	59	12.12%
Orthogonal ($g = 0.5$)	59	12.28%
Orthogonal ($g = 1.0$)	59	11.80%
Orthogonal ($g = 1.5$)	59	12.18%
Trun. Normal (σ=0.05)	59	12.25%

Table 3.4.7 – Comparison of generalisation performance for DNN models configured with different initialisation strategies and without batch normalisation layers. Other components of the baseline model are maintained.

3.4.5 Cost Function Evaluation

Modelling high-frequency limit order book (LOB) dynamics and trading strategies is a complex task given the various actions a trader can take depending on the state of the market and LOB. Critical to the DNN used to model this relationship is the choice of a robust cost function that accurately calculates the error of the predictions that inform the DNN optimisation algorithms updates to the internal learnable parameters. Cost functions provide a framework to compare the distributions of the training output values with the true output, requiring an easily differentiable mathematical form that allows gradient descent optimisation algorithms to efficiently update network weights in the direction of good representational areas during iterative training. This section performs an ablation evaluation of the baseline DNN models categorical cross-entropy (CE) cost functions empirical performance, learning speeds and convergence properties, by perturbing the cost function into other common forms utilised in the deep learning community. The formulas of the comparative cost function are outlined in Section 3.1.6.

Baseline DNN models that employ categorical CE cost functions attain a generalisation error of 3.65%, training accuracy of 95.72%, and take 596 seconds to converge on average of the 59 evaluation tuple datasets. The use of CE cost functions to train DNN models is prevalent in the deep learning literature (Janocha, 2017), given that it provides an interpretable error that spurs the optimisation algorithm to minimise divergence between the true and predictor output probability distributions. DNNs trained using a CE cost function framework attain the highest training accuracy and speed to convergence on average for the evaluation models. Table 3.4.8 reports the difference between the training accuracy and convergence speed of CE and several comparative cost functions. These include the mean square error (MSE), mean absolute error (MAE), squared hinge loss (SHL), hinge loss (HL), LogCosh (LCOSH), KL Divergence (KLD), and cosine proximity (COS). Figure 3.4.6 graphically presents the comparative analysis of the CE trained DNNs versus comparative models.

DNN Models		Divergence from CE	
Cost Functions	Firms	Training Accuracy	Convergence Speed (Sec)
Cross-Entropy	59	0.00%	0
Mean Square Error	59	-0.10%	5
Mean Absolute Error	59	-0.52%	128
Squared Hinge Loss	59	-0.09%	49
Hinge Loss	59	-0.67%	136
LogCosh	59	-0.05%	70
KL Divergence	59	-0.12%	5
Cosine Proximity	59	-0.24%	64

Table 3.4.8 – Divergence between training accuracy and convergence speed for baseline DNNs trained with categorical cross-entropy and DNNs trained with comparative cost functions. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

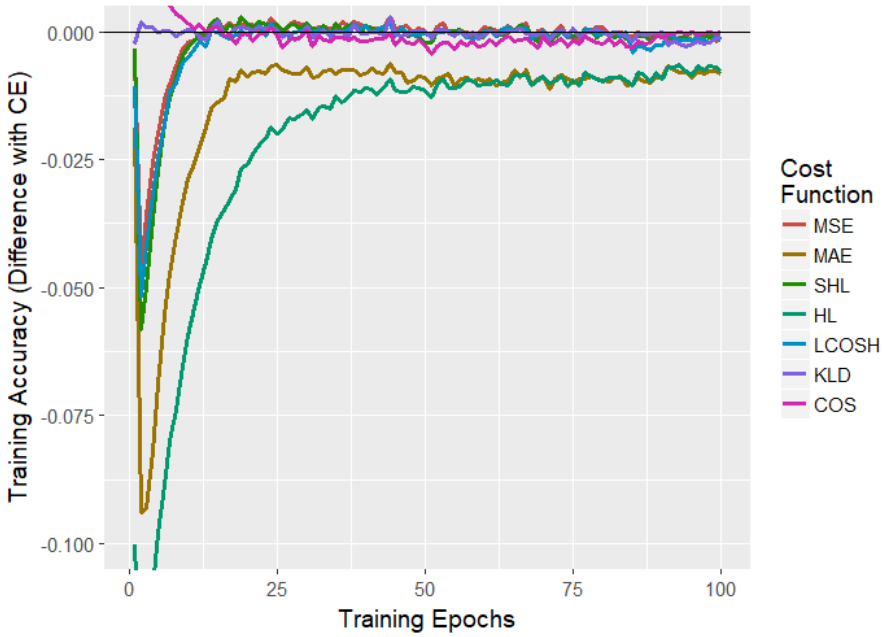

Figure 3.4.6 – Difference during the learning process between training accuracy of baseline DNNs trained with CE cost functions and DNNs trained with comparative cost functions. Values below zero indicate a higher accuracy for CE trained DNNs.

The high-dimensional nature of the dataset allows for the DNN to build very accurate representations, minimising the cost function efficiently as the weights are updated in the model. Neither of the cost functions analysed struggle to learn effectively or spur convergence issues. The first point to note is the relatively inferior performance of models trained with first-order mean absolute error (MAE) and hinge loss (HL) cost functions, attaining on average 0.52% and 0.67% lower training accuracies, respectively, than the baseline cross-entropy (CE) model at convergence. These results confirm the findings of Lee (2015) that second-order mean error and hinge loss functions outperform their non-exponent versions. The slower learning capabilities of models trained with these first-order cost functions are further exemplified by analysing the cost functions training and generalisation curves which shows that models trained with HL and MAE take 9 and 7 epochs, respectively, to reach 80% training accuracy whilst CE trained models reach this mark in an average of 4 epoch iterations. The multi-category nature of the output classifier impacts the dynamics of the optimisation process for MAE given that the cost function is theoretically more suitable for solving binary class problems. MAE cost function gradients generally

take on the predictor value, one or zero, which leads to weights jagging around a potentially non-optimal region of the error surface during optimisation.

Mean square error (MSE) is another cost function commonly deployed in DNN models with its performance in the experiment being comparable to CE cost functions, averaging only 0.1% less in training accuracy averaged over the evaluation datasets. In comparable environments and under certain conditions, deploying cross-entropy cost function leads the DNN to converge faster and at a more optimal local minimum than the MSE due to the inherent structure of the MSE derivative when using softmax output activation functions (Golik, 2013). The logic for this becomes apparent when computing the partial derivatives $\delta\mathcal{L}/\delta w_i$ of the cost function $\mathcal{L}$ w.r.t each weight w_i. The partial derivative for the MSE cost function includes a term for the derivative of the activation function, $\boldsymbol{\phi}'(\mathbf{z})$, in hidden layers, and $\boldsymbol{\varphi}'(\mathbf{z})$ in the output layer. When applying a softmax, $\varphi_{SoftMax}$, output activation function, the derivatives of the output activation function, $\boldsymbol{\varphi}'(\mathbf{z})$, tend towards zero for large positive or negative activation input values, $\mathbf{z}$. The practical implication here is that the gradients approach very low values which slows down the learning process considerably. To allay this issue the CE cost function is devised in such a way that its partial derivative excludes the activation function derivative term, $\boldsymbol{\varphi}'(\mathbf{z})$, which gets cancelled out when applying the chain rule during learning. Thus, whilst MSE and CE are both positive for all real value inputs, and the cost functions tend towards zero as the predictors closely approximate the actual output values, CE has the added advantage of propagating larger errors through the network resulting in faster learning and convergence (Goodfellow, 2016).

It is worth noting the 0.05% improvement in training accuracy when using log hyperbolic cosine (LogCosH) cost functions to train the DNNs compared to when using the MSE error. In fact, LogCosH networks actually attain the lowest generalisation error of all the comparative cost functions analysed, with only a 0.05% lower average training accuracy that DNN trained with CE cost functions. Furthermore, the LogCosH also has the lowest generalisation error after CE of the comparative cost function frameworks, attaining an average error 0.14% more than CE, indicating its successful ability to inform good predictions of trader actions outside the training dataset. Both MSE and LogCosH cost functions perform similar mathematical operations within a quadratic framework, though LogCosH errors are impacted by a slightly smaller magnitude when the model outputs a significantly incorrect predictor outlier. The slightly better performance of LogCosH indicates that some classifications may be wildly incorrect, generally iceberg and immediate-or-cancel orders at the BBO that get mixed with limit orders, which the LogCosH does not punish as harsh as MSE cost functions, could be a potential explanation for their improved headline performance. The advantage of not punishing significant outlier predictions is that the weights perturbations generated are less exaggerated when an incorrect prediction is made, thus, the network does not become destabilised during training which is a risk for DNNs with an MSE cost function.

The similar performance of DNN models trained with cross-entropy and Kullback-Leibler (KL) Divergence cost functions can be theoretically explained by the significant alignment of both functions' probabilistic assumptions and their identical interaction with the softmax activation used in output layers. In the context of machine learning optimisation, KL divergence and categorical cross-entropy solve equivalent problems given that KL Divergence is defined as the cross-entropy between the correct classifier and the model predictor minus the entropy of the true class empirical distribution, a fixed variable that does not vary when model parameters fluctuate. Thus, the results should be near identical

for both cost functions, however, one explanation for the difference could be that several classes have very low probability values in the class distributions, specifically iceberg orders. This potentially could serve as a type of regularising effect where the KL cost function punishes incorrect predictions of this class slightly different to cross-entropy given the inclusion of the fixed true probability entropy in its function.

Finally, the good training accuracy of DNN models trained with the squared hinge loss (SHL), averaging only 0.09% less than CE cost functions, is in line with academic research (Tang, 2013; Janocha, 2017), and presents an alternative cost function for use in DNNs over the more traditional CE and MSE forms. SHL cost functions ameliorate concerns regarding the non-differentiable nature over the continuous domain of its first-order form by squaring the hinge loss. This could potentially explain the better performance of SHL over HL cost functions in the DNN models. The non-smooth nature of the SHL leads to a faster training time per epoch over models using CE cost functions, however, they take significantly more epochs to converge resulting in an average additional time of 49 seconds to converge over CE models.

The decision to select categorical CE is based on the architectural design of our model, wide and deep with ELU activations in the hidden layer. Furthermore, the DNNs trained with CE are conducive to the flow of gradients backwards through the model given that the functions partial derivative excludes the derivative of the output activation, $\boldsymbol{\varphi}'(\mathbf{z})$. This rationale is confirmed by the superior performance of DNN's trained with CE cost functions over other comparative functions. Thus, the CE cost function is deployed in the optimised DNN.

3.4.6 Optimisation Algorithm Evaluation

First-order adaptive gradient descent-based optimisation algorithms have extended traditional stochastic gradient descent (SGD) techniques used to optimise DNN models by perturbing learnable parameters in more effectively reduces cost function errors for datasets with a high-dimensional feature space. This section involves an ablation evaluation of the baseline DNNs application of Adaptive Moment Estimation (Adam) (Kingma, 2015) against vanilla SGD, Momentum SGD (Rumelhart, 1986), Nesterov Accelerated Gradient (NAG) (Nesterov, 1983; Sutskever, 2013), AdaGrad (Duchi, 2011), AdaDelta (Zeiler, 2012), AdaMax (Kingma, 2015) and Nesterov Adam (NAdam) (Dozat, 2016) optimisation algorithms. Each of the evaluated optimisation algorithms weight update formulas, parameter specifications, and definable hyper-parameters, are summarised in Table 3.4.9.

Results from the comparative evaluation of the baseline Adam optimised DNNs against the SGD and adaptive algorithms are summarised in Table 3.4.10, with generalisation curves for the training accuracy and generalisation error of the algorithms depicted in Figure 3.4.7, respectively. A total of 45 different permutations of the optimisation algorithm were tested, however, for performance comparison purposes only one configuration for each algorithm is presented in the table below. Additional analysis on hyper-parameters is presented later in the section.

Optimisation Fun.	Weight Update Formula	Parameter Formulas	Hyper-Parameter
SGD	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - (\alpha * \nabla_w \mathcal{L})$	NA	α
Momentum	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m + \mathbf{V}^{m+1}$	$\mathbf{V}^{m+1} \leftarrow \gamma \mathbf{V}^m - \left(\alpha * \nabla_w \mathcal{L}(\mathbf{w}^m)\right)$	α, γ
Nesterov Mom.	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m + \mathbf{V}^{m+1}$	$\mathbf{V}^{m+1} \leftarrow \gamma \mathbf{V}^m - \left(\alpha * \nabla_w \mathcal{L}(\mathbf{w}^m + \gamma \mathbf{V}^m)\right)$	α, γ
AdaGrad	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - \alpha \cdot diag(\mathbf{G}^m)^{\frac{1}{2}} \nabla_w \mathcal{L}(\mathbf{w}^m)$	$\mathbf{G}^m = \sum_{j=1}^{m} \left(g_i^j\right)^2 \quad g_i^j = \frac{\delta \mathcal{L}}{\delta w_i^j}$	α
AdaDelta	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - \left(\frac{\alpha}{\sqrt{(\mathbf{V}^m)^2 + \epsilon}} \cdot \mathbf{g}^m\right)$	$\mathbf{V}^m \leftarrow \gamma(\mathbf{V}^{m-1})^2 + (1-\gamma)(\mathbf{g}^m)^2$ $\mathbf{g}^m = \nabla_w \mathcal{L}(\mathbf{w}^m)$	α, γ
Adam	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - \left(\frac{\alpha}{\sqrt{\hat{\mathbf{V}}^m + \epsilon}} \cdot \hat{\mathbf{M}}^m\right)$	$\hat{\mathbf{M}}^m \leftarrow \frac{\beta_1 \mathbf{M}^{m-1} + (1-\beta_1)\mathbf{g}^m}{1-(\beta_1)^m}$ $\hat{\mathbf{V}}^m \leftarrow \frac{\beta_2 \mathbf{V}^{m-1} + (1-\beta_2)(\mathbf{g}^m)^2}{1-(\beta_2)^m}$	$\alpha, \beta_1, \beta_2, d$
AdaMax	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - \left(\frac{\alpha}{1-(\beta_1)^m} \cdot \frac{\mathbf{M}^m}{\mathbf{U}^m}\right)$	$\mathbf{M}^m \leftarrow \beta_1 \mathbf{M}^{m-1} + (1-\beta_1)\mathbf{g}^m$ $\mathbf{U}^m \leftarrow \max(\beta_2 \mathbf{U}^{m-1}, \lvert\mathbf{g}^m\rvert)$	$\alpha, \beta_1, \beta_2, d$
Nesterov Adam	$\mathbf{w}^{m+1} \leftarrow \mathbf{w}^m - \left(\frac{\alpha}{\sqrt{\hat{\mathbf{V}}^m + \epsilon}} \cdot \bar{\mathbf{M}}^m\right)$	$\bar{\mathbf{M}}^m \leftarrow \beta_1 \frac{\beta_1 \mathbf{M}^{m-1} + (1-\beta_1)\mathbf{g}^m}{1-(\beta_1)^m} + \frac{(1-\beta_1)\mathbf{g}^m}{1-(\beta_1)^m}$ $\hat{\mathbf{V}}^m \leftarrow \frac{\beta_2 \mathbf{V}^{m-1} + (1-\beta_2)(\mathbf{g}^m)^2}{1-(\beta_2)^m}$	$\alpha, \beta_1, \beta_2, d$

Table 3.4.9 – Optimisation algorithm summary including weight update formula, parameters and hyper-parameters. See Section 3.1.8.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Optimisation	Hyper-Parameter	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
SGD	$\alpha = 0.1$	93.02%	0.199	0.034	3.84%	0.147	0.028	735	8.3	88
Momentum	$\alpha = 0.1, \gamma = 0.8$	94.86%	0.147	0.028	2.92%	0.112	0.023	608	8.4	73
Nesterov Mom.	$\alpha = 0.1, \gamma = 0.8$	94.96%	0.145	0.029	2.95%	0.113	0.023	607	8.4	72
AdaGrad	$\alpha = 0.5$	96.02%	0.116	0.026	2.69%	0.102	0.020	500	8.5	59
AdaDelta	$\alpha = 2, \gamma = 0.99$	95.46%	0.139	0.024	2.99%	0.113	0.022	669	9.1	73
Adam	$\alpha, \beta_1, \beta_2 = 0.1, 0.9, 0.95$	96.82%	0.096	0.021	2.58%	0.109	0.022	582	9.0	64
AdaMax	$\alpha, \beta_1, \beta_2 = 0.1, 0.9, 0.95$	96.75%	0.099	0.023	2.66%	0.110	0.022	542	8.8	62
NAdam	$\alpha, \beta_1, \beta_2 = 0.1, 0.9, 0.95$	96.94%	0.092	0.020	2.54%	0.109	0.022	576	9.2	63

Table 3.4.10 – Tabular optimisation algorithm ablation evaluation of baseline DNN trained with Adam against seven comparative algorithms. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

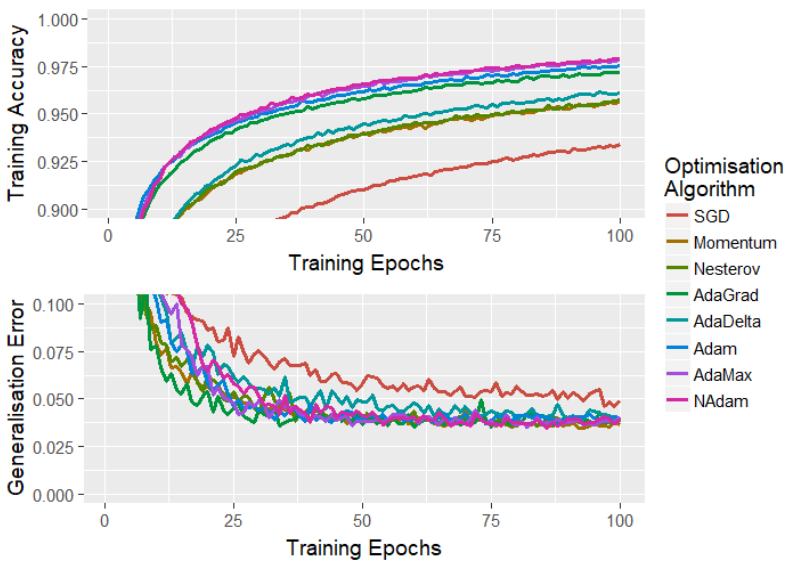

Figure 3.4.7 – Graphical comparison of training accuracy and generalisation error for DNNs trained with various optimisation algorithms. Note that the hyper-parameters selected for each optimisation algorithm are defined in Table 3.4.10.

Evidently, NAdam optimisation performs most effectively at modelling the relationship between AT firms' actions and LOB dynamics. Models trained on the evaluation dataset of 59 brokers using NAdam attain an average generalisation error of 2.54%, training accuracy of 96.94% and convergence speed of 576 seconds, performing marginally better than both SGD and the comparative adaptive algorithms along the dimensions of accuracy and generalisation. This motivates the decision to utilise NAdam in the optimised model with optimised hyper-parameters, determined in later sections, to improve training accuracy and predictive performance.

