Hierarchical Modelling for Financial Data

Zihao Zhang
Worcester College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2020

I dedicate this dissertation to my parents Bin Zhang and Zhaohui Li

for their endless love and support.

Acknowledgements

I would like to thank my supervisors Stephen Roberts and Stefan Zohren for their tremendous support and peerless guidance throughout my PhD. It has been a privilege to work with them and they have provided me with aspirational ideas and cultivated me to be proactive and inspirational.

I thank the members of the Machine Learning Research Group: Adam Cobb, Ahsan Alvi, Bernardo Pérez Orozco, Bryan Lim, Daniel Poh, Diego Granziol, Favour Mandanji Nyikosa, Henry Chan, Olga Isupova, Pengfei Zhang, Ivan Kiskin, Siddartha Ghoshal, Vu Nguyen, Wolfgang Fruehwirt, Yincong Zhi and Yunpeng Li. Special thank goes to Xiaowen Dong who has helped me throughout the process, and Binxin Ru and Xingyue Pu with whom I have built great friendship.

I am very grateful to the Oxford-Man Institute of Quantitative Finance who sponsored my study and provided me with computing facilities and all other support. Finally, I thank Kexin Zhong for her kindly understanding and my parents Bin Zhang and Zhaohui Li for their endless love and encouragement.

Statement of Originality

I declare that this thesis is my own work and it is submitted for the degree of Doctor of Philosophy at the University of Oxford. No part of this thesis has been submitted for any other qualification. The main chapters (Chapters 3, 4, 5 and 6) of this thesis were co-authored with my supervisors Stephen Roberts and Stefan Zohren.

Chapter 3: Zhang, Z., Zohren, S., & Roberts, S. (2019). DeepLOB: Deep convolutional neural networks for limit order books. IEEE Transactions on Signal Processing, 67(11), 3001-3012.

Chapter 4: Zhang, Z., Zohren, S., & Roberts, S. (2018). BDLOB: Bayesian deep convolutional neural networks for limit order books. Third workshop on Bayesian Deep Learning (NeurIPS 2018), Canada.

Zhang, Z., Zohren, S., & Roberts, S. (2019). Extending deep learning models for limit order books to quantile regression. Proceedings of Time Series Workshop of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

Chapter 5: Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep Reinforcement Learning for Trading. Journal of Financial Data Science, 2(2), pp.25-40.

Chapter 6: Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep Learning for Portfolio Optimisation. Journal of Financial Data Science.

Abstract

Algorithmic trading has gained great popularity and has been heavily used by institutional investors. Such a trading system involves mathematical and statistical modelling that makes decisions to buy or sell financial securities by following automated trading instructions. In this thesis, we research machine learning based data-driven approaches to construct such trading rules.

We utilise deep learning models to predict stock price movements and develop trading strategies based on the derived signals. Instead of using hand-crafted features, we propose neural network architectures to extract nonlinear features from limit order books data. We expand this work to study Bayesian Deep Learning models and Quantile Regression, obtaining uncertainty estimates on predictive outputs. These uncertainty estimates can be used for position sizing to improve trading performance.

However, a mapping from predictive signals to trade positions is non-trivial to construct. To solve this problem, we research Reinforcement Learning algorithms to optimise a measure of expected utility of final wealth, directly outputting trade positions thus bypassing the explicit forecasting step. Reinforcement Learning algorithms are used to trade individual assets and, subsequently, we extend this end-to-end training framework to portfolio optimisation and directly maximise the Sharpe ratio for portfolios, maximising return per risk.

Contents

List of Figures

Chapter 1

Introduction

Financial trading has been a widely researched topic and a variety of methods have been proposed over several decades. Broadly speaking, we may categorise these into fundamental analysis, technical analysis and algorithmic trading. Arguably, algorithmic trading has gained more interest over the past decade and now accounts for about 75% of trading volume in the United States stock exchanges [26]. The advantages of algorithmic trading are widespread, ranging from strong computing foundations and faster execution to associated risk diversification.

One key component of most trading systems is the creation of a predictive signal that can lead to alpha (excess return) and, to this end, two major classes of work have been proposed and widely adopted: stochastic econometric models and data-driven machine learning approaches [1]. Traditional econometric methods generally assume that the time-series under study are generated from, or can be well approximated by, a parametric stochastic process [23]. However, there is agreement that stock returns behave in more complex ways, typically highly nonlinearly [22, 143], and financial time-series are, in general, non-stationary and residuals are non-Gaussian distributed, which breaks the assumptions of traditional time-series models or linear models that require stationarity and normality of the process residuals.

To model this extra layer of complexity, machine learning techniques are well researched and applied. They are able to capture arbitrary nonlinear relationships with little, or no, prior knowledge regarding the input data [5]. In particular, Deep Learning [61] models have shown great success to study nonlinear and high-dimensional data through a series of hierarchically structured internal representations. The universal approximation theorems [35] state that a neural network can approximately represent any continuous function with a desired precision even just with a single hidden layer. The ability to process nonlinear data and flexible function learning make deep learning models good candidates to study financial time-series. In this thesis, we study network architectures to model high-frequency micro-structure financial data.

However, we notice that traditional trading systems form a two-step optimisation problem where the first step is minimising the error between targets and estimates. Due to the low signal-to-noise ratio of financial data and the dynamic nature of markets, the effectiveness of predictive signals, such as estimated price returns, varies through time. If signals with good predictive power existed, participants would trade this information away as arbitrage opportunities occur, so it is practically difficult to develop signals with consistently good predictive power.

In addition, it is non-trivial to transform predictive signals into actual trade positions. As an example, predictive horizons on price returns are often relatively short (one day or a few days ahead if using daily data). However, large trends can persist for weeks or months with some moderate consolidation periods. We therefore not only need a signal with good predictive power but also a signal that can consistently follow trends and avoid large turnovers. Designing such trade positions often requires us to set thresholds for entering or exiting trades by using predictive signals. The choices of these thresholds are discretionary and we have to fine-tune hyperparameters on validation sets, which often leads to overfitting problems in practice. In this thesis, we showcase how Reinforcement Learning [146] can be used to bypass this forecasting

step entirely and to optimise our trading performance directly.

In summary, there are two big research questions we focus on in this thesis. Firstly, we use Deep Learning models to forecast financial time-series and develop trading strategies based on the derived signals. We propose neural network architectures to extract nonlinear features from raw time-series data and utilise Bayesian Deep Learning models and Quantile Regression to obtain uncertainty estimates on predictive outputs. These uncertainty estimates can be used for position sizing to improve trading performance. Secondly, we consider Reinforcement Learning algorithms [146] as a methodology to optimise some expected utility of final wealth, directly outputting trade positions and bypassing explicit forecasting steps. Reinforcement Learning algorithms are used to formulate trading strategies for individual assets and we extend this end-to-end training framework to portfolio optimisation, directly maximising the portfolio Sharpe ratio in the subsequent section. The major contributions of this thesis are summarised in the following section.

1.1 Contributions

1.1.1 Financial Time Series Forecasting

Financial time-series are highly stochastic, so generic feature extraction methods such as Principal Component Analysis (PCA) [172] are often applied before the modelling process. However, these extraction methods are static pre-processing steps, which are not optimised to maximise the overall objective of our interest. Arguably, one of the key contributions of modern deep learning is the addition of feature extraction and representation as a part of the learned model. The Convolutional Neural Network (CNN) [95] is a prime example, in which information extraction in the form of filter banks is automatically tuned to the utility function that the entire network aims to optimise. We investigate the usage of such deep neural networks to predict price

movements from limit order books (LOBs) data for equities. Our architecture utilises convolutional filters to capture the spatial structure of limit order books as well as Long Short-Term Memory (LSTM) [73] modules to capture longer time dependencies among resulting feature maps. We show that our methods outperform a large collection of baseline methods with feature extraction implicit as part of the training process.

1.1.2 Financial Time Series Forecasting with Uncertainty

Standard deep learning models are typically non-probabilistic and thus only provide a point estimate for the target output. However, financial time-series (such as price returns) are heterogeneous, highly peaked and have fat tails compared to a normal distribution [34]. Point estimation cannot describe the full distribution of returns but understanding such risk exposure is vital for financial applications. In order to obtain well calibrated uncertainty information, we explore Bayesian Deep Neural Networks (BNNs) [16] and Quantile Regression [91] (QR) methods.

Given the complexity of modern neural network models, Bayesian Neural Networks are often trained by approximations to full Bayesian inference such as variational inference [158]. We apply variational dropout [56] to our networks, showcase how uncertainty information relates to position sizing and how trading performance can be improved by avoiding bad trades. Whilst variational dropout has been well established and deployed across many application domains prior to our work [180], we have not found any other literature applying it to financial time-series modelling and position sizing, especially in the context of micro-structure data.