Vanilla SGD is evaluated with learning rates, $\alpha \in 0.01, 0.1$, attaining the highest average training accuracy of 93.02%, training loss of 0.199 and convergence time of 735 seconds when trained using a learning rate of 0.1. When the learning rate is set lower to 0.01, training converges with an average accuracy of only 84.75% and significantly higher generalisation error of 10.54%. These results align with the conventional heuristic that rates should be set high enough to ensure the algorithm augments weight updates towards an optimal region on the cost functions error surface, but not too high that the algorithm oscillates around an optimal basin without converging to the local minima. Among all optimisation algorithms tested the low learning rate SGD performs the poorest. Both DNN models configured with vanilla SGD take over 88 epochs on average to converge, compared to the baseline model trained with Adam that trains over 64 epochs on average. The slow learning process of SGD is expected due to the inherent architectural prerequisite that the fixed non-adaptive learning rate be set low to allow it to search over the error surface before converging slowly and inefficiently towards a local minimum (Bengio, 2012).

Momentum SGD improves the stability and convergence of neural networks over vanilla SGD (Rumelhart, 1986) by introducing a velocity vector tailored for negotiating complex error surfaces with high curvatures that are prone to wild variance oscillations during gradient descent. The multi-dimensional feature space derived from the FTSE100 LOB dataset and the high number of action classifications indicate that the cross-entropy cost function will likely have a complex error surface. Momentum SGD algorithms are tested with learning rate 0.1 and momentum rates, $\gamma \in 0.2, 0.8$, to control the acceleration rate applied to gradients based on their past direction of travel. Momentum terms take an exponential moving average to encourage more recent iteration values and punish older values. The faster momentum rate of 0.8 results in a training accuracy of 94.86% which is 1.62% points higher on average than models trained with γ of 0.2, whilst also reducing the generalisation error by 0.72% and converging 72 seconds quicker on average. Superior performance with higher momentum algorithms indicates that more information is impounded in gradients further in the past that improves the algorithms ability to more accurately perturb weights after each iteration of training. Applying Nesterov momentum with the Nesterov accelerated gradient (NAG) optimisation algorithm (Nesterov, 1983; Sutskever, 2013) has mixed results when evaluated against Momentum SGD. Training accuracy improves slightly on average, though the models' generalisation ability is also reduced by a minor amount on average across the evaluation DNNs. Both algorithms take almost an identical time to converge on average. This potentially indicates that the exponentially decaying velocity term used in Momentum SGD is not excessively updating the weight parameters after the gradient and velocity terms are calculated, such that the perturbation of the weights is not large enough to require significant augmentation or correction to be applied by the NAG algorithm.

Recently introduced adaptive optimisation algorithms, including AdaGrad (Duchi, 2011) and AdaDelta (Zeiler, 2012), have achieved significant improvements in classification accuracy over vanilla SGD and basic momentum algorithms in tasks including speech recognition (Senior, 2013), image classification (Duchi, 2011; Hadgu, 2015; Zhuangzhuang, 2016), text classification (Duchi, 2011; Zeiler, 2012; Pennington, 2014) and in deep networks with computational constraints (Dean, 2012). The results of this chapter align with the literature in that adaptive SGD algorithms largely outperform traditional vanilla SGD and momentum algorithms. AdaDelta trained with an α of 2 and γ 0.99 attains a training accuracy of 95.46% on average across the models, with the AdaGrad algorithm improving on AdaDelta, reaching a training accuracy of 96.02% and lower generalisation error of 2.69%. Both adaptive models outperform SGD and Momentum optimiser algorithms. Adaptive learning rates generally integrate a per-weight parameter annealing process into the models learning rate hyper-parameter with a functional form composed of both current and past gradient values. This contrasts with fixed learning rate schedulers that reduce the learning rate over time according to a pre-defined schedule based on the number of learning epochs completed, impacting the learning rates of all weights equally.

AdaGrad applies iterative weight updates on a dynamic per-dimension basis based on the global learning rate, α, scaled by the inverse of the squared sum of historical gradient magnitudes, G. The design of the algorithm introduces a natural annealing process as learning rates gradually decrease inversely to the total sum of past squared gradients, g. AdaGrad optimisation is highly sensitive to the learning rate hyper-parameter selection choice and initial gradients, which influences the annealing trajectory (Duchi, 2011). This issue is evident when performing an evaluation of our DNN models trained using AdaGrad optimisation with various scaling or learning rates, $\alpha \in 0.01, 0.1, 0.5$. The results of the evaluation are reported in Panel A of Table 3.4.11, showing a positive correlation between higher scaler learning rates and three measures of performance – training accuracy, generalisation error and convergence speed. A potential explanation for these results lies in the role that the scalar learning rate plays in the optimisation algorithm.

AdaDelta was designed to overcome the issue of continual learning rate decay towards zero, which is an inherent property of the AdaGrad algorithm, by restricting the window size for accumulated prior iteration gradients to a local estimate and adding a momentum term. The algorithm is theoretically capable of a more optimistic search than AdaGrad for locally convex structures at which point learning rates decay until convergence (Zeiler, 2013). However, the results presented in Panel B of Table 3.4.11, indicate that optimisation employing AdaDelta's window technique does not attain better accuracy or generalisation performance than the AdaGrad optimisation algorithms. The AdaDelta optimised DNNs evaluated take hyper-parameter values, $\alpha \in 0.5, 1.0, 2.0$, and, $\gamma \in 0.95, 0.99$. AdaDelta DNN models trained with a learning rate scalar of 2.0 and momentum rate of 0.99 perform best, with training accuracy 95.46% and generalisation error 2.99%. However, the performance of all AdaDelta models are inferior to DNNs trained with the AdaGrad algorithm that utilises a global learning rate of 0.5. A reduction in AdaDelta's momentum term does not seem to improve training performance, indicating that higher attribution should be given to past gradients implicit in the exponential moving average of past gradients, $\mathbf{V}^{m-1}$, as opposed to the current iteration gradient, $\mathbf{g}^{m}$. This result aligns with findings from the evaluation of Momentum SGD, which also performs better when higher weight is given to past gradients during the weight perturbation process. In addition, the algorithm responds well to learning rate scalars being initialised to higher values indicating that injecting faster learning through more

exaggerated weight updates in the early learning epochs induces a commensurate and more holistic search of the cost function surface. Wider search in the initial stages of learning may be conducive to performance as the algorithm moving towards a good region on the error surface with low error basins, before the accumulation of gradients anneals the learning rate, culminating in convergence at a lower error than if the learning rate was set lower and the algorithm travelled slower in the initial periods.

	Panel A - AdaGrad Learning rate α			Panel B - AdaDelta Learning rate α, Momentum γ			
Metric	**0.01**	**0.1**	**0.5**	**0.5 / 0.99**	**1.0 / 0.99**	**2.0 / 0.99**	**2.0 / 0.95**
Training Accuracy (SD)	93.43% (0.030)	95.84% (0.024)	96.02% (0.026)	93.98% (0.028)	94.92% (0.026)	95.46% (0.024)	94.82% (0.025)
Generalisation Error (SD)	4.81% (0.035)	3.18% (0.023)	2.69% (0.020)	3.95% (0.029)	3.40% (0.026)	2.99% (0.022)	3.39% (0.025)
Convergence Time (Sec)	650	502	500	809	733	669	663

Table 3.4.11 – Grid search evaluation of DNNs configured and trained with AdaGrad (Panel A) and AdaDelta (Panel B) optimisation algorithms. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Hyper-parameters are defined for each algorithm and performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

The higher performance of DNN models optimised with Adagrad vis-à-vis AdaDelta, evident in Table 3.4.11, is an interesting result given both the theoretical advantages of AdaDelta and a series of studies that have found AdaDelta to perform better for several machine learning problems (Zeiler, 2012). AdaGrad has a significant limitation in that learning rates naturally anneal towards zero as the model trains over more epochs and accumulates more gradients into the learning rate denominator. AdaDelta is expected to placate the issue of a monotonically decreasing scalar learning rate into an infinitesimally small value that terminates learning, by restricting the gradient window size. One potential reason for the improved performance of AdaGrad could be the effective selection of AdaGrad hyper-parameters, or conversely, neglecting to widen the hyper-parameter search grid for the AdaDelta algorithm, in a way that would find parameters more amenable to training the LOB dataset and improving the model's convergence properties. Another hypothesis derives from the finding that AdaGrad learning algorithms are expected to perform well and have better convergence properties in sparse environments (Dean, 2012) with flat error surfaces. Sparse representations are best learnt when learning rates are set high to allow sparse hidden neurons, those that infrequently change, to update with larger step sizes, and neuron weights that are regularly updated with higher gradients begin to update with smaller step sizes as training progresses towards a local minimum basin and learning rates anneal naturally. The sparse nature of our DNN, high-dimensional input feature space and use of Dropout regularisation, combined with the theoretical explanation above, together provides a rationale for the better performance of AdaGrad over AdaDelta.

The results from analysing the Adaptive Moment Estimation (Adam) optimisation algorithm and its extension NAdam and AdaMax indicate that NAdam trains DNNs capable of predicting trader actions given the LOB features with the highest accuracy for the training dataset and lowest generalisation error for the test dataset. This motivates the selection of NAdam in the optimised model. The next three sections evaluate the Adam, AdaMax and NAdam within a framework that studies the impact of three relevant hyper-parameters on our models' optimisation process – the learning annealing rate, the bias correction and the decay scheduler. Learning annealing rates are the initial learning rates imputed into

the model that anneal as gradient values change. Bias corrections are terms that correct for zero-initialised vectors. Schedule decay rates are an expert-defined rate that decays the initial learning rate on a per-epoch basis during the learning process. A grid search is performed over a manually defined set of possible hyper-parameters. Grid development is bounded by the mathematical constraints of the hyper-parameters and grounded in the underlying theoretical research into the relevant optimisation algorithm design. Other hyper-parameter search options, such as random search or hand-tuning of parameters, are rejected given the strong academic research already completed in the area that has tested the relevant regions of the grid space in which optimal hyper-parameters rest. Furthermore, employing Bayesian optimisation, such as tree-based models or Gaussian Process methods, are not considered given the high computational resources required in evaluating even a single DNN model, even when parallel parameter tuning procedures, similar to Bergstra (2014) are implemented.

3.4.6.1 Learning Annealing

Adam and its two extensions, AdaMax and NAdam (Kingma, 2015; Dozat, 2016), attain superior training, generalisation and convergence performance for FTSE100 CLOB DNN's trained on the 59 evaluation datasets for a series of initial learning rates, α, compared to vanilla SGD and other adaptive algorithms. Adaptive optimisation algorithms with dynamic learning rates have been shown to be most effective for training models with sparse input feature representations (Ruder, 2017), a trait of our FTSE100 CLOB dataset and generated features. Adam-type optimisation algorithms conduct natural annealing using an inversion parameter that takes the decaying squared historical gradients second moment over a given period, $\hat{V}_i^m$, which is smoothed and augmented by incorporating a momentum component, $\hat{M}_i^m$, that takes the exponentially decaying mean of past iterations first moment gradients, which are both bias-corrected given they are initialised towards zero.

In this section, a grid search evaluation of Adam-based algorithm learning rates is performed over the learning rate hyper-parameter space, $\alpha \in 0.001, 0.01, 0.1$. None of the other model parameters or configuration settings are augmented, with the bias correction terms β_1 of 0.9 and β_2 of 0.95, and schedule decay rate of zero remaining the same for all models evaluated. Results are tabulated in Table 3.4.12, with the generalisation error over the training process for the variously tuned and configured algorithms graphically presented in Figure 3.4.8. Evidently, the NAdam algorithm with an initialised learning rate 0.1 attains superior performance over all comparative DNNs, attaining a 96.94% training accuracy and generalisation error of 2.54%. All three algorithms are tuned optimally with high learning rates of 0.1 driving faster learning without causing the system to diverge, with the NAdam result mirrored by both Adam and AdaMax which reach average training accuracy levels of 96.82% and 96.75%, respectively. One can see from the figure that the three DNNs trained with Adam-based algorithms that utilise an initial learning rate of 0.001 tend to converge in a suboptimal local minimum with higher average errors than DNNs trained with higher learning rates.

The superiority of Adam-type optimisation algorithms over other SGD resides in the fact it performs learning annealing naturally within the algorithm, it scales well for datasets with sparse representations, including the FTSE100 data, and the magnitude of learning parameters are invariant to diagonal rescaling of the gradients (Kingma, 2015). Evidently, the performance of the optimised model can benefit from increasing the initial learning rate of the baseline model from an alpha α of 0.001 to an α of 0.1.

Figure 3.4.8 – Average generalisation error during learning for DNNs trained with Adam-based optimisation algorithms with varied learning rates. All other parameters, configurations and hyper-parameters remain the same as the baseline, including fixed learning rates 0.1, β_1 0.9 and β_2 0.95.

	Grid Search (Learning rate a)		
Metric	**0.001**	**0.01**	**0.1**
Adam – Accuracy (SD)	95.43% (3.65%)	96.48% (2.80%)	96.82% (2.58%)
AdaMax – Accuracy (SD)	94.94% (4.00%)	96.16% (3.00%)	96.75% (2.66%)
NAdam – Accuracy (SD)	95.60% (3.77%)	96.56% (2.82%)	96.94% (2.54%)

Table 3.4.12 – Grid search evaluation of DNNs configured and trained with Adam, AdaMax and NAdam optimisation algorithms with learning rates of 0.001, 0.01, and 0.1. Results are reported for the training accuracy with the generalisation error reported in brackets. All other parameters, configurations and hyper-parameters of the DNN remain the same including the bias correction terms and schedule decay rate of zero. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Hyper-parameters are defined for each algorithm and performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

3.4.6.2 Bias Correction

Bias correction hyper-parameters, β_1 and β_2, are evaluated for the Adam based models using a matrix hyper-parameter search grid for the optimal scalar learning rate of 0.1. Bias β_1 corrects for the initialisation of the first moment of the gradient momentum vector, $\mathbf{M}$, whilst bias β_2 performs a similar operation to the second moment learning rate inversion, $\mathbf{V}$. The value of hyper-parameter bias term β_1 should be set to equilibrate the importance between past and current gradients, whilst β_2 needs to be sufficiently large to drive convergence as learning rates begin to anneal when the weights reach the bottom of the error surface basin near the local minimum as the neural network cements specific patterns and the gradients increase in sparsity (Kingma, 2015). This section evaluates DNNs trained with values of bias correction, $\beta_1 \in 0.8, 0.9, 0.99$, and, $\beta_2 \in 0.9, 0.95, 0.99$, and reports the training accuracy and generalisation error for Adam and NAdam in Table 3.4.13 and 3.4.14, respectively.

DNN models trained with Adam optimisation appear to perform best at training the model using the baseline model's momentum bias correction β_1 of 0.9, reaching average training accuracy of 96.56%, though with a slightly higher generalisation error than models trained with the higher 0.99 value. The result is similar for the β_2 correction term with the baseline model's selection of a mid-grid value of 0.95 training models with higher accuracy and with a better ability to generalise. Applying a lower decay rate to prior iteration momentum, which represents the exponentially decaying average of past gradients, improves model performance and aligns with the results seen in the analysis of the AdaDelta and AdaGrad optimisation algorithms. This implies that past weights hold useful information that can be incorporated into the algorithms current learning process.

The evaluation of DNN models trained with NAdam and different bias correction hyper-parameter values further confirms this thinking. Higher values of β_1, which maintain the past gradient momentum vector, improve model performance, with a β_1 of 0.99 resulting in 96.8% training accuracy and a 0.07% average reduction in generalisation error over models trained with bias of 0.9. This bias term also plays an important role in the Nesterov component of NAdam by augmenting the look-ahead momentum term, $\bar{\mathbf{M}}^m$, thus, the success of models trained with higher bias terms may indicate that the Nesterov process is working, with weights updated successfully into a good region on the error surface by the momentum term before being corrected in the direction of the gradient. Conversely, the NAdam optimised models perform better with lower β_2 bias correction terms, reaching the highest training accuracy of 96.55% and lower error of 2.67% when β_2 is set to the lowest value of 0.9 on the hyper-parameter grid. This indicates that more weight should be given to the squared weight gradients computed in the current iteration, rather than past exponentially moving average squared gradients. Second moment bias correction values of β_2 that approach one are necessary for training DNNs robust to sparse gradients (Kingma, 2015), allowing the model to code unique and specialised patterns learnt from the data when tuning the model weights. Given the sparse representations inherent in the FTSE100 CLOB dataset, setting higher second-order bias correction is necessary to learn important features in the dataset, and to anneal learning when reaching an optimal error surface basin.

These results indicate that the optimal DNN model should employ the NAdam optimisation algorithm and perturb the bias corrections terms to equal β_1 of 0.99 and β_2 of 0.9.

	Adam Bias Terms Grid Search		
β_1/β_2	0.9	0.95	0.99
0.8	X	96.41% (2.77%)	X
0.9	96.44% (2.73%)	96.56% (2.67%)	96.23% (2.91%)
0.99	X	96.25% (2.54%)	X

Table 3.4.13 – Grid search evaluation of the bias term hyper-parameters, β_1, and, β_2, for DNNs trained with Adam adaptive optimisation. Hyper-parameter β_1 is defined along the vertical axis and β_2 is defined along the horizontal axis. Results are reported for the training accuracy with the generalisation error reported in brackets. All other parameters, configurations and hyper-parameters of the DNN remain the same including the learning rate terms and schedule decay rate of zero. Analysis is performed across the 59 DNN evaluation ISIN-Date-Broker dataset tuples.

	NAdam Bias Terms Grid Search		
β_1/β_2	0.9	0.95	0.99
0.8	X	96.41% (2.83%)	X
0.9	96.55% (2.67%)	96.48% (2.80%)	96.35% (2.97%)
0.99	X	96.80% (2.73%)	X

Table 3.4.14 – Grid search evaluation of the bias term hyper-parameters, β_1, and, β_2, for DNNs trained with Nesterov Adam adaptive optimisation. Hyper-parameter β_1 is defined along the vertical axis and β_2 is defined along the horizontal axis. Results are reported for the training accuracy with the generalisation error reported in brackets. All other parameters, configurations and hyper-parameters of the DNN remain the same including the learning rate terms and schedule decay rate of zero. Analysis is performed across the 59 DNN evaluation ISIN-Date-Broker dataset tuples.