Quantile Regression is another measure of risk exposure, as it predicts the conditional quantiles of the target distribution. We propose a network architecture that can simultaneously estimate multiple return quantiles, from both long and short positions, by training with different quantile loss functions. We show that returns from long and short positions are statistically different due to changing spreads and a separate

modelling approach for each provides us with useful and non-stationary information regarding risk exposure.

1.1.3 Deep Reinforcement Learning for Trading

To overcome the difficulty of mapping predictive signals to trade positions, we develop Reinforcement Learning (RL) algorithms to infer trading strategies. The goal of RL is to learn a policy which maximises expected cumulative rewards via an agent interacting with an uncertain environment. We note that this framework is equivalent to an investor who aims to maximise expected cumulative trade returns. We develop this connection between modern portfolio theory and the RL reward hypothesis and show that they are equivalent if a linear utility function is used.

Our analysis focuses on deep RL algorithms including Deep Q-learning Networks [113, 160], Policy Gradients [171] and Advantage Actor-Critic approaches [112]. Both discrete and continuous action spaces are explored and we utilise features from time-series momentum and technical indicators to form state representations. In addition, volatility scaling [117, 67] is applied to improve reward functions by scaling up trade positions while volatility is low, and vice versa. We test our algorithms on the 50 most liquid futures contracts from 2011 to 2019, and investigate how performance varies across different asset classes including commodities, equity indices, fixed incomes and Foreign Exchange (FX) markets. The experiments show that the proposed algorithm can follow large market trends without changing positions and can also scale down, or hold, through consolidation periods.

1.1.4 Portfolio Optimisation

In our reinforcement learning work, our goal was to optimise some performance metrics of trading individual assets. We now extend this idea to portfolio optimisation and adopt deep learning models to directly optimise the portfolio Sharpe ratio. The

framework we present circumvents the requirements for forecasting expected returns and allows us to directly optimise portfolio weights by updating model parameters. Instead of selecting individual assets, we trade Exchange-Traded Funds (ETFs) of market indices to form a portfolio. Indices of different asset classes show robust correlations and trading them substantially reduces the spectrum of available assets to choose from. We compare our method with a wide range of algorithms and present results indicating that our model obtains the best performance over the testing period, from 2011 to the end of April 2020, including the financial instabilities which occurred during the first quarter of 2020. A sensitivity analysis is included to understand the relevance of input features and we further study the performance of our approach under different cost rates and different risk levels via volatility scaling.

1.2 Thesis Outlines

The remainder of the thesis is structured as follows. In Chapter 2, we introduce background materials, including a literature review and an overview of the fundamentals of neural networks. Chapter 3 describes the foundational concepts of financial time-series forecasting and presents our neural network architecture. Each component of our model is detailed and we discuss why features extracted from deep learning models are superior to static feature engineering methods, such as Principal Component Analysis (PCA).

Chapter 4 extends the work to allow uncertainty to be estimated over the predicted outputs. The first part of this chapter introduces Bayesian statistics and shows how variational dropout can be utilised to obtain appropriate posterior distributions over predictions. In the second half of the chapter we focus on Quantile Regression and design a neural network architecture that can simultaneously estimate multiple return quantiles from both long and short positions.

In Chapter 5, we develop Deep Reinforcement Learning algorithms to design trading strategies for continuous futures contracts. We showcase how trading problems can be formulated as a Markov Decision Process (MDP) [12] where an agent interacts with the environment at discrete time steps. Following Chapter 5, in which strategies are designed for trading individual assets, Chapter 6 studies portfolio optimisation. We present a framework that allows us to directly obtain portfolio weights by maximising the portfolio Sharpe ratio.

Finally, we conclude our findings in Chapter 7 and consider extensions and potential future work.

Chapter 2

Background

This chapter presents the fundamentals underpinning this thesis. We start with a literature review that introduces related work on financial time-series forecasting, deep reinforcement learning for trading and portfolio optimisation. We review classical parametric models that assume definite price dynamics and more recent data-driven machine learning approaches. Subsequently, we present a preliminary overview of deep learning models, describing Multilayer Perceptrons (MLPs) [61], Convolutional Neural Networks (CNNs) [95], Recurrent Neural Networks (RNNs) [144] and Long Short-Term Memory (LSTM) [73].

2.1 Literature Review

Financial Time Series Forecasting: Research on the predictability of stock markets has a long history in the financial literature e.g. [2, 7]. Although opinions differ regarding the efficiency of markets, many widely accepted studies show that financial markets are, to some extent, predictable [52, 17, 104, 105]. Two major classes of work which have been widely used to forecast financial time-series are, broadly speaking, statistical (econometric) parametric models and data-driven machine learning approaches [1].

Traditional econometric statistical methods generally assume that the time-series under study are generated from a parametric process [23]. For example, time-series models, such as autoregressive (AR), moving average (MA) and autoregressive moving average (ARMA) models, are widely used in finance and represent different stochastic processes [75]. An AR breaks down the stochastic process into two parts, where the first part applies a linear function to past values modelling the conditional mean of the process and the second part introduces a random error with zero mean to control for unpredictable deviations. While, these models generally require a stationary process of which the underlying probability distribution does not change over the time. There is, however, agreement that stock returns behave in more complex ways, typically highly nonlinearly [22, 143]. The assumption of strict or covariance stationarity for financial time-series data is not met in practice.

In addition, traditional econometric models focus on middle-to-low frequency financial data such as daily or monthly data. For example, the famous Fama-French Model [44] states that value and small-cap stocks outperform markets on a regular basis, so they include value risk and size risk factors to the market risk model. These models are analysed by using daily data and the construction of factors are built from econometric understanding. However, due to technology development, institutional funds now tend to trade at much higher frequency to amplify profits and to reduce costs. For example, many funds locate their operating units next to exchanges to reduce latency and they process data in milliseconds [25]. Econometric understanding can be hardly drawn from data with such a high frequency and price movements are largely dominated by speculation or short-term supply and demand, undermining the effectiveness of traditional econometric models.

Machine learning techniques, on the other hand, are able to capture arbitrary nonlinear relationships with little, or no, prior knowledge regarding the input data [5]. Recently, there has been a surge of interest in predicting financial time-series

data using machine learning algorithms. These include, but not limited to, support vector machines [88, 83, 84], random forest [8, 129, 140] and multilinear discriminant analysis [153, 151]. In particular, deep neural networks [40, 124, 30, 128, 19] have attracted most attention because of the universal approximation theorems [35] and the ability of processing huge amounts of data. Deep learning models, now accelerated by GPUs, can easily deal with millions of observations and the work of [143] suggests that networks trained with a large amount of data can extract universal features from high-frequency financial data. Such models lead to robust and consistent generalisation performance over a long testing period across many securities. In this thesis, we follow this direction and research state-of-art deep learning models including CNNs and LSTMs applied to high-frequency micro-structure data.

Prior to our work, there have been limited published works that adopt CNNs [27, 42, 155] and LSTMs [41, 122, 143] to analyse limit order book data. In addition, the network architectures used in these papers are often inherited from models designed for image processing problems. We conduct a thorough investigation of the application of deep learning models to financial time-series data and propose a hybrid model that exploits multiple advantages of both CNNs and LSTMs. To the best of our knowledge, there was no prior work that combines CNNs with LSTMs to predict stock price movements and our research was the first extensive study to apply a nested CNN-LSTM model to raw market data. In addition, our network consists of an Inception Module [147] where a single hidden layer contains multiple convolutional layers with different filter sizes. The usage of the Inception Module in this context is novel and appears beneficial in inferring the optimal "decay rates" of the extracted features. Being aware of the difficulty of hyperparameter tuning and the concern of reproducibility in deep learning models, we have open-sourced our code [1] for this network architecture and hope to encourage more research in this field.

[1] https://github.com/zcakhaa/DeepLOB-Deep-Convolutional-Neural-Networks-for-Limit-Order-Books

Financial Time Series Forecasting with Uncertainty: Bayesian inference offers a profound way to express the uncertainty in predictions and decisions. Both Bayesian parametric [92] and nonparametric [72] methods have been extensively applied to time-series data. In particular, Gaussian Processes (GPs) [133, 136] possess the ability to represent multifold functions and have been established as an effective Bayesian technique for predicting financial returns [45, 114, 130, 59].

Interestingly, a GP is equivalent to a single-layer neural network in the limit of infinite network width, and this relationship has been studied in [121, 170, 96]. Both methods have served as powerful nonparametric tools for modelling, but deep learning models, such as CNNs with multiple layers, are able to extract features from raw input data. The chain architecture of a network leads to hierarchically structured internal representations and it turns out that such representations are highly effective in many application domains. For example, in image recognition problems, the work of [177] suggests that shallow layers can detect edges of an image and deep layers can connect derived features to depict high-level representations.

In contract, GPs do not construct "features" like a CNN can. Given the ability of our proposed network to extract features from raw input data and good results obtained from experiments, we focus on deep learning models and study Bayesian Neural Networks (BNNs) to model output uncertainty. BNNs are less developed for financial applications due to inherent difficulties of training and we aim to set up a framework and demonstrate the importance of considering extra risk information. We adopt dropout variational inference [56], as an approximation to BNNs, to predict short term movements in cash equity prices. To the best of our knowledge, this was the first work to apply BNNs to financial micro-structure data.

In addition to BNNs, we study Quantile Regression (QR), as another scalable method, to measure risk exposure. QR has been researched in the literature over many years. For example, the works in [91, 24, 29, 154] have utilised QR for various

financial applications. However, in order to obtain an uncertainty bound on outputs, separate models need to be constructed to estimate multiple quantiles of the target distribution. This process can be computationally demanding for high-frequency micro-structure data so, to solve this limitation, we combine QR with deep learning models and propose a network architecture that can simultaneously estimate multiple return quantiles by training with different quantile loss functions.

Deep Reinforcement Learning for Trading: Reinforcement learning can be generally categorised into model-based and model-free algorithms. Model-based methods use experience to construct a model that describes the transitions and outcomes in an environment. As a result, we attempt to model the environment and to obtain predictions for the next reward and state. With these transition information, we then choose a policy that optimises our rewards. On the contrary, model-free algorithms directly optimise a policy without estimating or using an environment model and one relies on sampling techniques to study the optimal policy through trial and error.

In the literature, model-based methods are less studied for financial markets due to inherent difficulties of modelling the dynamics of financial time-series. We find very limited work on this method and we here introduce two papers that we think are interesting. In the work of [176], the authors propose a model-based deep reinforcement learning for dynamic portfolio optimisation and argue that model-based methods are better than model-free approaches in terms of stability and risk consideration. The work of [164] builds an environment model that emulates stock market and the results suggest that the policy trained within the environment model can be adapted to the real market and still remain profitability. The environment model can be also used as a simulator to study market impact and to avoid real monetary loss. However, apart from these works, research in this domain seems to remain preliminary.

Unlike limited work we find for model-based methods, model-free algorithms have attracted more attention in financial applications. Model-free methods can be further categorised into three directions: critic-only, actor-only and actor-critic approaches [54]. The critic-only approach, mainly Deep Q-network, is the most published method in this field [14, 80, 148, 74, 135] where a state-action value function, Q, is constructed to represent how good a particular action is in a given state. Discrete action spaces are adopted in these papers and an agent is trained to long or short a position with full investment. However, a fully invested position incurs large risk during high volatility periods, exposing one to severe risk when opposite moves occur. Ideally, one would like to scale up or down positions according to current market conditions. Doing this requires one to have large action spaces, however, the critic-only approach suffers from large action spaces as we need to assign a score for each possible action.

The second most common approach is the actor-only approach [116, 115, 100, 38] in which the agent directly optimises the objective function without computing the expected outcome of each action in a state. Because a policy is directly learned, actor-only approaches can be generalised to continuous action spaces. In order to study the distribution of a policy, the Policy Gradient Theorem [146] and Monte Carlo methods [110] are adopted in training and models are updated until the end of each episode where all rewards are collected by the agent. However, we often experience slow learning and many samples are needed to obtain an optimal policy, as individual bad actions will be considered good as long as the total rewards are good, thus taking long times to adjust these actions.

The actor-critic approach forms the third major category of RL methods and aims to solve the above learning problems by updating the policy in real time. The key idea is to alternatively update two models where one, the actor, controls how an agent performs given the current state and the other, the critic, measures how good the chosen action is. However, this approach is the least well studied method in financial

applications with limited research [98, 11, 174]. In this thesis, we study all three methods and comment on techniques that are necessary for training RL models on financial data. In particular, we find that the training of vanilla Deep Q-networks is unstable and results do not converge. Techniques, including fixed Q-targets [160], Double DQN [68] and Dueling DQN [163], are necessary training procedures and we carefully list our choices of hyperparameters. In addition, we introduce volatility scaling to reward functions and this step is important for training the model with multiple securities. More details are included in Chapter 5.

Portfolio Optimisation: There is a considerable literature available on this topic, so we here highlight key concepts that are popular in the industry and in academic study. Modern Portfolio Theory (MPT) or mean-variance analysis [106] is arguably the most popular method and has been well studied and used in many institutional portfolios. It solves a constraint optimisation problem to derive portfolio weights by maximising expected return of a portfolio with a given variance. Despite its popularity, the assumptions made are open to criticism as they are often not well matched to real financial markets. In particular, returns are assumed to follow a Gaussian distribution in MPT, therefore investors only consider the expected return and variance of portfolio returns to make decisions. However, it is widely accepted (see for instance [34, 182]) that returns tend to have fat tails and extreme losses are more likely to occur in practice, leading to severe drawdowns that are not bearable.

Stochastic Portfolio Theory (SPT) was recently proposed in [48, 50]. Unlike other methods, SPT aims to achieve relative arbitrages, selecting portfolios that can outperform a market index with probability one. Such investment strategies have been studied in [46, 47, 138, 173]. However, the number of relative arbitrage strategies remains very small, as the theory does not suggest how to construct such strategies. We can check whether a given strategy is a relative arbitrage, but it is non-trivial to

develop one. In this thesis, we include a particular class of SPT, functionally generated portfolios (FGP) [49], in our experiments. However, the results suggest that this method delivers inferior performance compared to other algorithms and generates large turnovers, making it unprofitable under heavy transaction costs.

Our approach is therefore to directly optimise the portfolio Shape ratio by using deep learning models. The idea of this end-to-end training framework was first considered in [116, 115]. These papers mainly focus on optimising the performance on a single asset, so there is little discussion on how portfolios should be optimised. Furthermore, the testing period in prior works is from 1970 to 1994. Our dataset is up to date and we study the behaviour of our strategy historically and under the current crisis in 2020 due to COVID-19.

2.2 Neural Network Preliminaries

This section summarises the key foundational concepts of neural networks. Neural networks are functions that map inputs to outputs through a series of layers composed of nonlinear computation nodes (neurons). Modern deep networks have multiple such layers and include many variants of arrangements. In this section, we introduce Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM). Together these form the major taxonomies of most deep neural network structures.

2.2.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs), historically also known as feedforward neural networks, represent a canonical neural network model. The goal of such a network is to infer a function f. For regression analysis, the function defines a mapping $y = f(\boldsymbol{x}|\boldsymbol{\theta})$ that estimates the relationship between an input $\boldsymbol{x} \in \mathbb{R}^{N_x}$ and an output (here a

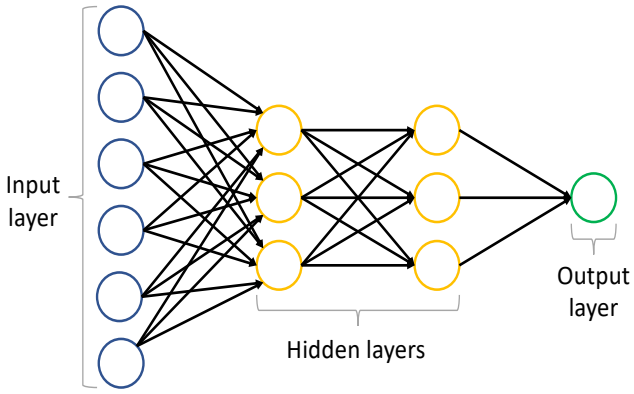

Figure 2.1: A multilayer perceptron with 2 hidden layers in which each hidden layer has 3 neurons.

scalar) $y \in \mathbb{R}$. The vector $\boldsymbol{\theta}$ is the set of all model parameters and we update these parameters to discover the best function approximation.

A typical MLP is organised into a series of layers in a chain structure, with each layer being a function of the layer that precedes it. We can define the first layer as:

$$\boldsymbol{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}), \tag{2.1}$$

where $\boldsymbol{h}^{(1)} \in \mathbb{R}^{N_1}$ represents the first hidden layer with a number of N_1 neurons, and the parameters $\boldsymbol{W}^{(1)} \in \mathbb{R}^{N_1 \times N_x}$ and $\boldsymbol{b}^{(1)} \in \mathbb{R}^{N_1}$ are the neural weights and biases respectively. $g^{(1)}(\cdot)$ is the activation function that allows networks to model nonlinearities, and there are numerous activation functions to choose from. General choices include sigmoid function, hyperbolic tangent function, Rectified Linear Units (ReLU) [119], and Leaky Rectified Linear Units (Leaky-ReLU) [103]. The ReLU function is the most general activation function for deep networks and empirical studies suggest starting with the ReLU though advise a comparison with other activation functions [111]. In practice, the typical choice of an activation function still depends on application domains and requires validation experiments. The same principle of choosing activation functions can be applied to other network structures.