3.4.6.3 Learning Rate Decay

A final evaluation regarding the use of a learning rate decay component in the Adam-type optimisation algorithms is performed. Baseline Adam and NAdam DNN models with fixed learning rates 0.1, β_1 0.9 and β_2 0.95, are compared against models trained with the same hyper-parameters but with a schedule decay component added to the learning rate. Decay rates, d, decrease the initialised learning rate α by a scaling factor after learning is completed in each epoch. Models built with decay rates, $d \in 0.01, 0.001$, are evaluated with performance results presented in Table 3.4.15, with the cross-entropy test dataset loss results per epoch depicted in Figure 3.4.9 over the 100 epochs trained.

Adding a schedule delay component negatively impacts the training accuracy, generalisation ability and convergence properties of DNNs models trained with both Adam and NAdam. The magnitude of the impact seems to relate to the size of the schedule decay, with higher decay rates more negatively impacting model performance. Furthermore, from the figure it is evident that the schedule decay is causing the DNN to overfit after approximately 30-40 epochs, with the test loss beginning to increase for all the functions except Adam with decay 0.01.

From Panel A in Table 3.4.15, DNN's trained using the Adam optimisation algorithm and a schedule decay of 0.01 attain 2.8% lower accuracy on average over the baseline model trained with no decay component. Generalisation error also increases by 1.41% and the models take on average 11% longer to converge than the baseline model. From Panel B, the results are not as pronounced for DNNs trained with NAdam, however, the training loss for models trained with 0.01 decay see an average 15% increase in training loss over the DNN models evaluated. Models with decay components also generalise worse to unseen data, though they do converge at a faster rate on average. Adam optimised DNNs tend to be negatively affected by a larger magnitude than NAdam DNNs, with one possible explanation being that the Nesterov component of NAdam to a degree corrects the increasingly weaker weight updates as the training iterations grow and the decay on the learning rate is more pronounced.

The hyper-parameter evaluation performed in this section indicates that the optimised DNN will not benefit from including a schedule decay component in the learning optimisation process, rather the initial learning rate should be allowed to anneal naturally as a function of the size of weight gradients.

	Panel A - Adam Schedule Decay Rate d			Panel B – NAdam Schedule Decay Rate d		
Metric	0	0.001	0.01	0	0.001	0.01
Training Accuracy (Training Loss)	96.82% (0.096)	96.25% (0.115)	94.02% (0.220)	96.94% (0.092)	96.33% (0.108)	96.31% (0.109)
Generalisation Error (Test Loss)	2.58% (0.109)	3.04% (0.115)	3.99% (0.177)	2.60% (0.109)	2.79% (0.108)	2.83% (0.109)
Convergence Time (Sec)	582	495	612	576	552	540

Table 3.4.15 – Grid search evaluation of schedule decay rate for DNNs configured and trained with Adam (Panel A) and NAdam (Panel B) optimisation algorithms. All other parameters, configurations and hyper-parameters remain the same as the baseline, including fixed learning rates 0.1, β_1 0.9 and β_2 0.95. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Hyper-parameters are defined for each algorithm and performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

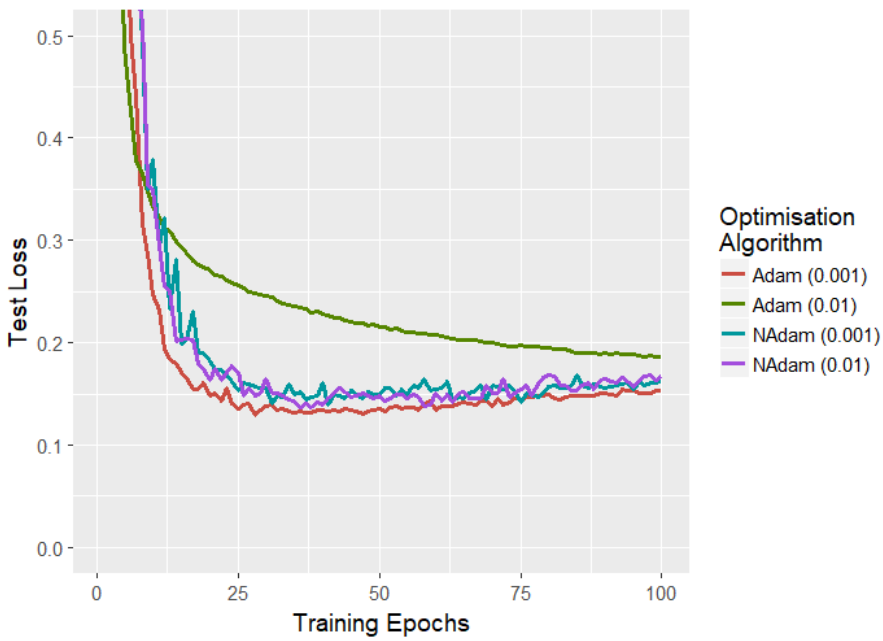

Figure 3.4.9 – Average test loss during learning for DNNs trained with Adam and NAdam optimisation algorithms with varied schedule decay rates. All other parameters, configurations and hyper-parameters remain the same as the baseline, including fixed learning rates 0.1, β_1 0.9 and β_2 0.95.

3.4.7 Network Depth & Dimensionality

The ability of feedforward fully-connected DNNs to improve capacity by learning more abstract hierarchical representations from deeper networks with wider dimensions, architecturally defined as networks with more layers and neurons within those layers, is examined by comparing the baseline DNN against models of various structural configurations. Optimisation algorithms presented in deep learning academia are capable of learning increasingly complex non-linear functions from richer representations (Bengio, 2013) and larger hypotheses space, facilitated by parallel computing, and advancements in computational graph programming languages. Modern networks can be trained and optimised with a very high number of parameters as networks become deeper and wider, though with the trade-off of higher computational cost, proneness to overfitting, and risk of poor generalisation to unseen data. Theoretically, a sufficiently wide single hidden layer network is capable of approximating any continuous bounded non-linear function according to the universal approximation theory (UAT) (Cybenko, 1989;

Hornik, 1991). However, networks width is required to be exponential in their input dimensions to meet this property. Building an approximator that meets all the properties of the UAT is impractical in real-world applications and rarely possible in experimental domains. The high-dimensional nature of our FTSE100 CLOB dataset further makes the task of fitting a single hidden layer network capable of performing accurate predictions given that the number of neurons required in that hidden layer may be extremely large. Thus, developing deep network with less dimensional width is both practical, computationally possible, and may result in more effective models that can generalise outside the training data. This section presents an evaluation of DNNs' capability of learning more complex functions and improve predictive performance as architectures become deeper and wider.

The impact of DNN depth is tested on the evaluation dataset of 59 ISIN-Date-Broker models designed with one, two, three and four hidden layers of 500 neurons each, using the specifications of the baseline model except for the augmentation in the layer component. Performance results for training accuracy, generalisation and convergence are reported in Table 3.4.16, with training and generalisation curves graphed in Figure 3.4.10. As one would expect, deeper architectures of the baseline DNN model with more hidden layers provides increased expressive power, attaining higher levels of training accuracy and lower generalisation error when trained on the datasets during evaluation. The four-layer network exhibits good convergence properties, with an average training accuracy of 95.85%, a categorical cross-entropy training loss of 0.137, and generalisation error of 3.6%. These results are a 2.09% and 8.69% improvement on the single layer network for training accuracy and generalisation error, respectively. The depth efficiency of DNNs is referred to as an expressive power given that a reduction in layers results in an exponential decrease in the parameters and capacity of the network. Thus, shallower networks with fewer parameters seem to do well at memorising relationships in the data, attaining a high accuracy of 93.76%, but are not able to develop a complex representation space that allows an algorithm to learn general patterns and relationships applicable for making predictions in the real-world domain. These networks cannot compete with the expansive generalisation abilities and predictive power of deeper networks that can learn more abstract features, as evident in the results presented. The specific nature of our dataset may also hinder shallow networks ability to learn important relationships for generalisation given that algorithmic trading exists within a highly complex business environment with firms competing to configure algorithms in a high-dimensional realm with algorithms potentially taking in a near infinite set of inputs.

The trade-off between depth and computational cost, as explained previously, bears fruition in the results, with a strong positive correlation between the number of layers and training time until convergence. Single layer DNNs take on average 371 seconds to converge to a solution, though a sub-optimal one, whilst four-layer networks take 238% longer to attain the same objective with the difference that they output a more optimal mapping function solution. One can note the low training time of 4.1 seconds per epoch for single layers which is relatively quick for complex and vast datasets such as those deployed in this chapter. The performance of the baseline four-layer model provides a fair trade-off between the slight improvements in generalisation gained from deepening the network versus the increased computational cost. These results provide evidence that the representation power and training accuracy of the DNNs improve as layers are added to the network, without a subsequent increase in dimensionality, aligning with the theoretical work of Erhan (2010), Telgarsky (2016) and Eldan (2016). The results do not indicate that overfitting is a concern even in the four-layer network as the

generalisation error of 3.6% is very low and stable as epochs rise. Furthermore, one can see that an increase in training accuracy by fitting the predictive function to the training data does not induce a commensurate increase in the generalisation error as evident in Figure 3.4.10.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Depth (Layers)	Neuron Structure	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
One Layer	500	93.76%	0.207	0.028	12.29%	0.460	0.072	371	4.1	90
Two Layers	500/500	95.74%	0.139	0.022	5.95%	0.236	0.045	556	7.2	77
Three Layers	500/500/500	95.81%	0.137	0.025	4.42%	0.172	0.036	698	10.1	69
Four Layers	500/500/500/500	95.85%	0.137	0.027	3.60%	0.138	0.029	886	13.0	68

Table 3.4.16 – Tabular network depth ablation evaluation of the baseline model configured with different numbers of hidden layers. Each hidden layer is composed of 500 neurons. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

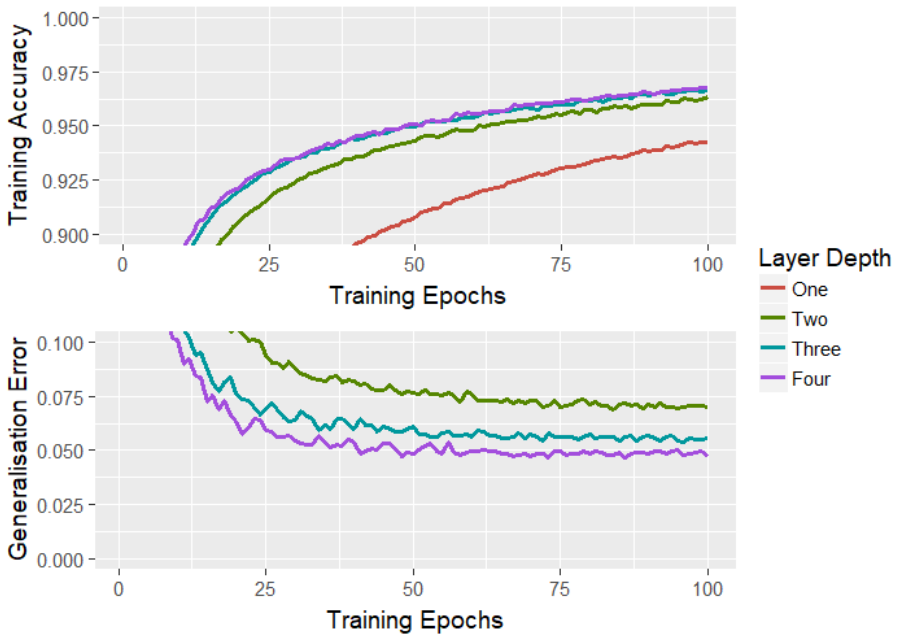

Figure 3.4.10 – Graphical comparison of training accuracy and generalisation error for DNNs configured with one, two, three and four layers composed of 500 neurons each.

Injecting additional neuronal parameters into the DNN architecture in the form of configuring hidden layers with wider dimensions and additional neurons, have been shown to achieve state-of-the-art results when applied to deep (He, 2015) and even single-layer networks (Ba, 2014). Other studies note that shallow networks are difficult to train given their inflexible nature and reduced representation space relative to deeper architectures (Bengio, 2013; Seide, 2011), with depth over width preferred in most cases under a strict parameter budget (Eigen, 2013). This section next evaluates the capacity of DNNs to model data successfully based on their level of dimensionality, that is, how many neuron parameters are fitted in each hidden layer. For this experiment the baseline DNN specifications and architectural configuration of four hidden layers is maintained, though the three comparative models are trained with varied numbers of 100, 500 and 1000 neurons in each layer. Results presented in Table 3.4.17 are in line with the empirical literature that adding width to networks enables the DNN to learn more complex functions, hence, improving the training accuracy though at the cost of more computational resources, a similar case to adding depth in the network. Four-layer baseline DNN's trained with 1000 neurons

have an average training accuracy of 96.52% at convergence, which is 4.11% higher than narrower 100 neuron networks with the same learning and architecture configurations. This demonstrates that the learning properties of the FTSE100 dataset is more amenable to wider and deeper neural networks.

An interesting result arises when evaluating the networks generalisation ability and computational training cost. Networks with 500 neurons, similar to the baseline model, generalise better to the test dataset than narrower or wider networks, attaining a generalisation error of only 3.6% on average across the evaluated DNNs. As expected, the computational overheads of significantly wider networks may be prohibitive for practical learning tasks. This becomes apparent by analysing the 1000 neuron networks performance vis-à-vis the 500 neuron DNN, with the wider network attaining only 0.75% higher training accuracy, but with a higher generalisation error and a significantly longer training time, taking an additional 961 seconds on average to reach a convergence state compared to the baseline 500 neuron model. Evidently, the longer training time and tendency of the wider network to mildly overfit by memorising patterns in the training dataset detracts from its increased expressiveness. This becomes apparent from the fact that the training accuracy of the wider network tends to increase without a corresponding increase in the test accuracy after only 68 epochs compared to the baseline models 81.

From a practical perspective, one can hypothesise that wider networks improve DNN accuracy and convergence properties, assuming a constant number of hidden layers, given that a larger quantum of weight parameters require relatively smaller weights during optimisation to approximate the same function, thus leading to a stable and consistent learning regime. It is also important to note that the models are trained using a CPU which impacts the computational training speed relative to GPU processors capable of processing numerous small kernels inherent in wide networks through parallel operations performed on large tensors. CPUs are more agreeable when performing sequential computations, layer to layer in deep networks, whilst training wider networks will generally benefit from distributed parallel computational architectures where operations are executed concurrently.

Once more the findings are in-line with theoretical research that wider networks can improve the accuracy and expressiveness power to approximate non-linear functions for rectifier networks (Lu, 2017), wide residual networks (Zagoruyko, 2017), and feedforward networks (Ba, 2014). A third potential evaluation could be performed on single layer shallow, but very wide networks, similar to that employed by Ba (2014) using a set of mimic network models, to determine whether they are capable of learning a good non-linear function that explains the input-output relationship. Ba (2014) argues that the success of *deep* architectures may be due to the match between the current learning procedures, processes, algorithms and design elements usually deployed in the literature to optimise these network, rather than the 'deep' nature of these networks. This section does not provide evidence supporting this view.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Width (Neurons)	Neuron Structure	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
100	100/100/100/100	92.41%	0.238	0.036	4.08%	0.158	0.029	351	4.5	79
500	500/500/500/500	95.77%	0.137	0.027	3.60%	0.138	0.029	886	13.0	68
1000	1000/1000/1000/1000	96.52%	0.117	0.024	3.65%	0.141	0.030	1847	28.9	64

Table 3.4.17 – Tabular network width dimensionality ablation evaluation of the baseline model configured with a different number of neurons in each hidden layer. Each DNN is configured with four hidden layers of varied neuronal structure. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

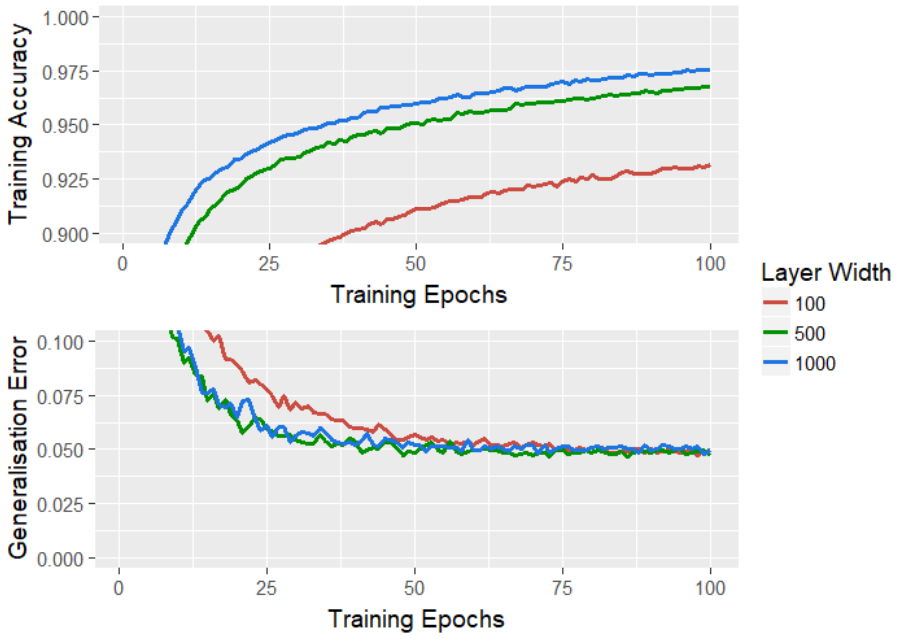

Figure 3.4.11 – Graphical comparison of training accuracy and generalisation error for DNNs configured with four layers composed of 100, 500 and 1000 neurons in each layer.

The ablation evaluation conducted in this section indicates that the four-layer baseline DNN with neuronal dimensions of 500/300/300/200 neurons provides an appropriate architecture for the FTSE100 dataset. The architecture provides sufficient capacity for the network to develop a rich representation space of learnable functions and the ability to impound abstract hierarchical relationships from the LOB features. Furthermore, the configuration of the DNN controls for a trade-off between an increased risk of overfitting and costly computational requirements when expanding the networks dimensions and depth. Hence, this architecture can be maintained for the optimised DNN.

3.4.8 Batch Size Evaluation

The batch size hyper-parameter B represents the size of mini-batch samples propagated through the network during each learning iteration to update the model's learnable parameters and is generally chosen using empirical inspection or ad hoc heuristics. An epoch is completed when all mini-batches have passed through the network once, before the process is repeated. Optimal or equilibrium mini-batch sizes generally differ depending on the machine learning problem, input feature dataset structure or learning objective, which motivates this evaluation of the performance of the baseline DNN trained with different batch sizes. An ablation evaluation is performed along the dimensions of accuracy, generalisation and computational speed by comparing the baseline model, which trains with a batch size 128, against DNNs configured with augmented batch size parameters, $B \in 16, 32, 64, 128, 256, 512, 1024$. These hyper-parameter values represent common batch sizes used in deep learning literature (Bengio, 2012; Krizhevsky, 2012; He, 2015) and follows the heuristic of using exponents n of base 2, that is, 2^n. The evaluation utilises mini-batch methods of gradient descent optimisation, rather than purely single-sample stochastic or batch gradient descent methods, given their computational simplicity, stable convergence properties and regularising effective on the network.