The subsequent hidden layers of a MLP can be similarly defined as:

$$\boldsymbol{h}^{(l)} = g^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}), \tag{2.2}$$

where $\boldsymbol{h}^{(l)} \in \mathbb{R}^{N_l}$ represents the l-th hidden layer with weights $\boldsymbol{W}^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ and biases $\boldsymbol{b}^{(l)} \in \mathbb{R}^{N_l}$. Figure 2.1 shows an example of a MLP with 2 hidden layers. In essence, each hidden layer of the network is typically vector valued and consists of neurons (3 units in this case) that compute a linear combination with the nonlinear activation of the previous layer. The final output is a function of the last hidden layer and we compute objective functions to minimise errors between target outputs and estimates.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are an extension of MLPs that process input data with a grid-like topology. For example, we can treat single variate time-series data as a 1-D grid with regular time intervals [99] and image data as a 2-D grid of pixels [93]. CNNs have been widely applied in many application domains, for example, object tracking [162] and object detection [60].

CNNs have two types of layers: convolutional layers and pooling layers. Convolutional layers are the main components of a CNN and a single convolutional layer consists of a number of filters that extract local spatial relationships of input data. A typical convolutional filter $\boldsymbol{K}$ transforms input data $\boldsymbol{I}$ by utilising convolution operation:

$$\boldsymbol{S}(i,j) = (\boldsymbol{I} * \boldsymbol{K})(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \boldsymbol{I}(i+m, j+n)\boldsymbol{K}(m,n), \tag{2.3}$$

where $\boldsymbol{S}$ is the result matrix and (i,j) represents the indexes of rows and columns of

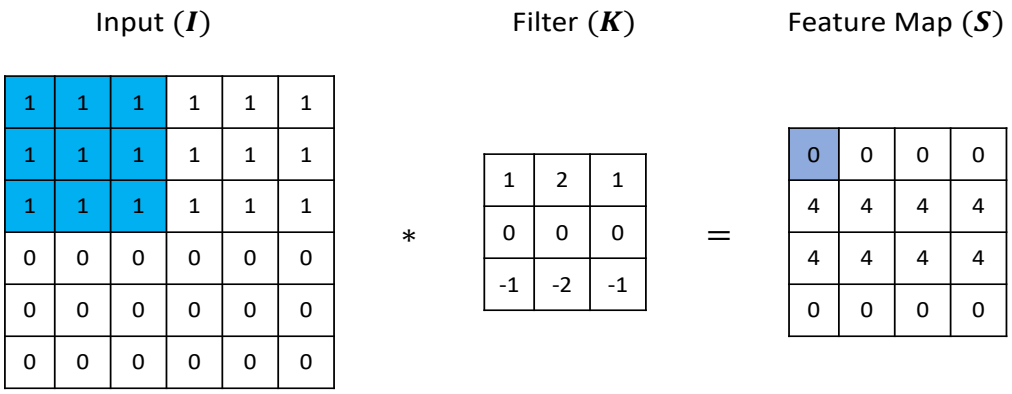

Figure 2.2: An example of filter convolution where $\boldsymbol{S}(0,0) = \sum_{m=0}^{2}\sum_{n=0}^{2}\boldsymbol{I}(0+m, 0+n)\boldsymbol{K}(m,n)$.

$\boldsymbol{S}$. The convolution operation is denoted as $*$ and the filter $\boldsymbol{K}$ has the dimension of $(M \times N)$. To illustrate this process, we present an example in Figure 2.2. A single convolutional layer can contain many such filters in which each filter convolves through input data with a different set of parameters. The resulting matrices from these filters are often referred to as feature maps and, similar to MLPs, we can feed them to another convolutional layer and also pass them through an activation function to model nonlinearities.

A pooling layer also has a grid-like structure and it replaces the output at a certain location of feature maps with a summary statistic of the nearby outputs. For example, the popular max pooling layer [184] takes the maximum value of a certain region of feature maps, and the average pooling calculates the average value of that region. The work of [18] discusses the usage of different kinds of pooling in various situations, but in many applications, the choice of pooling requires domain knowledge and experiments.

In all cases, we apply pooling to make the representation from pooling layers being approximately invariant to small translations of input data. Such invariance can be helpful if we care more about the existence of some features rather than the exact location of them [61]. For example, in image classification problems, we only need to know if an image contains object characteristics of interest instead of knowing

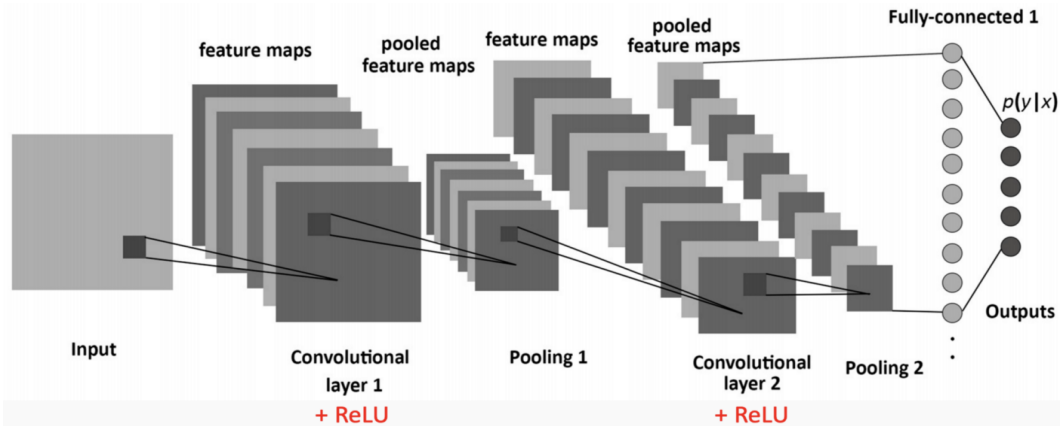

Figure 2.3: An example CNN network with a combination of convolutional, pooling and fully connected layers. Image adapted from [32].

the location of them. However, for time-series data, the location or timing of some features is critical and the usage of pooling layers requires some caution. We discuss this further in Chapter 3 where we describe our network architecture.

Finally, we can combine convolutional and pooling layers to construct a convolutional network. An example of a typical CNN used in image classification is presented in Figure 2.3. Other famous network architectures include "AlexNet" [93], "VGGNet" [142] and etc. In this thesis, we show that a careful design of network architecture can also better model high-frequency limit order books data.

2.2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are often used for sequence modelling and have been particularly well studied for natural language processing problems [175]. Many RNNs have been proposed for time-series applications since time-series data possess sequential characteristics and make RNNs a natural choice.

We can define a single input for any time-series data as $\boldsymbol{x}_{1:T}$ where $\boldsymbol{x}_t$ represents the features at time t and T is the length of the lookback window of the input. For MLPs, we need to first flatten $\boldsymbol{x}_{1:T}$ and feed it to subsequent hidden layers. However, doing this breaks the time dependences and treats features at different time stamps

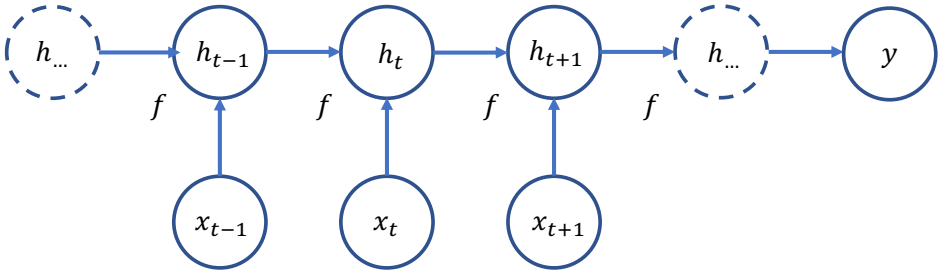

Figure 2.4: A single layer recurrent network that processes information from the input and the past hidden state.

independently. RNNs have a hidden state that acts as an memory buffer, summarising past information and recursively updating the state with new observations at each time step. This process naturally propagates the information from past to present and extracts temporal relationships from the input data. We can write the hidden state of a RNN as:

$$\begin{aligned} \boldsymbol{h}_t &= f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t | \boldsymbol{W}, \boldsymbol{b}), \\ &= g(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}), \end{aligned} \tag{2.4}$$

where $\boldsymbol{W}_h \in \mathbb{R}^{N_h \times N_h}$, $\boldsymbol{W}_x \in \mathbb{R}^{N_h \times N_x}$, $\boldsymbol{b} \in \mathbb{R}^{N_h}$ are the linear weights and biases of a hidden state and $g(\cdot)$ is an activation function. N_h is the number of units in the hidden states and we have N_x features at any time t for a single input. To illustrate this process, an example RNN is presented in Figure 2.4.

However, due to their recursive structures, RNNs suffer from vanishing or exploding gradient problems [13] when computing the derivatives of the hidden states with respect to the parameters. This problem makes gradients difficult to back-propagate and causes an unstable training process.

Long Short-Term Memory networks (LSTMs) were proposed to solve above limitations by operating a gating mechanism that efficiently controls the propagation of past information [73]. Similar to RNNs, a LSTM also updates its hidden state recursively, but it has an additional cell state $\boldsymbol{c}_t$ coupled with a series of gates at each hidden

state:

$$\begin{aligned}
\text{Input gate:} \quad & \boldsymbol{i}_t = \sigma(\boldsymbol{W}_{i,h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{i,x}\boldsymbol{x}_t + \boldsymbol{b}_i), \\
& \text{where } \boldsymbol{W}_{i,h} \in \mathbb{R}^{N_h \times N_h}, \boldsymbol{W}_{i,x} \in \mathbb{R}^{N_h \times N_x} \text{ and } \boldsymbol{b}_i \in \mathbb{R}^{N_h}, \\
\text{Output gate:} \quad & \boldsymbol{o}_t = \sigma(\boldsymbol{W}_{o,h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{o,x}\boldsymbol{x}_t + \boldsymbol{b}_o), \\
& \text{where } \boldsymbol{W}_{o,h} \in \mathbb{R}^{N_h \times N_h}, \boldsymbol{W}_{o,x} \in \mathbb{R}^{N_h \times N_x} \text{ and } \boldsymbol{b}_o \in \mathbb{R}^{N_h}, \\
\text{Forget gate:} \quad & \boldsymbol{f}_t = \sigma(\boldsymbol{W}_{f,h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{f,x}\boldsymbol{x}_t + \boldsymbol{b}_f), \\
& \text{where } \boldsymbol{W}_{f,h} \in \mathbb{R}^{N_h \times N_h}, \boldsymbol{W}_{f,x} \in \mathbb{R}^{N_h \times N_x} \text{ and } \boldsymbol{b}_f \in \mathbb{R}^{N_h}.
\end{aligned} \tag{2.5}$$

Here $\boldsymbol{h}_{t-1}$ is the hidden state of a LSTM at time $t-1$ and $\sigma(\cdot)$ represents the sigmoid activation function. We use $\boldsymbol{W}$ and $\boldsymbol{b}$ to represent weights and biases at different gate operations. Subsequently, the current cell state and hidden state can be written as:

$$\begin{aligned}
\text{Cell state:} \quad & \boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_{c,h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{c,x}\boldsymbol{x}_t + \boldsymbol{b}_c), \\
\text{Hidden state: :} \quad & \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned} \tag{2.6}$$

where $\boldsymbol{W}_{c,h} \in \mathbb{R}^{N_h \times N_h}$, $\boldsymbol{W}_{c,x} \in \mathbb{R}^{N_h \times N_x}$, $\boldsymbol{b}_c \in \mathbb{R}^{N_h}$, $\odot$ is the element-wise product and $\tanh(\cdot)$ is the hyperbolic tangent activation function. The hidden state $\boldsymbol{h}_t$ summarises the information from past states and current observations just like a RNN does, but this gating mechanism efficiently addresses the vanishing gradient problem. In this thesis, we apply LSTMs to various financial applications and discuss their effectiveness in modelling financial time-series data.

Chapter 3

Financial Time Series Forecasting for Limit Order Book Data [1]

3.1 Introduction

One of the key components of a trading system is a predictive signal that indicates future price movements. In this chapter, we develop a large-scale deep learning model to predict price movements from limit order book (LOB) [127] data of cash equities. In today's competitive financial world more than half of the markets use electronic LOBs to record trades [137]. Unlike traditional quote-driven marketplaces, where traders can only buy or sell an asset at one of the prices made publicly by market makers, traders now can directly view all resting limit orders[2] in the limit order book of an exchange. Because limit orders are arranged into different levels based on their submitted prices, the evolution in time of a LOB represents a multi-dimensional problem with elements

[1]The content of this chapter has been published in [181] and the codes are available at https://github.com/zcakhaa/DeepLOB-Deep-Convolutional-Neural-Networks-for-Limit-Order-Books.

[2]Limit orders are orders that do not match immediately upon submission and are also called passive orders. This is opposed to orders that match immediately, so-called aggressive orders, such as a market order. A LOB is simply a record of all resting/outstanding limit orders at a given point in time.

representing the numerous prices and order volumes/sizes at multiple levels of the LOB on both the buy and sell sides.

A LOB is a complex dynamic environment with high dimensionality, inducing modelling complications that make traditional methods difficult to cope with. Mathematical models such as the vector autoregressive model (VAR) [186] or the autoregressive integrated moving average model (ARIMA) [3] often rely on handcrafted features of the data. However, given the billions of electronic market quotes that are generated everyday, it is natural to employ more modern data-driven machine learning techniques to extract such features.

We design a novel deep neural network architecture that incorporates both convolutional layers as well as LSTM units to predict future stock price movements in large-scale high-frequency LOB data. In order to avoid the limitations of handcrafted features, we use a so-called Inception Module [147] to wrap convolutional and pooling layers together. The Inception Module helps to infer local interactions over different time horizons. The resulting feature maps are then passed into LSTM units which can capture dynamic temporal behaviours.

We test our model on a publicly available LOB dataset, known as FI-2010 [124], and our method remarkably outperforms all existing state-of-the-art algorithms. However, the FI-2010 dataset is only made up of 10 consecutive days of down-sampled pre-normalised data from a less liquid market. To ensure the generalisation ability of our model, we further test it by using one year order book data for 5 stocks from the London Stock Exchange (LSE). Our model delivers robust out-of-sample prediction accuracy across stocks over a test period of three months.

As well as presenting results on out-of-sample data (in a timing sense) from stocks used to form the training set, we also test our model on out-of-sample (in both timing and data stream sense) stocks that are not part of the training set. Interestingly, we still obtain good results over the whole testing period. We believe this observation

shows not only that the proposed model is able to extract robust features from order books, but also indicates the existence of universal features in the order book that modulate stock demand and price.

To show the practicability of our model we use it in a simple trading simulation. We focus on sufficiently liquid stocks so that slippage and market impact are small. Indeed, these stocks are generally harder to predict than less liquid ones. Since our trading simulation is mainly meant as a method of comparison between models, we assume that trading takes place at mid-price[3] and compare gross profits before fees. The former assumption is equivalent to assuming that one side of the trade may be entered into passively and the latter assumes that different models trade similar volumes and would thus be subject to similar fees. Under these simplifications, our model delivers significantly positive returns with a relatively small risk.

Although our network achieves good performance, a complex "black box" system, such as a deep neural network, has limited use for financial applications without some understanding of the rationale behind the model predictions. Here we exploit the model-agnostic LIME method [134] to highlight highly relevant components in the order book to gain a better understanding between our predictions and model inputs. Reassuringly, these conform to sensible (though arguably unusual) patterns of activity in both price and volume within the order book.

3.2 Data, Normalisation and Labelling

3.2.1 Limit Order Books

We first introduce some basic definitions of LOBs. For classical references on market microstructure the reader is referred to [65, 125] and for a short review on LOBs in particular we refer to [63]. Here we follow the conventions of [63]. A LOB has two

[3]The average of the best buy and best sell prices in the market at the time.

types of orders: bid orders and ask orders. A bid (ask) order is an order to buy (sell) an asset at or below (above) a specified price. The bid orders have prices $\mathbf{P}_b(t)$ and sizes/volumes $\mathbf{V}_b(t)$, and the ask orders have prices $\mathbf{P}_a(t)$ and sizes/volumes $\mathbf{V}_a(t)$. Both $\mathbf{P}(t)$ and $\mathbf{V}(t)$ are vectors representing values at different price levels of an asset.