Evaluation results reported in Table 3.4.18 show that the baseline DNN model with 128 mini-batch size converges over 661 seconds on average with a generalisation error of 3.74% and standard deviation of 0.031, the lowest error of all seven models analysed. Results regarding model accuracy and performance are in line with expectations, with very small batch sizes attaining lower levels of training accuracy. For example, DNN models trained with mini-batch sizes of 16 samples attained an average training accuracy of 87.8%, which is 8.93% below models trained with 512 samples, the most accurate models applied to the training dataset. However, DNN's optimised with large mini-batches that perform accurately at predicting classifiers in the training dataset tend to overfit, losing their ability to generalise to unseen data samples (LeCun, 1998). This phenomenon arises due to the fact that DNNs with large mini-batches generally converge to sharp minimisers, with larger positive eigenvalues of the training function, focussing on regions on the error surface closest to the initialised weights. In contrast, smaller mini-batches converge to flatter minimisers, with small positive eigenvalues, given the noise inherent in gradients estimated with smaller batch sizes (Keskar, 2017). Noise in these smaller samples acts as a quasi-regulariser and also enables a wider and potentially gainful explorative search over the error surface given the induced oscillations in the optimiser algorithm from the stochasticity inherent in smaller samples. Large batch DNNs find it more difficult to escape saddle points and sharp minimisers, whilst DNNs trained with smaller batch sizes are capable of iterating the function out of a sub-optimal sharp basin given their relatively noisier gradients.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Batch Size	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
16	59	87.80%	0.343	0.053	6.34%	0.239	0.044	2400	31.6	76
32	59	91.83%	0.236	0.043	4.79%	0.179	0.038	1394	19.4	72
64	59	93.92%	0.182	0.036	4.12%	0.153	0.034	856	12.8	67
128	59	95.54%	0.143	0.025	3.74%	0.144	0.031	661	9.1	72
256	59	96.52%	0.144	0.018	3.76%	0.162	0.034	506	7.0	72
512	59	96.73%	0.195	0.016	4.02%	0.203	0.035	420	5.5	76
1024	59	96.35%	0.319	0.020	5.30%	0.307	0.055	397	4.8	83

Table 3.4.18 – Tabular batch size ablation evaluation of the baseline model configured a batch size B hyper-parameter of 128 compared against DNNs trained with different batch sizes, with other configurations and parameters remaining the same as the baseline model. Analysis is performed across the 59 DNN evaluation Broker dataset tuples. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed.

The results in Table 3.4.18 are bolstered by illustrating the U-shaped relationship between batch size and the DNNs average generalisation error for converged models presented in Figure 3.4.12. This figure confirms the U-shaped phenomenon, with DNNs trained on mini-batch sizes of 16 converging with an average generalisation error of 6.34% which reduces as the batch size increases to 128 samples, attaining an average error of 3.74%, before rising as the batch size increases, terminating with DNNs trained with 1024 samples converging with an average error of 5.3%. Applying dynamic sampling techniques that initialise with a lower batch sizes before growing gradually in line with the training epochs (Byrd, 2012; Friedlander, 2012) or utilising adaptive batch sizes such as AdaBatch (Devarakonda, 2017) does not seem necessary given the accuracy and stability of the baseline DNN model with mini-batch size 128.

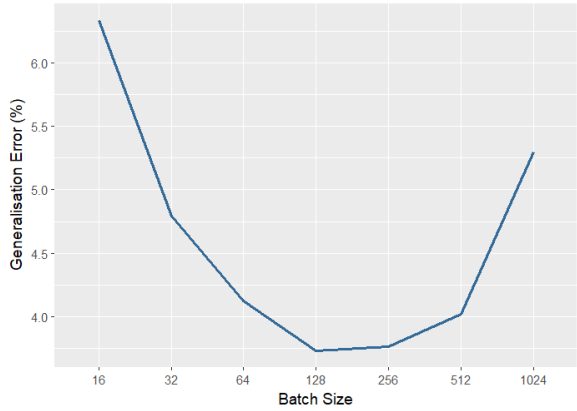

Figure 3.4.12 – Representation of average generalisation error (%) for the 59 DNN evaluation Broker dataset tuples trained with the baseline model configured with different batch sizes.

The mini-batch size employed by the DNN is negatively correlated with both the training time per epoch and the average speed of convergence. These results appear logical given that smaller batch sizes undergo more iterations per epoch, with models using 16 samples per mini-batch taking 6.6 times longer to complete one learning iteration than models trained with 1024 samples. The impact of batch size on model convergence speed follows a similar trajectory with higher batch size models converging faster than lower batch size models. This property of DNNs is captured in the test loss curves presented in Figure 3.4.13. An analysis of the cost function loss for the test dataset during learning for DNNs trained with different batch sizes shows that large 1024 batch size networks take 55 epochs on average to converge to an average cross-entropy test loss under 0.5. In contrast, networks trained with smaller batch sizes of 16 and 32 samples can attain this level after 5 epochs on average before the performance of these networks begins to sharply plateau. Gradient descent-based learning is a sequential iterative process, though non-sequential techniques such as parameter averaging or natural gradient SGD (Povey, 2015) have been deployed to allow for multi-core GPU parallelism to improve speed of learning for small batch size models. However, the accuracy of the baseline model negates the need for its implementation.

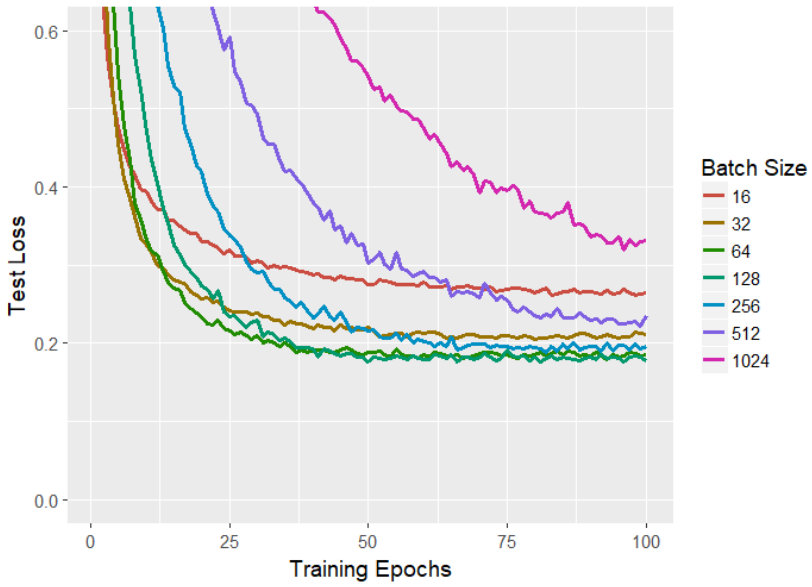

Figure 3.4.13 – Average test loss during learning for baseline DNNs trained with differently configured batch sizes.

In conclusion, the current architecture of the baseline model which utilise mini-batches of 128 samples does not require any configuration changes to the batch size parameter for the optimised DNN.

3.4.9 Regularisation Evaluation

Regularisation techniques including dropout with various dropout rates (Hinton, 2012b; Srivastava, 2014), LASSO L1 penalisation (Tibshirani, 1996), and L2 weight decay or Tikhonov regularisation (Tikhonov, 1943) are evaluated against the baseline DNN that implements both early stopping and dropout with rates of 0.5, 0.4 0.4, and 0.3 per hidden layer. Regularisation is deployed in the baseline DNN to prevent neurons developing complex neuronal co-adaptations between abstract feature representations learnt during network training that leads to overfitting of the input-output function to the training set and provides poor generalisation for a novel test dataset (Srivastava, 2014). The tendency of a DNN to overfit to the training set is generally correlated with the level of complexity of the non-linear mapping function and the number of learnable parameters in the model. Penalties applied to the cost function, in the form of L1 and L2 regularisation, and implicit weight penalties, due to the fitting of dropout layers, aim to prevent weights becoming excessive and the complexity of the function becoming over cumbersome, reducing the risk of overfitting whilst ensuring the simpler functions do not introduce too much bias into predictions.

3.4.9.1 Dropout Regularised Networks

This section evaluates DNN's ability to break these co-adaption patterns and reduce the risk of overfitting by fitting a random dropout mask, with probability p, to the hidden layers of the network. The baseline DNN model is built with dropout layers in each hidden layer each use dropout rates between 0.3 and 0.5. This model is compared against a DNN trained with L2 and dropout and a DNN trained with no dropout layers. The utility of including dropout layers in the DNN model is assessed by first confirming the severe overfitting affecting networks with no dropout, despite these flaws an analysis of each networks generalisation capability is then completed.

Results in Figure 3.4.14 demonstrate that DNN models without Dropout layers trained on the FTSE100 CLOB dataset completely fail to explain the underlying mapping function of the data due to severe overfitting that occurs when the model begins training. The figure compares the training accuracy and validation accuracy for the DNNs trained with and without dropout layers, over the 59 evaluation ISIN-Date-Broker datasets. For the purpose of illustration, early stopping is *not* used in this example. The model trained with no dropout immediately learns neurons weights for the training dataset that almost perfectly explains the inherent non-linear mapping functions for input features and output classes evident by the training accuracy rising towards a prediction accuracy close to one. However, the optimised learnt function has completely overfit to the training dataset, generalising poorly to the test dataset, graphically illustrated by the validation accuracy curve that sits significantly below the training accuracy curve. Conversely, the baseline model with dropout improves both the training accuracy and generalisation over the learning epochs, with both curves tracking each other rather closely with minimal oscillations given that dropout networks generally have smoother convergence properties (Srivastava, 2014). Overfitting of the baseline dropout network begins much later around the 65-70 epoch mark on average across the datasets.

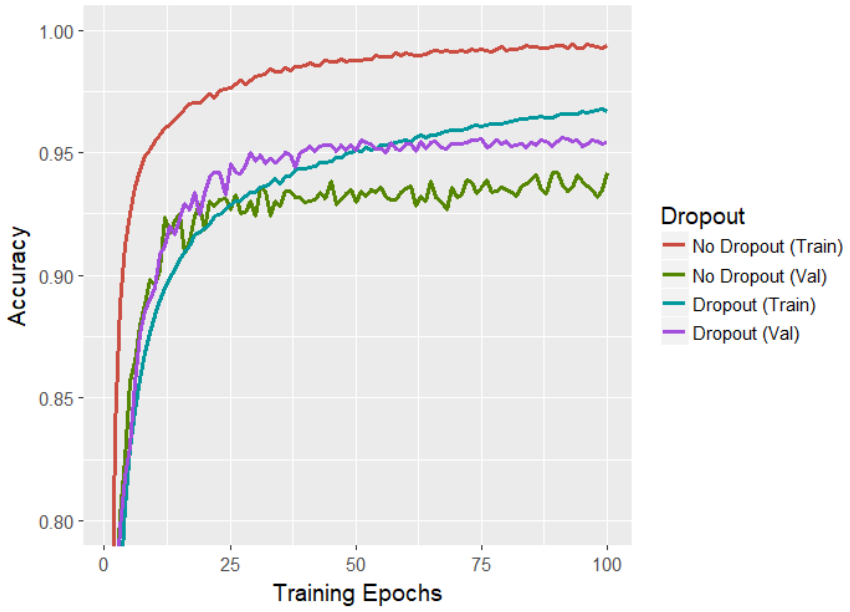

Figure 3.4.14 – Comparison of training and validation dataset accuracy for DNNs trained with (baseline) and without dropout. Results are averaged over the sample datasets and *no* early stopping techniques are applied.

Regardless of the severe overfitting from annulling dropout layers from the baseline architecture, a basic analysis of the convergence properties and generalisation ability of both models is performed with results reported in Table 3.4.19. Applying dropout during mini-batch gradient descent in the baseline DNN reduces the average generalisation error by 2.86% over models trained with no dropout layers in the network, though taking double the time to reach a convergence state. From Figure 3.4.14 one can note that the immediate overfitting of the no dropout DNNs causing the early stopping algorithm to terminate training effectively as soon as the minimum number of epochs, 39, is reached as per the stopping criterion. At this point the model has already hugely overfit the training dataset. The improvement in model performance and reduction in proneness to overfitting likely arises from dropout layers' introduction of stochasticity into the network architecture to dissolve large complex co-adaptations into smaller more generalisable co-adaptations. This is theoretically explained by viewing dropout networks as a bagged ensemble of sub-models which introduce a masking vector that makes the inclusion of any neuron during a training iteration unreliable. By thinning the networks with dropout, forward propagation of the input feature vector and backpropagation of gradients to update non-dropped neuron weight connections is performed quicker. Hence, the lower number of epochs for training the non-dropout network, 39, versus the dropout network, 72. Two arguments against dropout are the (1) reduction in network capacity from dropped neurons to reduce complex co-adaptions and (2) the additional time required to train the model given that each training iteration involves the dropping of neurons and consequential development of a new random architecture with noisy stochastic and unique gradients. However, the benefits of reduced overfitting to the training set and stable convergence properties when implementing dropout in DNNs in this case of the FTSE100 dataset outweigh the need to reduce training time to convergence and concerns regarding reduced model capacity.

DNN Models	Test Set			Epochs
Regularisation	Error	Loss	SD	Mean
Dropout	3.74%	0.144	0.031	72
No Dropout	6.60%	0.261	0.051	39

Table 3.4.19 – Comparison of generalisation capability of DNNs trained with and without dropout layers.

Evaluation of the hidden layer dropout rate hyper-parameter, p, requires a comparison of DNNs tuned with different rates of randomly dropping units, $p \in 0.2, 0.4, 0.6, 0.8, 0.95$. Table 3.4.20 reports the empirical results for the training loss, test loss and training time per epoch for DNNs trained with various dropout rates, averaged over the evaluation datasets. The results indicate that lower dropout rates lead to the model converging quicker with fewer epochs, though with severe overfitting issues at earlier epochs as more neurons are maintained in the model and more complex co-adaptations to explain the input are learnt. This issue is illustrated in Figure 3.4.15 which graphs the training and test loss as the dropout rate increase. The U-shape of the test loss is evident as models trained with p of 0.2 have a test loss of 0.13, decreasing to 0.1 for networks with dropout 0.6, before increasing to 0.498 for networks trained with a dropout of 0.95. The common issue for dropout networks with low masking probability is that overfitting occurs relatively quickly, whilst networks trained with higher rates will generally underfit that data as patterns cannot be developed through neurons that are constantly extricated from the manifold. Srivastava (2014) finds a region between dropout rates of 0.4 to 0.8 where the models convergence and fitting properties are sound. The results in this section indicate that the region 0.3 to 0.5 may be more agreeable for the FTSE100 dataset. DNN models with dropout of 0.4 and 0.6 attain the lowest test dataset cost function loss of 0.103 and 0.1, which also have low dispersion in their results computing standard deviations of 0.007 and 0.008, respectively. The findings in Table 3.4.20 and Figure 3.4.15 confirms the choice of selecting dropout rates between 0.3 and 0.5 for hidden layers of the baseline model architecture as correct for the tuple DNNs trained on the FTSE100 dataset.

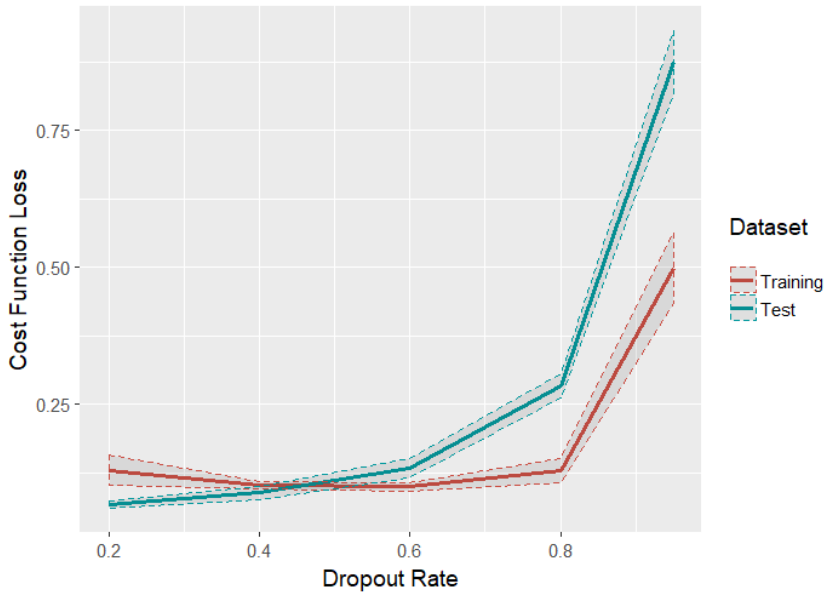

Figure 3.4.15 – Comparison of training and test cost function loss for DNNs trained across the evaluation datasets with various dropout rates. Confidence bands are defined at the 90% level.

DNN Models	Dropout Rate (p)				
Metric	0.2	0.4	0.6	0.8	0.95
Training Loss (SD)	0.067 (0.003)	0.089 (0.006)	0.133 (0.009)	0.284 (0.011)	0.875 (0.029)
Test Loss (SD)	0.130 (0.014)	0.103 (0.007)	0.100 (0.008)	0.129 (0.011)	0.498 (0.032)
Time per Epoch (Sec)	15.8	9.5	7.6	7.5	7.5

Table 3.4.20 – Grid search evaluation of dropout rate for DNNs. All other parameters, configurations and hyper-parameters remain the same as the baseline. Analysis is performed across the 59 DNN evaluation Broker dataset tuples.

3.4.9.2 L2 & L1 Norm Regularisation

Traditional regularisation techniques including LASSO L1 penalisation (Tibshirani, 1996), and L2 weight decay or Tikhonov regularisation (Tikhonov, 1943) are evaluated against the baseline DNNs that implement both early stopping and dropout. From the results in Table 3.4.21, it is evident that Dropout significantly improves training performance and reduces generalisation error relative to both models trained with and without dropout that apply L1, L2 or both regularisation methods.

In practice, L1 regularisation is commonly used for DNNs with sparse feature spaces to recover sparse signals in the network by concentrating weight regularisation during learning in a small number of relatively large weights whilst leaving a large number of small weights unregularised. The results provide evidence that DNNs with L1 regularisation attains poor average test accuracy of 77.72% with high dispersion, potentially indicating that punishing weights with L1 norms is not conducive to the multi-dimensional and well-defined feature space of the FTSE100 dataset. Furthermore, the fact that L1 regularisation provides instances where the function is not differentiable may lead to divergence in learning optimal weights during gradient descent which requires stable and differentiable functions. Thus, L1 regularisation has no beneficial regularising effect for the DNNs trained in this chapter further evident by the high 22.77% generalisation error when L1 is combined with dropout and 25.77% when additionally combined with L2 regularisation.