Figure 3.1 illustrates the above concepts. The upper plot shows a slice of a LOB at time t. Each square in the plot represents an order of nominal size 1. This is done for simplicity, in reality different orders can be of different sizes. The blue bars represent bid orders and the yellow bars represent ask orders. Orders are sorted into different levels based on their submitted prices, where L1 represents the first level and so on. Each level contains two values: price and volume. On the bid side, $\mathbf{P}_b(t)$ and $\mathbf{V}_b(t)$ are 4-vectors in this example. We use $p_b^{(1)}(t)$ to denote the highest available price for a buying order (first bid level). Similarly, $p_a^{(1)}(t)$ is the lowest available selling order (first ask level). The bottom plot shows the action of an incoming market order to buy 5 shares at time $t+1$. As a result, the entire first and second ask-levels are executed against that order and $p_a^{(1)}(t+1)$ moved to 20.8 from 20.6 at time t.

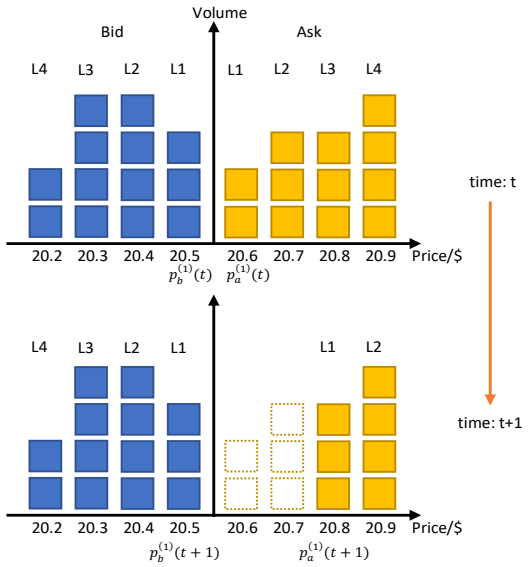

Figure 3.1: A slice of the LOB at time t and $t+1$. L1 represents the respective first level, L2 the second, and etc. $p_a^{(1)}(t)$ is the lowest ask price (best ask) and $p_b^{(1)}(t)$ is the highest bid price (best bid) at time t.

3.2.2 Input Data

We test our model on two datasets: the FI-2010 dataset [124] and one year length of limit order book data from the London Stock Exchange (LSE). The FI-2010 dataset [124] is the first publicly available benchmark dataset of high-frequency limit order data and extracted time-series data for five stocks from the Nasdaq Nordic stock market for a time period of 10 consecutive days. Many earlier algorithms are tested on this dataset and we use it to establish a fair comparison to other algorithms. However, 10 days is an insufficient amount of data to fully test the robustness and generalisation ability of an algorithm as the problem of overfitting to backtest data is severe and we often expect a signal to be consistent over a few months.

To address the above concerns, we train and test our model on limit order book data of one year length for Lloyds Bank, Barclays, Tesco, BT and Vodafone. These five instruments are among the most liquid stocks listed on the London Stock Exchange. It is generally more difficult to train models on more liquid stocks, but at the same time, those instruments are easier to trade without price impact so making the simple trading simulation used to assess performance more realistic. The data includes all LOB updates for the above names. It spans all trading days from 3rd January 2017 to 24th December 2017 and we restrict it to the interval between 08:30:00 and 16:00:00, so that only normal trading activities occur and no auction takes place. Each state of the LOB contains 10 levels on each side and each level contains information on both price and volume. Therefore, we have a total of 40 features at each timestamp.

Overall, our LSE dataset is made up of 12 months, and has more than 134 million samples. On average, there are 150,000 events per day per stock. The events are irregularly spaced in time. The time interval, $\Delta_{k,k+1}$, between two events can vary considerably, from a fraction of a second to seconds and $\Delta_{k,k+1}$ is on average 0.192 seconds in the dataset. We take the first 6 months as training data, the next 3 months as validation data and the last 3 months as test data. In the context of high-frequency

data, 3 months test data corresponds to millions of observations and therefore provides sufficient scope for testing model performance and estimating model accuracy.

3.2.3 Data Normalisation and Labelling

The FI-2010 dataset [124] provides 3 different normalised dataset: z-score, min-max and decimal precision normalisation. We use data normalised by z-score without any emendation and found subtle differences when using the other two normalisation schemes. For the LSE dataset, we again use standardisation (z-score) to normalise our data, but use the mean and standard deviation of the previous 5 days' data to normalise the current day's data (with a separate normalisation for each instrument). We want to emphasize the importance of normalisation because the performance of machine learning algorithms often depends it. As financial time-series usually experiences regime shifts, using a static normalisation scheme is not appropriate for a dataset of one year length. The above method is dynamic and the normalised data often falls into a reasonable range. We use the 100 most recent states of the LOB as an input to our model for both datasets. Specifically, a single input is defined as $X = [x_1, x_2, \cdots, x_t, \cdots, x_{100}]^T \in \mathbb{R}^{100\times 40}$, where $x_t = [p_a^{(i)}(t), v_a^{(i)}(t), p_b^{(i)}(t), v_b^{(i)}(t)]_{i=1}^{n=10}$. $p^{(i)}$ and $v^{(i)}$ denote the price and volume size at i-th level of a limit order book.

After normalising the limit order data, we use the mid-price:

$$p_t = \frac{p_a^{(1)}(t) + p_b^{(1)}(t)}{2}, \tag{3.1}$$

to create labels that represent the direction of price changes. Although no order can transact exactly at the mid-price, it expresses a general market value for an asset and it is frequently quoted when we want a single number to represent an asset price.

Because financial data is highly stochastic, if we simply compare p_t and p_{t+k} to decide the price movement, the resulting label set will be noisy. In the works of [124]

and [155], two smoothing labelling methods are introduced. We briefly recall the two methods here. First, let m_- denote the mean of the previous k mid-prices and m_+ denote the mean of the next k mid-prices:

$$\begin{aligned} m_-(t) &= \frac{1}{k}\sum_{i=0}^{k} p_{t-i}, \\ m_+(t) &= \frac{1}{k}\sum_{i=1}^{k} p_{t+i}, \end{aligned} \tag{3.2}$$

where p_t is the mid-price defined in Equation (3.1) and k is the prediction horizon. Both methods use the percentage change (l_t) of the mid-price to decide directions. We can now define:

$$l_t = \frac{m_+(t) - p_t}{p_t}, \tag{3.3}$$

$$l_t = \frac{m_+(t) - m_-(t)}{m_-(t)}. \tag{3.4}$$

Both are methods to define the direction of price movement at time t, where the former, Equation 3.3, was used in [124] and the latter, Equation 3.4, in [155]. The labels are then decided based on a threshold (α) for the percentage change (l_t). If $l_t > \alpha$ or $l_t < -\alpha$, we define it as up ($+1$) or down (-1). For anything else, we consider it as stationary (0). Figure 3.2 provides a graphical illustration of two labelling methods on the same threshold (α) and the same prediction horizon (k). All the labels classified as down (-1) are shown as red areas and up ($+1$) as green areas. The uncoloured (white) regions correspond to stationary (0) labels.

The FI-2010 dataset [124] adopts the method in Equation 3.3 and we directly use their labels for fair comparisons to other methods. However, the produced labels are less consistent as shown on the top of Figure 3.2 because this method fits closer to real prices as smoothing is only applied to future prices. This is essentially detrimental for designing trading algorithms as signals are not consistent here leading to many

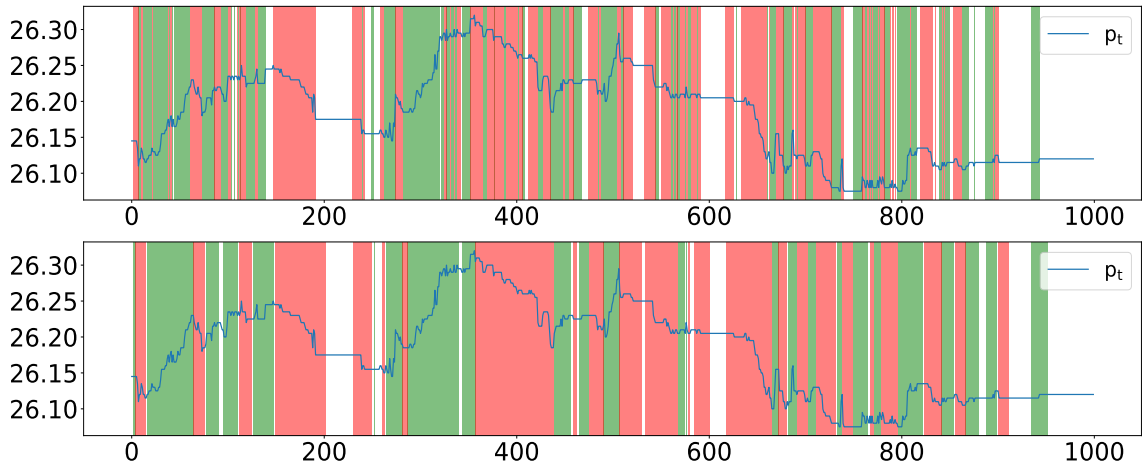

Figure 3.2: An example of two smoothed labelling methods based on the same threshold (α) and same prediction horizon (k). Green shading represents a +1 signal and red a -1. **Top:** [124]'s method and **Bottom:** [155]'s method.

redundant trading actions thus incurring larger transaction costs. We experimented with this approach in [124] on our data from the London Stock Exchange and found the resulting labels are rather stochastic, therefore we adopt the method in Equation 3.4 for our LSE dataset to produce more consistent signals.