L2 weight decay regularisation performs relatively well for DNNs with predictor classes that are highly correlated in the input features. L2 allows a range of features to take non-zero weight values during learning, leading to less sparsity in the input feature vector. The results indicate that applying a L2 regulariser with dropout improves on the results of L1 regularisation, though still attains a 3.46% lower training accuracy rate than DNN models trained solely with dropout. Thus, adding a L2 weight decay kernel regulariser term to the baseline model, which is configured with dropout but without L2, negatively impacts performance over the baseline model. From Table 3.4.21 one can note that the baseline model attains a generalisation error of 3.73% on average, whilst baseline DNNs with dropout that add an L2 penalty only reach average generalisation error of 11.11%, which contradicts the findings of Srivastava (2014) that dropout and L2 work well together. One possible explanation is that the authors propose applying max-norm regularisation which bounds the L2 weight constraint with constant c during learning, whilst the L2 norm proposed in this experiment is unconstrained. The question arises as to why dropout performs better than L2 regularisation. In the context of the FTSE100 dataset, the answer potentially rests in the fact that L2 regularisation scales better for models trained on relatively small datasets (Ng, 2004). Given that the datasets for some brokers are quite large with a high number of samples to train on, L2 does not perform relatively well as it may with a smaller dataset. Furthermore, whilst L2 applies a penalty to weights in the cost function, dropout performs regularisation implicitly by identifying and reducing the weights of redundant features through the masking or dropping of random neurons during training to prohibit the network learning spurious patterns.

The results presented in this section confirm the expectation that dropout works as an effective regulariser for complex models such as those trained in this chapter on the FTSE CLOB datasets, a result common in the literature (Srivastava, 2014; Phaisangittisagul, 2016). Dropouts ability to regularise deep and wide networks by implicitly penalising weights and breaking co-adaptions between neurons seems to be the most effective method of regularisation evaluated. There also appears to be minimal

evidence that adding additional regulariser terms, such as L1 or L2, will improve training accuracy or the models' generalisation ability. The marginally higher computational resources required to train dropout DNNs are not considered relevant in this study. In conclusion, the optimised model will maintain the current dropout regulariser without adding any additional weight penalising terms.

DNN Models		Training Set			Test Set			Training Time (Sec)		Epochs
Regularisation	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
Dropout	59	95.54%	0.143	0.025	3.74%	0.144	0.031	661	9.1	72
L1 + Dropout	59	28.74%	2.433	0.141	69.48%	2.412	0.190	572	6.1	94
L2 + Dropout	59	89.23%	0.647	0.046	11.11%	0.864	0.124	376	6.0	63
L1.L2 + Dropout	59	28.20%	2.454	0.135	70.63%	2.437	0.177	613	6.4	96
L1 + No Dropout	59	77.72%	1.381	0.177	22.77%	1.519	0.176	478	5.4	88
L2 + No Dropout	59	91.98%	0.479	0.064	9.04%	0.711	0.133	410	5.3	77
L1.L2 + No Dropout	59	75.23%	1.543	0.187	25.77%	1.688	0.187	478	5.7	83

Table 3.4.21 – Ablation evaluation of regularisation strategies is performed by comparing DNNs with additional regularisation terms, trained on the 59 evaluation datasets, to the baseline model that utilises dropout.

3.4.10 Batch Normalisation Evaluation

Batch Normalisation (BN) is a modular component of the DNN architecture generally incorporated into the feedforward layers of the network to achieve stable activation value distributions throughout the hidden layers resulting in improved learning performance and more efficient optimisation (Ioffe, 2015). This section performs three experiments. The first two evaluate the utility derived from applying BN to the baseline DNN. The third experiment analyses the weight distributions of the BN layers.

The first experiment is performed by comparing models trained using BN before and after the non-linearity of the activation function is applied. BN layers are generally placed after the fully-connected layer but before inputs, $\mathbf{z}^l$, are feed into the non-linearity activation function, $\boldsymbol{\phi}$, rather than applying the normalisation to the input feature vector, $\mathbf{x}^{l-1}$, according to the seminal paper on the matter (Ioffe, 2015). The primary argument for this architecture is that doing so prevents inputs entering a saturated regime immediately by normalising input into the activation function, whereas applying BN after the non-linearity is less theoretically robust given that the activation output values have already undergone transformation. The system employed by Ioffe (2015) tends to work very well for the FTSE100 CLOB dataset. The results of the comparison presented in Table 3.4.22 indicates that models trained with BN after the activation function completely fail to converge with any degree of predictive power. These models attain a generalisation error of 27.19% against the baseline model of 3.78% which applies BN before the non-linearity. The debate surrounding the correct application of BN has not been fully settled. For example, Mishkin (2015) attain similar results using both methods when testing his FitNet4 model on the CIFAR-10 dataset. Regardless, the success from applying the BN layer before the ELU activation function for our DNN model indicates there is no need to modify the architecture of the model.

The second experiment evaluates a DNNs trained with no BN. As expected, this model performs poorly when generalising to the test set with an average generalisation error of 15.67%. Furthermore, the high standard deviation in performance of 0.189 is significantly larger than the baseline BN DNN dispersion of 0.032. From the results one can argue that BN successfully attains its objective of correcting for

internal covariate shift in the FTSE100 dataset that arises from divergent distributions of neuron input values across hidden layers during training iterations of mini-batches (Ioffe, 2015). This occurs as feature input values undergo a series of compounded operations as they forward propagate through the network during training, shifting distribution densities from layer to layer depending on the synaptic weight connections. The baseline model's four-layer architecture compounds this problem as deeper networks amplify the impact on input distributions as values travel through the network. A potential reason why the non-BN DNN fails to learn sufficiently could be that the learning rate of the Adam optimisation algorithm and negative bound alpha of the ELU activation are both set too low, which in conjunction with the dropout layers may lead to the DNN failing to maintain a quasi-normalised stable network during learning. These results may be improved if the learning rates were set higher.

Batch Normalisation	Firms	Test Error	SD	Training Time per Epoch (Sec)
BN Before NL	59	3.78%	0.032	9.2
BN After NL	59	27.19%	0.248	9.1
No BN	59	15.67%	0.189	6.0

Table 3.4.22 – Evaluation of DNN models trained on the dataset with batch normalisation layers before the non-linearity (NL), after the non-linearity, and DNNs configured with no batch normalisation.

The final experiment analyses the weights used in BN layers to apply a linear transformation that scales and shifts the normalised activation input values by learnable parameters gamma, γ, and beta, β, respectively. These learnable BN weights are optimised on a per-neuron basis, though this section studies the distributional properties of the networks by amalgamating results across the evaluation datasets and neurons in each layer of the four-layer DNN baseline model. Density plots for beta and gamma weights are presented in Figure 3.4.16.

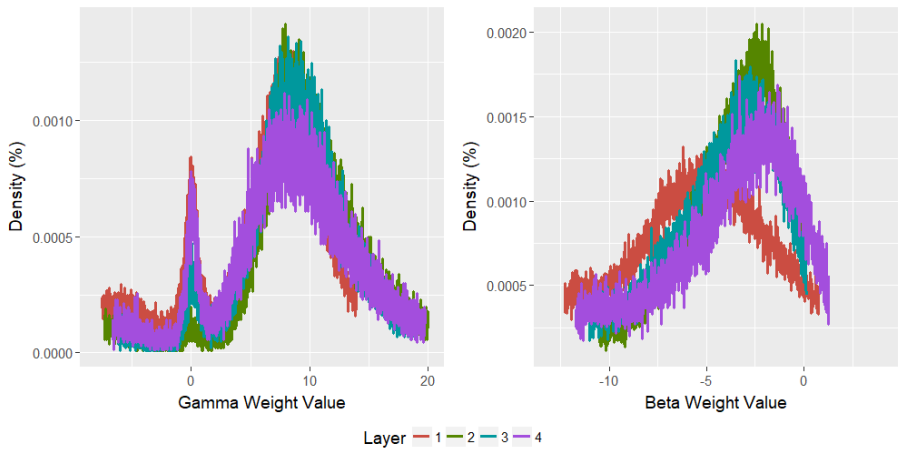

Figure 3.4.16 – Conditional density plots for Beta and Gamma weights of BN layers extracted from the 59 baseline trained evaluation DNNs.

BN was devised to correct for internal covariate shift by whitening activation values, allowing networks to use higher learning rates without amplifying activation values into non-linear functions saturation regions immediately (Ioffe, 2015). The benefits of BN are explicit in this evaluation section and motivate the decision to maintain BN layers in the optimised model.

3.5 Optimised Model & Evaluation

This section develops an *optimised* Deep Neural Network (DNN) model capable of explaining the behaviour of individual algorithmic traders interacting with UK equity FTSE100 CLOB high-frequency dataset by using insights from the evaluation performed above. DNN models built and experiments performed in this section are programmed with Keras 2.1.1 (Chollet, 2015) using TensorFlow v1.4 (Abadi, 2016) and cuDNN v7.0 (Chetlur, 2014) backend deep learning engines and primitive libraries. Keras provides a modular deep learning API framework for developing, training, optimising, and testing complex multi-layered neural network architectures, that can be run over TensorFlow. TensorFlow provides a comprehensive open-source deep learning library consisting of Python modules that allow for the efficient construction of computational graphs, which are capable of distributed training and processing in parallel across multiple CPU or GPUs. Computational data flow graphs represent the set of mathematical operations, manipulations and functions performed to and between nodes comprised of tensors, or multi-dimensional data arrays, which can for example represent an optimisation algorithm. TensorFlow is ubiquitous within practitioner and academic domains for developing and training DNNs, primarily given its ability to perform extensive array manipulations efficiently. Training of the DNN is conducted on an Intel i7 CPU using 32GB RAM and a NIVIDA GPU. Certain functions utilise parallel computing and distributed storage methodologies. There are significant technical challenges that require cloud computing storage and complex matrix computations for several experiments performed, given the vastness and diversity of the datasets analysed.

The optimised DNN model consists of 4 dense hidden layers with 500, 300, 300 and 200 neurons positioned in an interconnected feedforward multi-dimensional hierarchal architecture to ensure a high degree of feature abstraction and a vast representation space of possible non-linear functions. Each hidden layer applies an Exponential Linear Unit (ELU) neuronal activation with parameter α equal to 0.1 to provide the non-linearity for each layer, with output bounded by $(-1, \infty)$ (Clevert, 2015). Higher alpha values that bound the activation function closer to zero performed more effectively in the evaluation of the baseline model, given its propitious property of inducing more sparsity into the network. The output layer of the DNN utilises Softmax activation functions which is pertinent given the categorical nature of the FTSE100 CLOB classifiers. Batch Normalisation (BN) layers are added before the non-linearity in each hidden layer with e^{-3} epsilon ϵ, beta β initialised to zero, and gamma γ initialised to one, with no regularisation applied to the BN layers parameter weights (Ioffe, 2015). Dropout regularisation is used for each hidden layer with dropout probability rates of 0.5, 0.4, 0.4 and 0.3, respectively (Srivastava, 2014). Initialisation of the networks kernel weights is performed using the random orthogonal matrix technique with scaled gain factor g of 0.5 (Saxe, 2014). The neural network attempts to fit a complex non-linear mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, \mathbf{b})$, to the training dataset by minimising the categorical Cross-Entropy cost function, $\mathcal{L}^{CE}(\mathbf{w})$. Training of the network is performed using the Nesterov Adam (NAdam) adaptive optimisation algorithm with learning rate a of 0.1, β_1 of 0.99, β_2 of 0.9, schedule decay of zero and small e^{-3} epsilon ϵ (Kingma, 2015; Dozat, 2016). NAdam is preferred to the Adam algorithm given the positive results the algorithm attained in the previous evaluation. Batch sizes for each iteration is 128 data samples, and the algorithm for each dataset is run over 100 epochs. Early stopping techniques are applied using systematic rules that requires a trade-off between reducing training time and improving generalisation (Prechelt, 2012). A stopping criterion that halts training when the validation accuracy does not improve after a patience level of 10 epochs is

utilised. The architecture of the DNN model maintains a capacity of 472,030 parameters trained through each epoch learning epoch iteration.

Unique DNN models are developed for each Security, Date, Broker tuple, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$. Thus, a filtered dataset is extracted from the consolidated FTSE100 CLOB dataset, defined in Section 3.3, for each trader performing a minimum of 1,000 actions in a specific stock of a certain trading day. In total, 859 datasets that meet the trader action threshold are developed with a separate DNN model trained for each dataset. The objective of the DNN is to optimise the learnable weight $\mathbf{w}$ and bias $\mathbf{b}$ parameters for the non-linear function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w}, \mathbf{b})$, composed of the limit order book microstructural feature space, $\mathbf{x} \in \mathcal{F}$, and classifiers or actions executed by each trader, $\mathbf{y} \in \mathcal{A}$. Whilst features are developed over the whole trading day, the DNN models only focus on learning from the continuous trading period of 9 am to 4 pm. Primarily, this is to extricate strategies conducted during volatile periods in the market, before 9 am, and during pre-auction periods, after 4 pm, which offer no utility for comprehending the AT strategies of interest as defined in Section 2.3. This analysis extends from the evaluation by optimising the baseline model, which only looked at the 59 datasets with more than 10,000 actions, through the improvement of DNN configurations, parameters and hyper-parameters and the deployment of the model to *all* traders executing 1,000 actions. For the purposes of developing the DNN predictive model, a stratified 70/10/20 split of the data for each $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$ DNN tuple is partitioned into a training, validation and test dataset using random sampling without replacement.

3.5.1 Optimised Deep Neural Network Evaluation

DNN models are configured, built, trained and evaluated for each of the 859 DNN Broker tuples, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$. First, an experiment is designed to heuristically assess whether a firm is conducting an algorithmic trading strategy using inputs derived from the LOB. Firms deemed not to be conducting an algorithmic trading strategy for the purposes of this chapter are extricated from the dataset manifold. Next, an analysis of the optimised DNNs performance is conducted along dimensions of training accuracy, generalisation and computational speed reported.

Brokers are defined as conducting an algorithmic trading strategy if they utilise inputs from the LOB to inform their trading algorithms in a way that influences their decisions, actions and behaviour. DNN models are fitted to each of the 859 Broker tuples, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, with the models' ability to map the relationship between LOB input features and trader actions assessed. By analysing the internal dynamics and performance of each DNN, one can determine whether there exists some complex non-linear relationship between the LOB features and trader actions. When such a relationship cannot be formulated, it can be assumed that traders are taking actions based on some other phenomenon in financial markets or as a personal internal decision of the trader. A total of 57 firms are taken to ***not*** be employing an algorithmic trading strategy. That is, the DNNs trained on these firm's datasets fail to converge with a high degree of accuracy such that the model cannot predict the trader actions given their behaviour throughout the trader day when faced with a state of the LOB. Failure of convergence is defined as DNNs not attaining a minimum 70% validation accuracy during training. Whilst it is common that these 57 Brokers relationship can be mapped with higher accuracy, generally over 90%, for the training dataset, the model does not generalise outside the training data. Two potential reasons can explain why the DNN cannot accurately develop predictions of these trader actions. First, these

brokers could potentially have slow automated trading systems that are not captured by the DNN given that the firm's state representation of the LOB is stale. Second, the decisions of the relevant traders may be derived from implicit features to the firm that utilises a non-LOB feature based strategy to implement actions on the LOB. Research of the relevant firms not meeting the threshold indicates that four investment banks predominantly encompass the 57 samples are primarily, with these banks generally conducting these strategies under one of three or four Broker ID codes that the bank generally uses. Potentially, these banks have a separate desk that trades under these Broker ID codes through which some form of non-algorithmic order flow is submitted through to the trading venues. Regardless, the 57 DNN Broker tuples, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$ that fail to converge with a high degree of accuracy are extricated from the analysis and defined as executing a non-algorithmic trading strategy. Note that the 57 Brokers are split over five securities and five trading days indicating that less than three Brokers for each ISIN-Date dataset are annulled from the analysis.

An analysis of the remaining 802 optimised DNN Broker tuples, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$ for firms executing algorithmic trading strategies is undertaken along the performance dimensions of training accuracy, generalisation ability and computational speed. Results averaged across all Brokers in the dataset are reported in Table 3.5.1 for each ISIN.

Optimised DNN		Training Set			Test Set			Training Time (Sec)		Epochs
ISIN	Firms	Accuracy	Loss	SD	Error	Loss	SD	Mean	Per Epoch	Mean
GB0002875804	135	95.87%	0.1203	0.0233	7.36%	0.3814	0.0735	364	5.45	67
GB0005405286	167	96.89%	0.0939	0.0347	6.37%	0.4065	0.0640	493	7.02	70
GB0007980591	165	95.07%	0.1432	0.0371	8.83%	0.4525	0.0812	542	7.90	69
GB0009252882	162	97.18%	0.0851	0.0188	7.39%	0.4178	0.0697	565	7.59	74
GB00BH4HKS39	173	96.77%	0.0966	0.0278	5.09%	0.2984	0.0605	510	7.49	68
Overall	**802**	**96.38%**	**0.1073**	**0.0303**	**6.99%**	**0.3907**	**0.0710**	**499**	**7.15**	**70**

Table 3.5.1 – Evaluation of the optimised DNNs trained over the sample securities for the 802 Broker dataset tuples of traders assessed to be conducting algorithmic trading strategies. Performance is assessed along dimensions of predictive accuracy during training, and inference, in addition to computational and convergence speed. No data augmentation is used. Models are trained with early stopping. Total training time for optimised model evaluation is 158.29 hours on a 32GB i7 CPU using tensorflow distributed libraries.

Results indicate that the four hidden-layer optimised DNN model can be trained to classify the actions of AT firms given the state of the LOB with a training accuracy of 96.38% on average over the 802 datasets evaluated. These accuracy levels compare favourably with widely studied image (Clevert, 2015; Ioffe, 2015) and speech recognition (Maas, 2013) classification problems that are generally optimised with very deep networks. Model accuracy rates have low variability across the DNNs trained, with a minimum accuracy of 77.9% and standard deviation of 0.0303 reported across the models evaluated. Average loss of 0.1073 is formulated using a categorical cross-entropy cost function framework, with a value over zero indicating that some samples remain misclassified. The reported average generalisation error of 6.99% over the test datasets indicates that the DNN model generalises relatively well outside the training. However, there appears to be overfitting of the training dataset to a degree which is demonstrated by the relatively higher training to validation and test accuracy levels. This issue is caused by several DNN models with relatively lower sample sizes, closer towards the 1,000 action threshold, learning representations very quickly given the small dataset and high computational power of the model. One of several regularisation techniques employed in the optimised DNN, early stopping prevents severe overfitting whilst also controlling the level of computational resources expended by stopping training

when the model begins to overfit given the stopping criteria employed. Thus, early stopping criteria could potentially be augmented to incentivise an earlier convergence equilibrium for the DNN to improve its ability to generalise outside training. The convergence properties of the DNNs are in line with expectations taking an average 68 epochs per model to converge to an optimal local minimum, with a standard deviation of 21 epochs.