The FI-2010 dataset studies four different prediction horizons, where k equals to $10, 20, 50$, and 100 tick steps into the future, corresponding to about $2, 4, 10$, and 20 seconds in clock time. To follow up with this convention, we choose prediction horizons ($k = 20, 50$, and 100) for our LSE dataset, but we do not include $k = 10$ because we find that this prediction horizon is too short and makes little difference compared to $k = 20$. Price changes are very limited within such a short horizon, and a relatively longer prediction horizon is more meaningful for indicating future market movements. In this case, k equalling to 50 or 100 is more practically useful and signals trained with such prediction horizons are applied in applications such as market making and trade execution which have short holding periods.

3.3 Model Architecture

3.3.1 Overview

We here detail our network architecture, which comprises three main building blocks: standard convolutional layers, an Inception Module and a LSTM layer, as shown in Figure 3.3. The main idea of using CNNs and Inception Modules is to automate the process of feature extraction as it is often difficult in financial applications since financial data is notoriously noisy with a low signal-to-noise ratio. We only require the history of LOB prices and sizes as inputs to our algorithm. Weights are learned during inference and features, learned from a large training set, are data-adaptive, removing the above constraints. A LSTM layer is then used to capture additional time dependencies among the resulting feature maps.

3.3.2 Details of Each Component

Convolutional Layer: Recent development of electronic trading algorithms often submit and cancel vast numbers of limit orders over short periods of time as part of their trading strategies [70]. These actions often take place deep in a LOB and it is seen [63] that more than 90% of such orders end in cancellation rather than matching, therefore practitioners consider levels further away from best bid and ask levels to be less useful in any LOB. In addition, the work of [21] suggests that the best ask and best bid (L1-Ask and L1-Bid) contribute most to the price discovery and the contribution of all other levels is considerably less, estimated at as little as 20%. As a result, it would be otiose to feed all level information to a neural network as levels deep in a LOB are less useful and can potentially even be misleading. Naturally, we can smooth these signals by summarising the information contained in deeper levels. We note that convolution filters used in any CNN architecture are discrete convolutions, or finite impulse response (FIR) filters, from the viewpoint of signal processing [126].

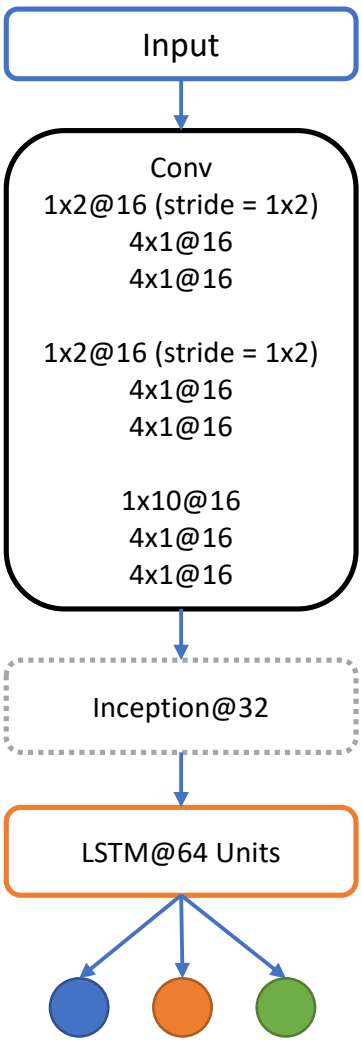

Figure 3.3: Model architecture schematic. Here (1×2)@16 represents a convolutional layer with 16 filters of size (1×2). '1' convolves through time indices and '2' convolves different limit order book levels.

FIR filters are popular smoothing techniques for denoising target signals and they are simple to implement and work with. We can write any FIR filter in the following form:

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k), \tag{3.5}$$

where the output signal $y(n)$ at any time is a weighted sum of a finite number of past values of the input signal $x(n)$. The filter order is denoted as M and b_k is the filter coefficient. In a convolutional neural network, the coefficients of the filter kernel are

not obtained via a statistical objective from traditional signal filtration theory, but are left as degrees of freedom which the network infers so as to extremise its value function at output.

The details of the first convolutional layer inevitably need some consideration. As convolutional layers operate a small kernel to "scan" through input data, the layout of limit order book information is vital. Recall that we take the most 100 recent updates of an order book to form a single input and there are 40 features per time stamp, so the size of a single input is (100×40). We organise the 40 features as following:

$$\{p_a^{(i)}(t), v_a^{(i)}(t), p_b^{(i)}(t), v_b^{(i)}(t)\}_{i=1}^{n=10}, \tag{3.6}$$

where i denotes the i-th level of a limit order book. The size of our first convolutional filter is (1×2) with stride of (1×2). The first layer essentially summarises information between price and volume $\{p^{(i)}, v^{(i)}\}$ at each order book level. The usage of stride is necessary here as an important property of convolutional layers is parameter sharing. This property is attractive as fewer parameters are estimated, largely avoiding overfitting problems. However, without strides, we would apply same parameters to $\{p^{(i)}, v^{(i)}\}$ and $\{v^{(i)}, p^{(i+1)}\}$. In other words, $p^{(i)}$ and $v^{(i)}$ would share same parameters because the kernel filter moves by one step, which is obviously wrong as price and volume form different dynamic behaviours.

We then add another two convolutional layers, each with a filter size of (4×1), to the first convolutional layer and these two layers are used to extract time-dependent features as we only convolve through the time index of our input. In other words, the first convolutional layer is used to extract dependencies across different feature types and the subsequent two layers are used to extract time-dependent information. We can consider these three layers as a single convolutional block and the idea of this block originates from image recognition problems [93] where empirical evidence

suggests that multiple layers with small convolutional filter lead to better performance. Our experience supports this observation and we, in total, have three such blocks in our model as shown in Figure 3.3.

Because the first layer only captures information at each order book level, we would expect representative features to be extracted when integrating information across multiple order book levels. We can do this by utilising another convolutional layer with filter size (1×2) and stride (1×2). The resulting feature maps actually form the micro-price defined by [58]:

$$\begin{aligned} p^{\text{micro price}} &= I p_a^{(1)} + (1 - I) p_b^{(1)}, \\ I &= \frac{v_b^{(1)}}{v_a^{(1)} + v_b^{(1)}}. \end{aligned} \tag{3.7}$$

The weight I is called the imbalance. The micro-price is an important indicator as it considers volumes on both bid and ask sides, and the imbalance between bid and ask size is a very strong indicator of the next price move. This feature of imbalance has been reported by a variety of researchers [123, 6, 20, 66, 102]. Unlike the micro-price where only the first order book level is considered, we utilise convolutions to form micro-prices for all levels of a LOB so the resulting feature maps are of size $(100, 10)$ after two layers with strides. Finally, we integrate all information by using a large filter of size (1×10) and the dimension of our feature maps before the Inception Module is $(100, 1)$. Also, each of these two layers is followed by two convolutional layers with filter size (4×1), which forms the convolutional block introduced above.

We apply zero padding to every convolutional layer so the time dimension of our inputs does not change and Leaky Rectifying Linear Units (Leaky-ReLU) [103] are used as activation functions. The hyperparameter (the small gradient when the unit is not active) of the Leaky-ReLU is set to 0.01, evaluated by grid search on the validation set.

We do not use any pooling layer except in the Inception Module. Although pooling layers help us find representations invariant to small translations of the input, the smoothing nature of pooling can cause underfitting. Common pooling layers are designed for image processing tasks, and they are most powerful when we only care if certain features exist in the inputs instead of where they exist [62]. Time-series data has different characteristics from images and the location of representative features is important. Our experiences show that pooling layers, at least, cause underfitting problems to the LOB data. However, we think that pooling is important and new pooling methods should be designed to process time-series data as it is a promising solution to extract invariant features.

Inception Module: We note that all filters of a standard convolutional layer have fixed size. If, for example, we employ filters of size (4×1), we capture local interactions amongst data over four time steps. However, we can capture dynamic behaviours over multiple timescales by using Inception Modules to wrap several convolutions together. We find that this offers a performance improvement to the resultant model.