Figure 3.5.1 displays the average DNN training and test accuracy over 100 training epochs as an average of the 802 DNN Broker tuples trained. The figures illustrate a high degree of commonality between the models trained for the Brokers trading in the different ISINs, a relatively high degree of accuracy and sound convergence speed and properties, aligning with the results in Table 3.5.1. The figure provides some evidence that overfitting is a concern, given that the validation cure is generally tracking the training curve but still behind by several percentage points on average and beginning to plateau as the model begins to stabilise and converge into in a local minimum. As discussed above, this is caused by several of the small datasets learning patterns in the data and optimising their model weights very quickly, with the early stopping criterion not terminating learning until several epochs later, resulting in a high training accuracy to test accuracy ratio. Furthermore, the ability of DNNs to learn the complex, abstract and non-linear mapping function relating inputs to outputs is evident by the significant and fast increase in training accuracy after initialisation, with the generalisation error dropping over 80% within the first 35-30 epochs on average. This result also indicates that the optimisation function is learning at a rate that is not too low and the algorithm converges to a good local minimum on the error surface rather quickly and in a stable manner. The trajectory of the training accuracy graph also indicates that the algorithm's learning rate is not tuned too high and the weight initialisation strategy employed initialises the DNN in an error surface region with useful topological features.

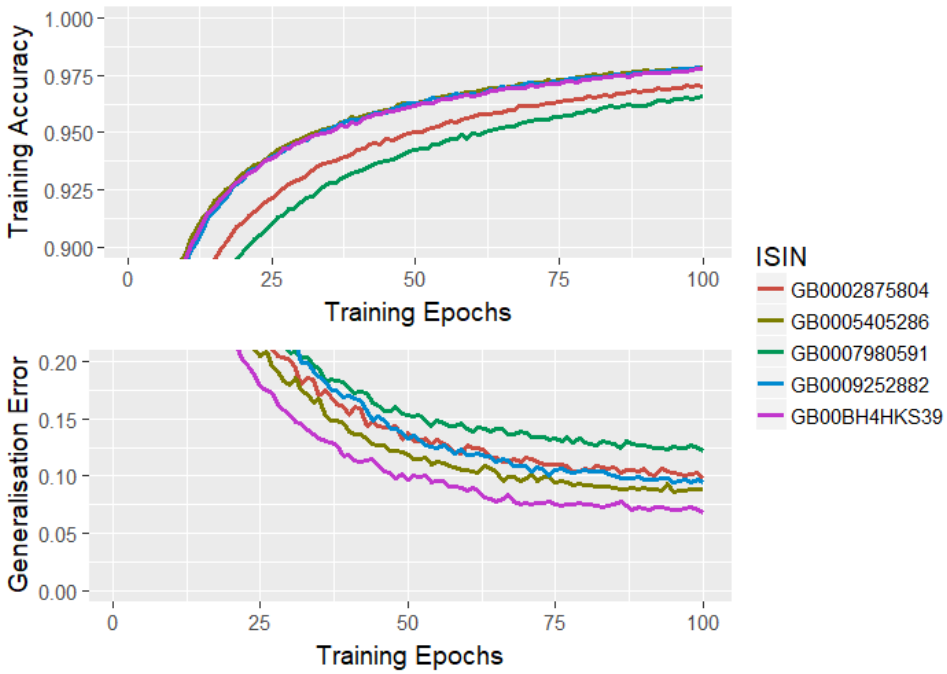

Figure 3.5.1 – Comparative evaluation of the accuracy and generalisation error for the optimised DNN models averaged over 802 Broker tuples.

3.5.2 Confusion Matrix & Quality Metrics

An analysis of the optimised DNN models quality utilises a confusion matrix and related quality metrics that quantify the DNNs success at correctly classifying samples with a high degree of accuracy versus samples where the models' predictions of trader actions given the state of the LOB is significantly wrong. There are three primary sources of error in most neural network classification problems – incorrect labelling, low input data quality, and residual model error. Given that the dataset is generated based on AT firms' activity on LOBs there is no human intervention in classifying actions but rather they are based on what message the firm sends to the exchange at a point in event time. However, as identified previously, there is an inherent limitation in the dataset where messages are recorded at the millisecond level, though they are identified by sequential identification numbers at the exchange level, which introduces potential labelling errors during extremely high message throughputs periods across exchanges. This issue fits at the nexus of incorrect labelling and reduced data quality. DNNs trained with impure data or classes are likely to perform poorly yet given the high accuracy of our DNN models there is evidence they capture a coherent and holistic representation of how AT strategies are executed and the relevance of various input features into those strategies.

Confusion matrixes provide a holistic image of the networks accuracy when classifying samples. Metrics regarding the size of individual classes, class distributions, and classification performance are capable of being extracted from the matrix data. For the combined set of 802 DNN Broker tuple $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$ datasets, a confusion matrix, $\mathbf{M}_{ij} \in \mathbb{R}^{\mathcal{A} \times \mathcal{A}}$, is generated for all classes in $\mathcal{A}$ that compares the models predicted class versus the sample's actual class, which is averaged across all datasets. The matrix diagonal $\mathbf{M}_{ii}$ quantifies the average percentage of samples that are correctly predicted by the model for each class, whilst the elements outside the diagonal epitomise the DNN's inability to correctly classify a traders' action. For example, element $\mathbf{M}_{ij}$ will refer to the samples for class i that were incorrectly classified as class j by the DNN.

The confusion matrix is reported in Table 3.5.2 and graphically represented in Figure 3.5.2. Note that Iceberg and Immediate-or-Cancel (IOC) orders are amalgamated into a single class for both the bid and ask side, given the relatively low proportion of these action types relative to limit orders. The confusion matrix indicates that the DNNs are successful at classifying most predictor classes in the training datasets with a high degree of accuracy across the diagonal. Limit orders at all levels up to five ticks from the BBO on both the bid and ask side, a total of 8 actions, are classified accurately on average 98.3% of the time. The DNN is relatively less capable of classifying market orders, which on the bid side are classified correctly for 90.7% of actions whilst on the ask side the model attains only a 90.6% accuracy. Furthermore, both market order types are incorrectly classified as one of nine other cases in more than 0.1% of cases, a figure which is significantly higher than limit orders which have an average of approximately three classes against which they are incorrectly classified in over 0.1% of cases. This indicates that market orders get mixed up with other actions that a trader can perform in comparison with limit orders which are mixed up with relatively fewer types of other orders. The rationale behind this result is that market orders sit at the nexus of the bid and ask side of the order book, where for a given set of LOB features traders could place a market order, or a limit, iceberg, or IOC order at or within the BBO, all trying to execute a trade with a degree of aggressiveness. Additionally, market orders may be driven by non-LOB or trader-specific factors not inherent within the FTSE100 dataset,

perhaps due to demand from a client for liquidity or to close out a position if a risk limit is reached. Thus, it becomes more difficult to predict whether a market order is going to be placed vis-à-vis passive limit orders. Iceberg orders attain the lowest prediction accuracy of all order types on average with orders on the bid side being correctly predicted by the DNN with only 60.9% accuracy on average and a marginally better 72.8% accuracy on the ask side. However, both these order types are relatively rare with prediction accuracy calculated off a base number of actions that is less than 0.1% of the total number of actions across the datasets.

DNN \ Actual	MO.B	IOC.B.P0	LO.B.P0	LO.B.P1	LO.B.P2	LO.B.P3	ICE.B.P0	MO.A	IOC.A.P0	LO.A.P0	LO.A.P1	LO.A.P2	LO.A.P3	ICE.A.P0
MO.B	90.7	0.7	0	0	0	0	0.2	0.1	1.2	0.9	0.1	0	0	4.8
IOC.B.P0	0.1	81.5	0.2	0	0	0	0.3	0.1	0.3	0	0	0	0	0
LO.B.P0	0.4	13.2	98.5	0.2	0	0	30	7.6	1.8	0.2	0	0	0	1.3
LO.B.P1	0.1	0.2	0.2	99.5	0.5	0	0.7	0.7	0	0	0	0	0	0
LO.B.P2	0	0	0	0.2	98.7	0.3	0	0.1	0	0	0	0	0	0
LO.B.P3	0	0	0	0	0.7	99.7	0	0.1	0	0	0	0	0	0
ICE.B.P0	0	0	0	0	0	0	60.9	0	0	0	0	0	0	0
MO.A	0.2	1.7	0.7	0.1	0	0	7	90.6	0.5	0	0	0	0	0.2
IOC.A.P0	0.1	0.3	0	0	0	0	0.1	0.1	82.3	0.2	0	0	0	0.1
LO.A.P0	7.4	2.4	0.2	0	0	0	0.9	0.5	13.7	98.3	0.1	0	0	19.9
LO.A.P1	0.8	0	0	0	0	0	0	0.1	0.2	0.3	99.5	0.2	0	0.9
LO.A.P2	0.1	0	0	0	0	0	0	0	0	0	0.3	98.8	0.1	0
LO.A.P3	0.1	0	0	0	0	0	0	0	0	0	0	1	99.9	0
ICE.A.P0	0	0	0	0	0	0	0	0	0	0	0	0	0	72.8

Table 3.5.2 – Confusion matrix results for the 802 Broker tuple DNNs trained. The vertical axis represents actions predicted by the DNN model when passing validation data through the model. The horizontal axis represents the actual actions executed by traders. Results are average across the datasets and reported as percentages. Values less than 0.1 are reported as 0.

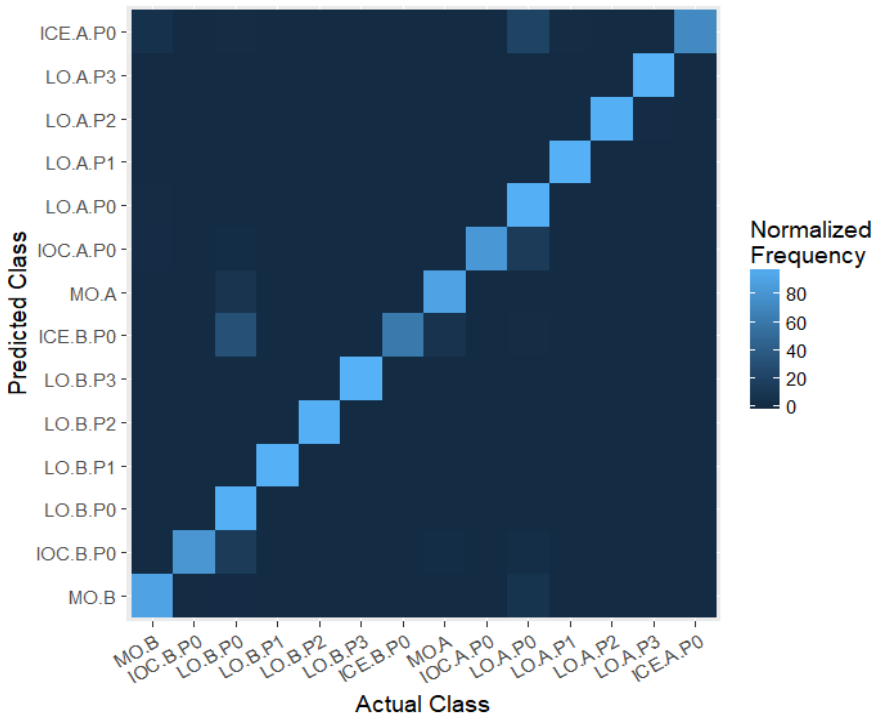

Figure 3.5.2 – Confusion Matrix visualisation of Broker tuple datasets after optimisation. Diagonal elements represent correct predictions.

Comparative analysis of the DNN predictive capabilities for specific classes is conducted using class-wise confusion metrics. These metrics provide an additional method of testing the relationship between two different classes. Several metrics are employed including a basic action count, a relative class size, accuracy rates and confusability metrics, F_{ij}, or class i and j, which measures the proportion of samples where class i is incorrectly classified as class j. The mathematical derivations of the various quality metrics follow:

$$Count(i) = \sum_{j=1}^{\mathcal{A}} \mathbf{M}_{ij} \qquad Relative\ Size(i) = \frac{Count(i)}{\sum_{i=1}^{\mathcal{A}} \sum_{j=1}^{\mathcal{A}} \mathbf{M}_{ij}} \qquad Accuracy\ Rate(i) = \frac{\mathbf{M}_{ii}}{Count(i)}$$

$$F_{ij} = \frac{\mathbf{M}_{ij}}{\sum_{j=1}^{\mathcal{A}} \mathbf{M}_{ij}}$$

DNN quality metrics for the available traders' actions are listed in Table 3.5.3 in addition to the action with the highest confusability percentage for each class. The primary actions executed are orders submitted at either the BBO or within one tick of the BBO, with both categories on the bid and ask side seeing over one million actions each across the entirety of the datasets. These four actions together represent 61.2% of all actions. As expected, both Iceberg and IOC orders are most commonly confused with limit orders at the BBO on their respective LOB side. An average of 30% bid and 19.9% ask Iceberg orders, and 13.2% bid and 13.7% ask IOC orders, are incorrectly classified as BBO limit orders, though these order types do have a relatively smaller base than limit orders. Another interesting result is that bid market orders are most often confused with limit orders on the ask side at the BBO, whilst the inverse is true for ask market orders which are confused as limit orders on the bid side for 7.6% of the actions. Evidently, there is a high degree of accuracy for most of the limit orders which indicates that the DNN is working well at classifying traders' actions correctly.

Action	Count	Relative Size	Accuracy Rate	Confusability	F_{ij} (%)
MO.B	127454	1.6%	90.7%	LO.A.P0	7.4%
IOC.B.P0	23478	0.3%	81.5%	LO.B.P0	13.2%
LO.B.P0	1214464	15.2%	98.5%	MO.A	0.7%
LO.B.P1	1222827	15.3%	99.5%	LO.B.P2	0.2%
LO.B.P2	480547	6.0%	98.7%	LO.B.P3	0.7%
LO.B.P3	918958	11.5%	99.7%	LO.B.P2	0.3%
ICE.B.P0	2184	0.1%	60.9%	LO.B.P0	30.0%
MO.A	131416	1.6%	90.6%	LO.B.P0	7.6%
IOC.A.P0	27941	0.3%	82.3%	LO.A.P0	13.7%
LO.A.P0	1233975	15.3%	98.3%	LO.A.P1	0.3%
LO.A.P1	1239623	15.4%	99.5%	LO.A.P2	0.3%
LO.A.P2	482433	6.0%	98.8%	LO.A.P3	1.0%
LO.A.P3	886435	11.1%	99.9%	LO.A.P2	0.1%
ICE.A.P0	2919	0.1%	72.8%	LO.A.P0	19.9%

Table 3.5.3 – Quality metrics computed for the 802 optimised DNN datasets. Metrics include basic count, relative size, accuracy rate, most common confusability class, and the confusability metric, F_{ij}, for the most common class.

3.5.3 Internal Synaptic Weights Analysis

Weight distributions that feed data through synaptic connections between DNN layers are graphically presented for each layer using violin plots in Figure 3.5.3 for the interval that covers 95 percent of the weights. Weights are amalgamated across the optimised DNN models for each layer with the range, mean, standard deviation, skewness and kurtosis reported in Table 3.5.4 for each layer separately.

Results indicate that all layers have stable and minor asymmetric distributional properties about the origin with an approximate Gaussian shape. As expected, the connection between the input layer and the first hidden layer exhibits a higher density of weights around zero value, with a mean of -0.07 and standard deviation of 4.88, indicating an absence of any single dominating input feature neuron. All hidden layers are platykurtic, with a negative excess kurtosis, indicating a lower probability of outliers further from the mean, with their values are relatively close to zero, indicating they nearly approximate a univariate normal distribution. Additionally, hidden layers are slightly positively skewed such that the tail in the positive region of the distribution is longer, with a higher proportion of the layer weights having negative over positive values. This is expected given the redundancy in certain features that are auto-correlated over time.

Deeper layers are capable of applying more complex abstractions to the input features leading to a wider representation space, evident by the relatively wider dispersion and flatter shape of layers L2, L3, and L4. These deeper layers all have a higher kurtosis and standard deviations than the weights of the first hidden layer L1. The wide range of weights in these deep layers is an additional indicator of the utility of adding more hidden layers to the network and provide a justification for deploying a deep learning model such as a DNN over for example, a decision tree-based random forest (RF) to explain the relationship between LOB features and trader actions. RFs tend to view input features in solace rather than considering the multitude of connections and abstractions developed through operations between multiple features, which the DNN does, that may explain traders' actions.

The output layer's weights are densely located around a mean of -0.15, evident by the high kurtosis of 2.64 indicating a heavy-tailed leptokurtic distribution. This phenomenon can be explained by the combination of a large number of categorical classes, high levels of DNN model accuracy, and the dominance of certain actions, such as limit orders, in true predictor labels. A further element to consider is the fact that several of the categorical class output neurons rarely fire, such as iceberg and IOC orders far from the BBO, with the synaptic weights connected to these neurons rarely updated and maintain low weight values close to zero.

Layer	Min	Max	Mean	SD	Skewness	Ex. Kur.
L1	-12.50	12.04	-0.07	4.88	-0.03	-0.18
L2	-16.93	15.17	-0.22	6.99	-0.10	-0.53
L3	-17.98	14.57	-0.42	6.91	-0.18	-0.39
L4	-19.25	15.53	-0.32	7.21	-0.23	-0.29
Output	-13.97	11.29	-0.15	4.01	-0.43	2.17

Table 3.5.4 – Synaptic weight distribution statistics for each layer of DNNs trained over the 802 Broker tuple datasets. Metrics include minimum weight, maximum weight, mean, standard deviation, skewness and excess kurtosis. Weights are winsorised at the 99% level.

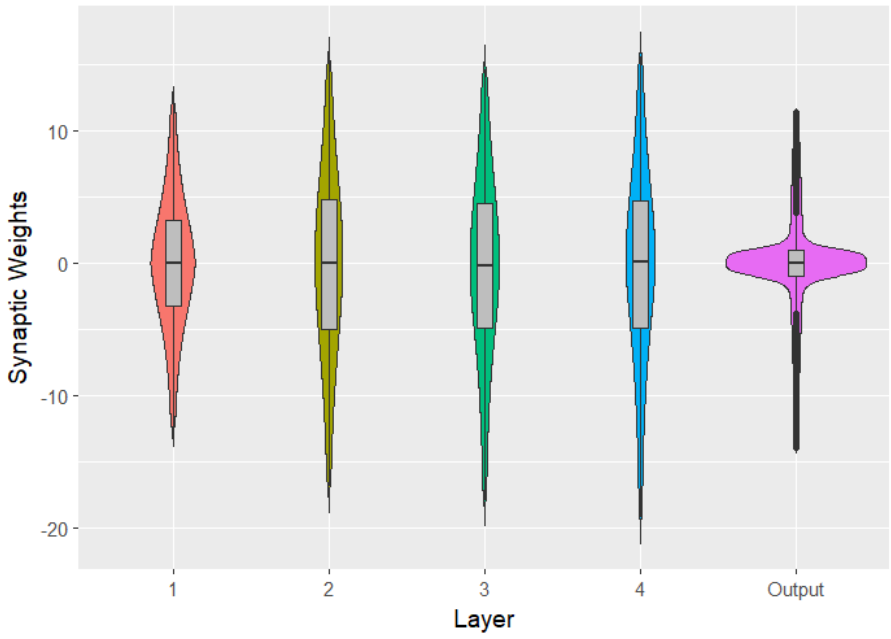

Figure 3.5.3 – Violin plots of hidden layer and softmax output layer weights for the optimised DNN model trained on the FTSE100 dataset. Weights for each layer encompass the interval consisting of 99% of weights.

3.5.4 Robustness Against Adversarial & Random Perturbations

DNN models must be sufficiently robust against perturbations in input feature values to validate the utility of the network as a predictor and its ability to generalise. Note that input features undergo pre-processing scaling and normalising operations as specified in Section 2.4. Sources of poor model performance and generalisation may arise due to a lack of robustness to the prevalence of adversarial samples in the dataset, perturbations in the model architecture, or random variations in input feature values.

Firstly, adversarial samples involve a perturbation, δ, to an input feature vector that provokes an incorrect or adversary-selected classifier when propagated through a DNN (Goodfellow, 2014; Papernot, 2016). Perturbations may not necessarily manipulate the sample such that it seems nefarious or exists outside the training data manifold, but rather can involve slight modifications that prohibit the DNNs ability to generalise well to test data. The ability of a single algorithmic trader to manipulate exchange or market-wide financial metrics is significantly low, thus, sufficient confidence that the input features of the dataset are not prone to adversarial manipulation exists. Defensive distillation is one technique used to reduce adversarial gradients, improving the resilience of DNNs to adversarial sample manipulation (Papernot, 2016). Secondly, robustness to DNN architecture or internal weight perturbations has been theoretically and empirically explored, for example, in cases where weight synapses are removed (Cheney, 2017), though has minimal relevance in the context of evaluating our DNN model. Finally, robustness to random perturbations in input feature values common in the noise-prone environment of financial markets is a fundamental property of a well-trained DNNs that is not overly dependent on the values of a certain subset of features or abstractions. Hence, this section evaluates the models' robustness against random perturbations in this context.

Robustness of DNN models to distortions created by random noise is analysed by examining the loss, performance accuracy and classifier decision surface of the model when applying random imperceptible perturbations to input features. Testing general model robustness is performed by adding sampled uncorrelated Gaussian noise, $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$, to each input feature, $x_i + \epsilon_i$. Adding random noise is a novel approach to augmenting data that simulates various classes of perturbations that may induce instability in the class predictor, which a robust model should be capable of impounding without significant reduction in accuracy rates.

Results for DNN models trained with an input feature vector that has random perturbations of 0.01, 0.05, 0.1, and one standard deviation from mean zero are averaged over all 802 models and reported in Table 3.5.5. As one would expect with a robust DNN model, minor perturbations to input values do not significantly affect training accuracy. Applying error perturbations of 0.01 standard deviations only results in a 0.1% decrease in training accuracy and additional 0.004 units of loss in the cross-entropy cost function on average over the optimised DNNs. As the random perturbations increase in relative magnitude, the impact on training accuracy becomes more pronounced. By adding random noise to input features drawn from a normal distribution with one standard deviation from mean zero results in a model with poor representative or generalisation power. These models exhibit severe overfitting in early epochs as the model fits directly to the training data though with minimal generalisation ability. The test generalisation error of the model begins to reduce gradually over learning epochs, stabilising at an average of 54.45% higher generalisation error than the optimised model.

	Average divergence from optimised DNN			
Error SD (σ_i)	Training Accuracy	Training Loss	Gen. Error	Test Loss
$\sigma = 0.01$	-0.10%	0.004	0.13%	0.003
$\sigma = 0.05$	-1.13%	0.030	2.79%	0.083
$\sigma = 0.1$	-3.12%	0.080	8.65%	0.265
$\sigma = 1$	-9.12%	0.259	54.45%	2.576

Table 3.5.5 – Analysis of DNN models robustness to random perturbations in the input feature vector. Training dataset and test dataset metrics are measured as the difference from the optimised DNNs trained with no noise in the input, averaged over the 802 DNNs. The vertical column defines the level of noise imputed into the dataset with model robustness performed by adding sampled uncorrelated Gaussian noise, $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$, to each input feature, $x_i + \epsilon_i$.

One can draw the conclusion that the DNN models are on average quite robust to small random perturbations in input feature values, though there exists a limit to the size of the noise applied to the models at which their generalisation error increases significantly, rendering them inapplicable for useful analysis. One can see this from the impact on training accuracy from small perturbations less than 0.1 standard deviations is only marginal, whilst the larger random perturbations lead the model to have poor convergence properties and fails to learn a good representation of the input-output relationship. Various methods have been developed to improve datasets that are not robust to noise. For example, Zheng (2016) applies stability training to improve class output robustness for Convolutional Neural Networks (CNN) with natural pixel distortions by training a constant model with a stability training objective on data with minor perturbations to input pixel values. However, given the robust nature of the optimised DNN models trained on the FTSE100 CLOB datasets, it does not appear necessary to apply these techniques.

3.5.5 Sample Size vs Accuracy

Deep learning models are designed to scale well with expansive datasets. The generalisation and accuracy of deep neural networks are enhanced by increasing the number of samples that the model can learn from (Goodfellow, 2016) at the cost of additional computational resources. More complex representations of the models' non-linear function, which are generally limited by model capacity and dataset size, can be learnt from datasets with more input-output samples. This section involves a basic analysis of the relationship between sample size and the models' performance and computational training requirements. However, given that each ISIN-Date-Broker DNN is predicated on a trading algorithm intrinsic to that Broker, it is difficult to draw direct parallels between different models based on merely their training sample size.

The correlation between sample size and training accuracy, test accuracy and training time per epoch is computed. The results indicate that DNNs trained with a higher number of samples generalise better and more accurately approximate the input-output mapping function than those trained with fewer samples. The correlation between sample size and test accuracy is 0.21 which predicates a positive relationship between sample size and the accuracy of the model when predicting samples in the test dataset, indicating better generalisation capabilities for DNNs trained with a larger quantum of samples. The computational cost from training DNNs with additional sample sizes is evident in the high correlation of 0.6 between the mean epoch training time and sample size. All DNNs trained in this section have a minimum sample size of 1,000 trader actions which from the results tabulated thus far, seems to indicate a vast enough dataset to draw good representations of the relationship between trader actions and LOB features.

3.6 Comparative Machine Learning Model Evaluation

In this section, a comparative framework is developed to assess the generalisation and predictive capabilities of Deep Neural Networks (DNN), trained on the unique ISIN-Date-Broker tuples $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, by comparing the models' performance against that of other common machine learning models deployed in the literature. Machine learning models approximate the non-linear function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$, by optimising synaptic weights in a way that more accurately predicts an algorithmic traders actions, $\mathbf{y}$, given a set of FTSE100 LOB features that represent the state of the financial environment, $\mathbf{x}$. Optimised DNNs developed over the previous section are evaluated along dimensions of accuracy, generalisation and learning speed through a comparative analysis encompassing five machine learning models – Random Forests (RF), C5.0 Decision Trees (C50), soft-margin Support Vector Machines (SVM), Multinomial Logistic Regression (MLR), and a Multinomial Naïve Bayes model (NB).

Evaluation is performed on each of the 59 ISIN-Date-Broker datasets that have a minimum 10,000 actions executed by the relevant Broker in a security across the trading day. A test harness is developed to split each ISIN-Data-Broker dataset into data subsets on which each model is trained, validated and tested, allowing an empirical assessment of each models predictive and generalisation power. All datasets are drawn from the same underlying distribution randomly, with the test set not exposed to the model during training. Training datasets also have a validation component, depending on the model, to placate overfitting concerns from the model memorising inherent patterns in the training dataset.

3.6.1 Generalisation & Predictive Power

Comparative machine learning models are trained then evaluated using a cross-validation analysis to assess their generalisation and predictive power. Performance is measured and averaged over the 59 ISIN-Date-Broker datasets through five dimensions – training accuracy, generalisation error, dispersion of errors, convergence speed and a Cohens Kappa statistic. Results for the analyses are presented in Table 3.6.1. Evidently, the superior predictive power and generalisation capability of DNN models is explicitly shown through the attainment of the lowest generalisation error, 2.04%, and highest training accuracy, 98.06%, among all comparative machine learning models assessed. Of the comparative models, RFs have the lowest average generalisation error of 2.91%, with both C50 decision trees and SVMs reporting comparable results with average errors under 4%. RFs are trained by growing 300 independent decision trees and using a random subspace of 17 features for splitting at each node. SVMs employ a cost basis of 100, gamma of 0.01 and epsilon 0.1, as default settings. C50 algorithms generate decision trees using a boosting technique with nodes split using an information gain criterion.

Practitioners in machine learning derive value from a model's generalisation and predictive power in real-world non-lab environmental domains. The evaluation performed on the comparative models indicates that DNN's are most successful at training a model that can generalisation well outside the dataset in which it is trained on, with a very low generalisation error, low error variance, and high Cohen's Kappa of 97.52%. Figure 3.6.1 provides a graphical depiction of how other models compare with the DNN's generalisation capabilities. The boxplots illustrate the dispersion of results for each model along the dimension of test accuracy, that is, how successful the model is predicting trader actions when exposed to the test dataset after training. As evident in the table and figure, one can see that RFs generalise well with C50 and SVM predictive power not significantly worse. Interestingly, the higher accuracy and predictive power of DNNs does not come at the cost of additional computational resources. These models train on average 20% faster than RFs. MLR and NB models do not compete with the more complex non-linear models in terms of performance, averaging generalisation errors of 11.17% and 22.36% over the Broker tuple evaluation datasets. However, they are capable of learning predictive functions in half the training time of DNNs.

The two log-linear models – MLR and NB – perform relatively poorly at training accurate mapping functions to explain the relationship between traders' actions, $\mathbf{y}_c$, and LOB features, $\mathbf{x}$. MLR generally provides a baseline framework against which other machine learning models can be compared. The MLR employed in this section utilises a logit link function, exponential activation function and softmax output layer to compress model estimates into a probabilistic interpretation of categorical classes. Weights are optimised using stochastic gradient descent (SGD) within the framework of a cross-entropy cost function. The model is trained using a maximum of 200 iterations to ensure severe overfitting does not occur. MLR is a discriminative model that estimates the probability of class c occurring given prior information regarding the feature vector, $p(\mathbf{y}_c|\mathbf{x})$, directly from the data, requiring minimal assumptions. In contrast, NB is a generative conditional probability model that learn the joint conditional probability distribution, $p(\mathbf{x}, \mathbf{y})$, before applying Bayes theorem to estimate $p(\mathbf{y}_c|\mathbf{x})$. NB models require strong assumptions that the data follows some novel distribution, which for the purpose of this evaluation a multinomial distribution is employed. NB models are based on the naïve assumption that features are independent.

These models ignore the *inter-feature* interactions and correlations that are captured by both non-linear DNNs and linear MLR models.

MLR models perform relatively well vis-à-vis NB models attaining an average training accuracy 11.19% higher and generalisation error 11.19% lower than NB models. However, NB models do train relatively quicker than MLR models, which aligns with the finding by Ng (2004) that NB models reach their asymptotic solution in fewer training iterations than MLR. Whilst both models use an identical hypothesis space, MLR models employ a different cost function and train using an SGD optimisation algorithm that fits all weights of the model together during learning. Conversely, NB models fit weights to LOB features independently based on correlations between features and classes. The success of discriminative MLR classifier models over generative NB models at accurately predicting trader actions in this chapter aligns with the theoretical and empirical literature (Ng, 2004).

Why do DNNs perform well in classifying traders' actions given the state of the LOB? The primary hypothesis is that DNNs are able to extract highly complex and abstract relationships between features due to the dynamism between intra-neuron connections in deep networks that comparative linear and decision tree-based models ignore. By building hierarchies of feature abstractions as individual features undergo non-linear activations and amalgamate in adjacent hidden layers, DNN's create a large representation space from which the non-linear mapping function that models the action-LOB feature relationship can be drawn. These unique properties have driven the near ubiquitous application if DNNs across the deep learning community for solving machine learning problems that require expressive and high-capacity models capable of learning for high-dimensional and complex features spaces. Conversely, decision tree models such as RFs and C50 algorithms respect the independence of LOB input features. Thus, these models are incapable of disentangling complex interconnections that may exist between features. This non-hierarchical property of decision trees prohibits the extraction of information or hidden explanatory factors in the raw data that may be relevant for modelling the input-output function. Raw LOB financial features by themselves can rarely predict the actions of algorithmic traders otherwise uniformity in strategy would collapse financial markets that attempt to match buyers and sellers. Rather, more abstract higher-level feature representations, formed through the amalgamation and non-linear transformation of multiple subsets of features, are more pertinent for use inside AT firms trading algorithms. Thus, DNNs provide a more relevant construct to observe, model and predict trader actions given the raw input features arising from the LOB state, when compared against other non-hierarchical machine learning models.

Machine Learning Model	Training Accuracy	Training Time (Min)	General. Error	Error SD	Cohens Kappa
Deep Neural Network	98.06%	8.9	2.04%	0.33%	97.52%
Random Forest	97.09%	10.7	2.91%	0.40%	96.47%
C5.0 Decision Trees	96.75%	10.4	3.25%	1.40%	97.36%
Support Vector Machine	96.66%	10.7	3.34%	1.35%	97.16%
Multinomial Regression	88.83%	3.7	11.17%	0.78%	86.40%
Naïve Bayes	77.64%	3.1	22.36%	1.06%	72.14%

Table 3.6.1 – Comparative analysis of machine learning models trained on 59 evaluation Broker tuple datasets. Performance is assessed along the dimensions of training accuracy, training time per model in minutes, generalisation error, error standard deviation, and Cohens Kappa.

The success of the DNN model in approximating the non-linear mapping function, $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$, justifies the decision to employ DNN's to model the underlying relationship between trader actions and LOB features. Once this relationship is accurately modified, it can be used to understand, define and segment firms by their algorithmic trading strategy, allowing for a unique perspective on how these strategies impact market quality, as presented in the next chapter of this thesis.

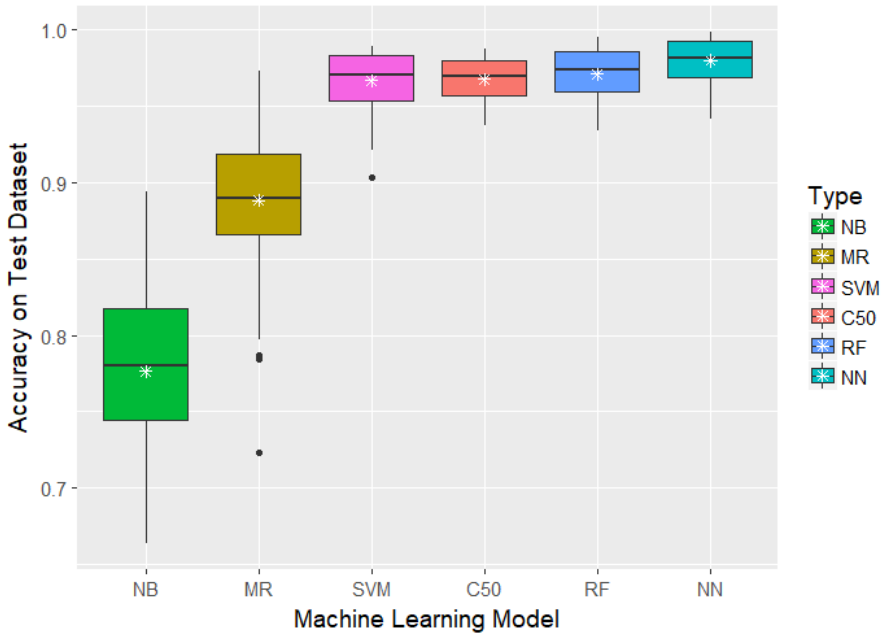

Figure 3.6.1 – Boxplot analysis of comparative machine learning models trained on 59 evaluation Broker tuple datasets. Midlines represent the median of the dataset, with the first and third quartile of the dataset represented by the hinges of the boxplot. Whiskers extend 1.5 times from the interquartile range, and the white dots represent the average prediction accuracy on the test dataset.

3.6.2 Confusion Analysis

A comparative evaluation of relevant machine learning models' quality of performance is conducted. This evaluation begins with a confusion analysis that studies each models' success at accurately predicting traders' actions from a given LOB feature vector in the test dataset. Confusion matrixes are first developed for the comparative models to provide a visual image of the models' accuracy at correctly classifying the test dataset. In addition, model accuracy is graphically depicted by comparing the correct reference versus the predicted classifier density distributions. The categorical nature of the classification problem requires a consideration of the models' capability of predicting each class, or action, individually. Quality metrics and a confusability analysis for three of the most common trader actions are studied, with the models compared through several quality metrics to evaluate their predictive capability. Note that Iceberg and Immediate-or-Cancel (IOC) orders are amalgamated into a single class for both the bid and ask side, given the relatively low proportion of these action types compared to limit orders. Furthermore, the confusion analysis focuses on the models with the strongest predictive power as per the above results – DNN, RF, SVM and C50 models.

3.6.2.1 Confusion Matrix

Confusion matrixes, $\mathbf{M}_{ij} \in \mathbb{R}^{\mathcal{A}\times\mathcal{A}}$, for classes $\mathcal{A}$ are generated and averaged across all the 59 evaluation datasets for four of the most successful comparative models – DNN, RF, C50, and SVM – with results presented in Figure 3.6.2. The high normalised frequency value for the matrix diagonal, $\mathbf{M}_{ii}$, for all models evaluated indicates that they are capable of accurately predicting the correct actions AT firms take given the LOB state, especially for limit orders and to a significant extent market orders. DNN models predict all limit orders correctly with an average accuracy of 98.7%, whilst RFs only average 97.6% and C50 decision trees 98.1% across the eight types of limit orders. Conversely, for the less popular market order action, DNNs only attain 89.4% of correct predictions, with RFs performing better with an average 91.6% accuracy. From the figures, one can note the poorer performance of RFs and SVMs at predicting iceberg and hidden orders. For these order types, the models have higher normalised frequency values which indicates a higher degree of confusion between the correct action and a range of alternative actions.

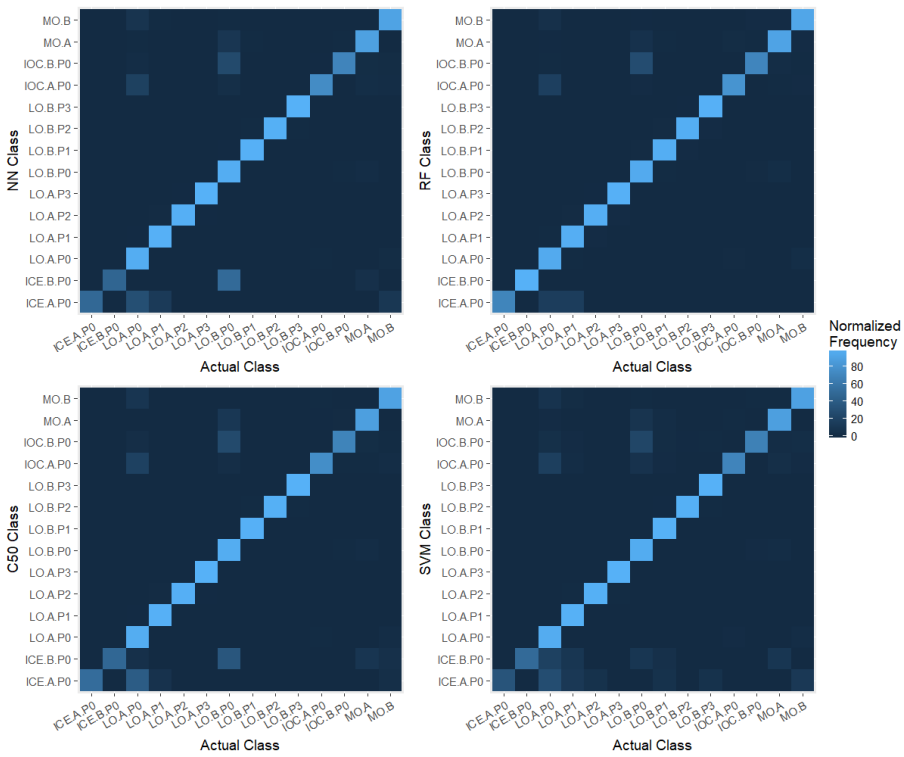

Figure 3.6.2 – Confusion matrix visualisation averaged over Broker tuple evaluation datasets for model predicted versus actual trader actions. Comparative models analysed include neural network (top left), random forests (top right), C50 decision trees (bottom left) and support vector machines (bottom right). Diagonal elements represent correct prediction.

3.6.2.2 Quality Metrics

Comparative evaluation of the different machine learning models requires a consideration of the quality of their performance in predicting different types of classes or actions of algorithmic traders. Various quality metrics are developed and presented for each model and three types of actions on the bid side of the LOB – market orders (MO.B), limit orders at the BBO (LO.B.P0), limit orders three to five ticks from the BBO (LO.B.P3). Quality metrics require the comparison between a single class, termed the positive class (P), and other alternative classes, termed negative classes (N). Positive classes may either

be true positives (TP) such that they are correctly predicted or false negatives (FN) where a negative class is incorrectly predicted as the positive class. Additionally, negative classes can be broken up into false positives (FP) where the negative class is incorrectly predicted and a true negative (TN) where the negative class is correctly predicted. For clarity these labels are graphically explained by Figure 3.6.3.

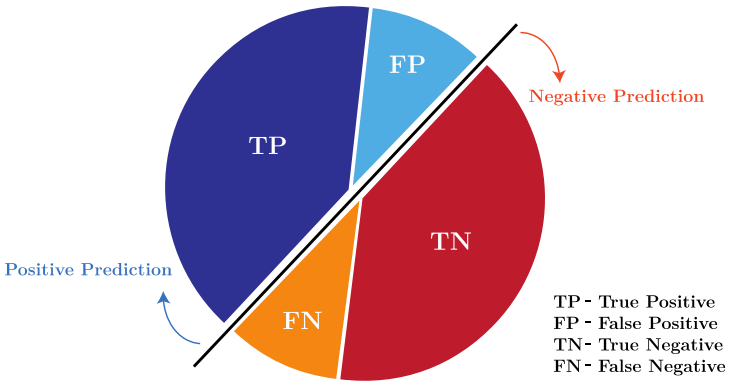

Figure 3.6.3 – Illustration of relevant quality metric components for both positive and negative predictions.

A series of quality metrics are presented for each model and the three actions or classes in Table 3.6.2. The first measure presented in the table are error rates which measure the percentage of incorrect predictions made by each model for a specific class. Results show that DNNs have the lowest error rates when predicting both LO.B.P0 and LO.B.P3 actions, with average rates of 2.9% and 0.2%, respectively. Three of the comparative models – C50, RF, and SVM – all have relatively low error rates under 6% and 1% for LO.B.P0 and LO.B.P3 actions, respectively. The predictive performance of DNN models is confirmed by computing balanced accuracy measures that replace conventional methods of averaging point estimates of accuracies by first correcting for biased classifiers in datasets that have imbalanced class count distributions (Brodersen, 2010). In line with the error rate results, DNNs outperform the other models for the limit orders tested, attaining 1% higher balanced accuracy over RFs for LO.B.P0 actions, whilst RFs outperform DNNs by 2.4% for MO.B orders.

Model	Action	Error Rate	Sensitivity	Specificity	Precision	F-Score	Balanced Acc.
C5.0 Decision Trees	LO.B.P0	3.3%	95.0%	99.5%	96.1%	95.5%	97.2%
Deep Neural Network	LO.B.P0	2.9%	95.1%	99.5%	96.3%	95.6%	97.3%
Random Forest	LO.B.P0	4.7%	93.1%	99.6%	92.8%	92.6%	96.3%
Support Vector Machine	LO.B.P0	5.7%	93.9%	99.6%	95.2%	94.4%	96.7%
C5.0 Decision Trees	LO.B.P3	0.3%	99.7%	99.5%	99.7%	99.7%	99.7%
Deep Neural Network	LO.B.P3	0.2%	99.7%	99.9%	99.8%	99.8%	99.8%
Random Forest	LO.B.P3	0.5%	99.3%	99.9%	98.9%	99.1%	99.6%
Support Vector Machine	LO.B.P3	0.3%	99.6%	99.9%	99.4%	99.5%	99.7%
C5.0 Decision Trees	MO.B	10.3%	82.6%	99.6%	70.3%	78.5%	91.1%
Deep Neural Network	MO.B	9.9%	87.6%	99.6%	76.6%	80.8%	93.6%
Random Forest	MO.B	7.5%	91.4%	99.4%	60.4%	69.6%	95.4%
Support Vector Machine	MO.B	11.0%	83.0%	99.5%	69.7%	74.6%	91.3%

Table 3.6.2 – Comparative analysis of machine learning models using quality metrics including error rate, balanced accuracy, sensitivity, specificity, precision and F-Scores. The quality analysis is performed on three action – LO.B.P0, LO.B.P3 and MO.B – with results aggregated over the 59 evaluation Broker tuple datasets. Metrics are computed by first calculating the true positive (TP), false positive (FP), true negative (TN), false negative (FN). True predictions (T) are equal to TP plus FP, whilst Negative prediction (N) are equal to TN plus FN. Quality metrics are defined as: $Error\ Rate = \frac{FP+FN}{P+N}$, $Sensitivity = \frac{TP}{P}$, $Specificity = \frac{TN}{N}$, $Precision = \frac{TP}{(TP+FP)}$, $F-Score = \frac{(1+\beta^2)*Precision*Recall}{(\beta^2*Precision)+Recall}$.

Comparative models are also evaluated across the three classes by their sensitivity and specificity scores. Sensitivity, also referred to as recall, is the number of correct positive predictions as a percentage of the total number of positive reference classes in the dataset. Conversely, specificity measures the correct negative predictions as a percentage of all negative classes. DNNs marginally outperform the comparative models on both measures for the two limit orders analysed, but once more the RFs attain the highest prediction rates for the market order. DNN's sensitivity measures for LO.B.P0 of 95.1% indicate that the model is good at predicting actions of firms when they place orders at the bid BBO. When analysing how LOB features impact different trading strategies it is important to understand why a trader will place an order at the BBO rather than cross the spread to make a trade. Thus, it is of critical importance for the model to correctly predict BBO orders given that these actions can be easily confused by the model for bid orders at one tick or even market orders. Most of the models correctly predict the LO.B.P3 action when it is executed by the trader with all models evaluated attaining sensitivity rates over 99%. The reason for the high correct positive predictions is that the model finds it easier to predict actions at the boundary of choices available, with LO.B.P3 being the choice to place an order furthest from the BBO that the action space allows. Thus, these orders are more rarely confused, whilst other limit orders near the BBO, where there are more actions available in a close action space, can be more easily confused. In regard to specificity quality metrics, the results show that most of the models have high average specificity measures for all three types of orders, with no model attaining a specificity under 98% on average. This indicates that the models are successful at correctly rejecting the positive class label when predicting negative classes.

Precision metrics are also developed to compare the number of correctly predicted positive classes with the total number of positive class predictions made. DNN's attain the highest precision scores for all three class actions tested. Precision for the LO.B.P0 of 96.3% is most closely matched by that of C50. In a multi-class problem, this measure provides a distinction between models that become confused with the correct class by selecting an array of other classes incorrectly. It can also be noted that the DNN performs best in the MO.B class, computing a precision metric of 80.8%, 6.2% higher than RF's, which had the lowest error rate for this class. This indicates that the RF may be selecting a large number of false positive MO.B predictors. There are clear indications that the RF performs well, however, the model does not appear to perform successfully at classifying difficult to model actions, such as market orders, iceberg, and limit orders at the BBO, without significant confusion, relative to the comparative machine learning models. One potential reason could be the dichotomised selection process employed by decision tree-based models, with trees split by the most relevant feature without consideration of other features that may be nearly as relevant to the explanation for the traders' action.

Finally, an F-Score is calculated as the harmonic mean of the precision and sensitivity metrics for each model-action pair. Essentially the metric provides a measure of accuracy by weighting the importance of precision over recall by using a beta factor, $\beta > 0$. DNN models attain the highest F-Score, based on a β of 1, for all three actions analysed in this evaluation. F-Scores of 95.6% for LO.B.P0, 99.8% for LO.B.P3 and 80.8% for MO.B explicitly convey the success of DNN models in performing the machine learning task set out in this chapter. The model can accurately predict actions performed by traders based on LOB dynamics more accurately than any other model, for the three classes studied in this section. C50 has the second highest F-Score for all three actions, within 1-2% of DNNs on average, performing more effectively than RFs when predicting MO.B orders.

3.6.2.3 Confusability Analysis

Primitive confusability analysis is performed for each model by computing class-wise sensitivities and confusability metrics, F_{ij}, to determine what actions are predominately confused with the correct true action taken by the trader. Confusion matrix, $\mathbf{M}_{ij}$, allows for the computation of a confusability metric for class i and j, $F_{ij} = \mathbf{M}_{ij} / \sum_{j=1}^{\mathcal{A}} \mathbf{M}_{ij}$, which measures the proportion of samples where class i is incorrectly classified as class j. The results of the confusability analysis is reported in Table 3.6.3.

	DNN		Random Forest		Support Vector Mach.		C5.0 Decision Trees	
Action	**Confusability**	F_{ij} (%)	**Confusability**	F_{ij} (%)	**Confusability**	F_{ij} (%)	**Confusability**	F_{ij} (%)
ICE.A.P0	LO.A.P0	29.03%	LO.A.P0	15.38%	LO.A.P0	27.78%	LO.A.P0	39.39%
ICE.B.P0	LO.B.P0	50.00%	ICE.A.P0	0.00%	LO.A.P0	18.18%	LO.B.P0	34.78%
LO.A.P0	MO.B	1.49%	MO.B	2.26%	MO.B	1.73%	MO.B	1.74%
LO.A.P1	LO.A.P0	0.39%	LO.A.P2	0.99%	LO.A.P2	0.29%	LO.A.P0	0.38%
LO.A.P2	LO.A.P1	0.85%	LO.A.P3	0.98%	LO.A.P3	0.74%	LO.A.P1	0.89%
LO.A.P3	LO.A.P2	0.35%	LO.A.P2	0.42%	LO.A.P2	0.28%	LO.A.P2	0.26%
LO.B.P0	MO.A	1.58%	MO.A	2.44%	MO.A	1.94%	MO.A	1.97%
LO.B.P1	LO.B.P0	0.41%	LO.B.P2	0.98%	LO.B.P2	0.25%	LO.B.P0	0.34%
LO.B.P2	LO.B.P1	0.57%	LO.B.P3	1.17%	LO.B.P3	0.68%	LO.B.P1	0.71%
LO.B.P3	LO.B.P2	0.20%	LO.B.P2	0.38%	LO.B.P2	0.21%	LO.B.P2	0.22%
IOC.A.P0	LO.A.P0	17.97%	LO.A.P0	16.98%	LO.A.P0	16.75%	LO.A.P0	18.50%
IOC.B.P0	LO.B.P0	25.18%	LO.B.P0	27.28%	LO.B.P0	22.02%	LO.B.P0	25.24%
MO.A	LO.B.P0	9.21%	LO.B.P0	5.39%	LO.B.P0	7.56%	LO.B.P0	10.54%
MO.B	LO.A.P0	7.79%	LO.A.P0	4.27%	LO.A.P0	6.75%	LO.A.P0	8.75%

Table 3.6.3 – Confusability analysis of all available actions in the action space separated by the individual machine learning model trained. Confusability refers to the incorrect action most commonly predicted by the model given the correct action represented in the vertical column. Results are aggregated over the 59 evaluation Broker tuple datasets.

Perhaps the most interesting result is the differentiation between what classes confuse the model for each action taken by the trader. For example, limit order at one tick from the best bid, LO.B.P1, are most commonly confused with orders at the BBO, LO.B.P0, for DNN and C50 models, whilst RF and SVM models mainly confuse this action with orders two ticks from the BBO, LO.B.P2.

A second note focusses on the relatively higher confusability scores for the iceberg and IOC orders, for both DNN and C50 models. Whilst the confusability scores are higher than SVMs for all four of the rarer order types, this is not necessarily a negative result. Having a high confusability rating, but low error probability is more conducive to models with good learning properties, as opposed to higher error rates but low confusability scores in the instance of the SVM. Confusability is a good measure of a model's ability to split the dichotomy between two actions based on the LOB features of the dataset, but this improved capability may arise due to the model overfitting the training data and performing poorer when generalising to the test set. This could potentially be the case with SVMs which split classes using a multitude of support vectors in multi-dimensional space.

Finally, another interesting result is the similarity in confusability measures for all four models when analysing the market order types, MO.A and MO.B, and the BBO order types, LO.A.P0 and LO.B.P0.

All four models have the same primary confusion, that is, the opposite side market order when predicting BBO limit orders, and the opposite side BBO order when predicting market orders. This is a critical result that requires further consideration in later chapters. However, it should be noted that these relatively higher confusion scores and patterns of predictive behaviour of the models may arise from non-optimal algorithmic trading strategies that place importance on executing a trade quickly by placing a market order, when the more optimal decision may be to place an order at the BBO. A variety of reasons may motivate this behaviour, including the submission of an order by a client that requires immediate liquidity, or inefficient algorithms that operate on stale LOB state information and features.

3.7 Conclusion

This chapter investigates, develops and trains an optimal Deep Neural Network (DNN) architecture for predicting the behaviour of algorithmic traders given the state of the limit order book (LOB) and broader financial markets. Utilising a proprietary broker-identified dataset of FTSE 100 component securities we can capture and consolidate all messages exchanged between algorithmic traders and UK equity trading venues. A consolidation of all trading activity is performed to offer a panoptic view of the state of the market for a security that is exposed to traders. We next extract a holistic set of 323 features from the consolidated LOB that are assumed to serve as inputs into firms' trading algorithms, with each feature primarily related to a specific algorithmic trading strategy. The neurologically motivated DNN developed in this chapter is tasked with finding complex non-linear patterns between abstract hierarchical features of the LOB and the actions executed by individual traders on the LOB, which are implicitly untangled by the DNNs deep neuronal connections, a complexity difficult for humans to grasp.

The first part of this chapter introduced the feedforward DNN architecture and various design elements, components and configurations commonly used by academics and practitioners to build high capacity models, expressive learnable functions and high-dimensional input feature spaces. Comparative machine learning models are then defined and assessed by their ability to develop abstract representation spaces, a procedure which is performed implicitly within the DNN during learning optimisation. We next explain the FTSE 100 consolidated LOB dataset including the sample securities studied and period of analysis.

Drawing insights from the above research, a comprehensive ablation analysis of the components, design elements and hyper-parameters of the DNN is performed to evaluate their relative utility in solving the machine learning problem of predicting trader actions. An evaluation of the DNNs' non-linear activation functions, architectural configurations, cost function frameworks, initialisation strategies, optimisation algorithms, batch size parameters, regularisation and normalisation procedures, among other DNN elements is performed. The ablation analysis and evaluations provide observations of how the DNN responds to augmented design elements and components allowing for a judgment regarding the optimised model architecture, configurations and hyper-parameters values that most effectively model the mapping function between LOB features and trader actions. We build and train an optimised DNN for each of the 802 unique ISIN-Date-Broker tuple datasets, $\langle \mathcal{S}, \mathcal{D}, \mathcal{B} \rangle$, for firms we define as algorithmic traders, attaining an average training accuracy of 96.4% and generalisation error of 6.9%. These results strongly indicate that the features generated to explain the financial environment provide relevant information in explaining traders' behaviour. To confirm the superiority of the model, we test the DNNs performance against comparative machine learning models finding that the DNN is more suited to mapping the

relationship between LOB features and trader actions given its ability to build complex hierarchical and abstract feature representations. Furthermore, the results perform comparably with the deep learning literature on classification problems characterised by auto-correlated time series of real-world phenomenon.

The manifest technological evolution in market microstructure and algorithmic trading systems over the past decade has challenged researchers, practitioners and regulators, motivating the creation of new tools and methodologies for understanding increasingly complex markets. Evidently, DNNs represent a powerful deep learning methodology open to regulators, with a remit for maintaining fair and effective markets, to comprehensively understand how and why algorithmic traders interact with the LOB. Furthermore, using techniques developed and presented in this thesis, regulators can employ DNNs to ascertain intricate representations in equity market trading data that drive specific algorithmic trading strategies. This enables regulators to perform oversight functions and develop policy prescriptions relevant and necessary for contemporary financial markets.