The idea of the Inception Module can be also considered as using different moving averages in technical analysis. Practitioners often use moving averages with different decay weights to observe time-series momentum [117]. If a large decay weight is adopted, we get a smoother time-series that well represents the long-term trend, but we could miss small variations that are important in high-frequency data. In practice, it is a daunting task to set the right decay weights. Instead, we can use Inception Modules and the weights are then learned during back-propagation.

In our case, we split the input into a small set of lower-dimensional representations by using 1×1 convolutions, transform the representations by a set of filters, here 3×1 and 5×1, and then merge the outputs. A max-pooling layer is used inside the Inception Module, with stride 1 and zero padding. "Inception@32" represents

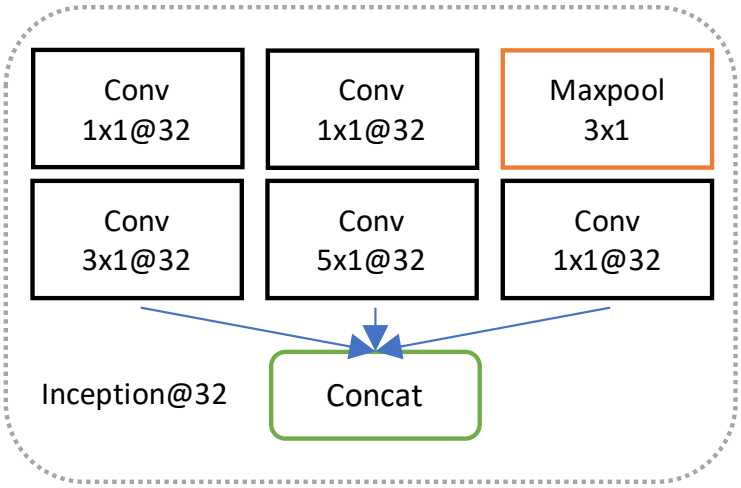

Figure 3.4: The Inception Module used in the model. For example, 3×1@32 represents a convolutional layer with 32 filters of size (3×1).

one module and indicates all convolutional layers have 32 filters in this module, and the approach is depicted schematically in Figure 3.4. The 1×1 convolutions form the Network-in-Network approach proposed in [101]. Instead of applying a simple convolution to our data, the Network-in-Network method uses a small neural network to capture nonlinear properties of our data. We find this method to be effective and it gives us an improvement on prediction accuracy.

LSTM Module and Output: After the Inception Module, the dimension of our feature maps is $(100, 96)$, meaning that we have 96 feature maps, each with 100 time steps. In general, fully connected layers are attached before the output layer. However, we have to flatten all inputs to the fully connected layer which breaks time-dependency for our feature maps. Also, due to the usage of Inception Module in our work, we have a large number of features at end. Just using one fully connected layer with 64 units would result in more than 630,000 parameters to be estimated, not to mention multiple layers.

In order to capture temporal relationships that exist in the extracted features, we replace the fully connected layers with LSTM units. The activation of a LSTM

unit is fed back to itself and the memory of past activations is kept with a separate set of weights, so the temporal dynamics of our features can be modelled. We use a single LSTM layer with 64 units in our work, resulting in about 60,000 parameters, leading to 10 times fewer parameters to be estimated. The dimension after the LSTM layer is 64 and we connect these units to the final output layer with a linear operator and use a softmax activation function, hence the final output elements represent the probability of each price movement class at each time step.

3.4 Experimental Results

3.4.1 Experiments Settings

We apply the same architecture to all our experiments in this section and the proposed model is denoted as DeepLOB. We learn the parameters by minimising the categorical cross-entropy loss. The Adaptive Moment Estimation algorithm, ADAM [89], is utilised and we set the parameter "epsilon" to 1 and the learning rate to 0.01. The learning is stopped when validation accuracy does not improve for 20 more epochs. This is about 100 epochs for the FI-2010 dataset and 40 epochs for the LSE dataset. A mini-batch of size 32 is used and we choose a small mini-batch size due to the findings in [85] in which they suggest that large-batch methods tend to converge to narrow deep minima of the training function, but small-batch methods consistently converge to shallow broad minima.

The label parameters (α) are fixed in the FI-2010 dataset and we directly use their labels for fair comparisons. For the LSE dataset, we choose α for each stock to have a balanced training set and the same α is then applied to the validating and testing sets. Table 3.1 lists all choices for the label parameters and we present the proportion of different classes in Figure 3.5. The labels are mostly balanced across training, validating and testing sets.

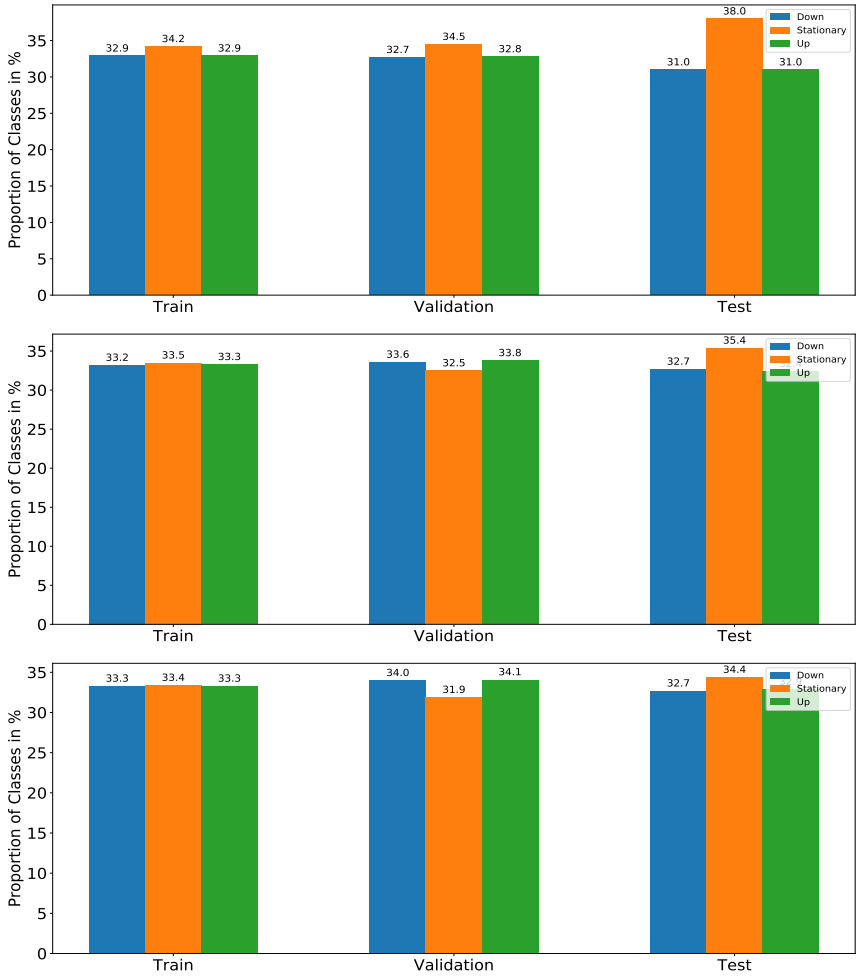

Figure 3.5: Label class balancing for train, validation and test sets for different prediction horizons (k) on the LSE dataset. **Top:** $k = 20$; **Middle:** $k = 50$; **Bottom:** $k = 100$.

Table 3.1: Label Parameters (α) for Different Prediction Horizons and Stocks (units in 10^{-4}).

	LLOY	BARC	TSCO	BT	VOD
k = 20	0.25	0.26	0.45	0.36	0.05
k = 50	0.50	0.55	0.90	0.68	0.35
k = 100	0.70	0.85	1.25	0.95	0.63
	GLEN	HSBC	CNA	BP	ITV
k = 20	0.37	0.01	0.01	0.17	0.25
k = 50	0.70	0.27	0.70	0.33	1.00
k = 100	1.00	0.45	1.20	0.50	1.50

3.4.2 Experiments on the FI-2010 Dataset

There are two experimental setups using the FI-2010 dataset. Following the convention of [128], we denote them as Setup 1 and Setup 2. Setup 1 splits the dataset into 9 folds based on a day basis (a standard anchored forward split). In the i-th fold, we train our model on the first i days and test it on the $(i+1)$-th day where $i = 1, \cdots, 9$. The second setting, Setup 2, originates from the works [155, 156, 157, 152] in which deep network architectures were evaluated. As deep learning techniques often require a large amount of data to calibrate weights, the first 7 days are used as the train data and the last 3 days are used as the test data in this setup. Note that the prediction horizons are different in two setups and we evaluate our model in both setups here.

Table 3.2 shows the results of our model compared to other methods in Setup 1. Performance is measured by calculating the mean accuracy, recall, precision, and F1 score over all folds. As the FI-2010 dataset is not well balanced, [124] suggests to focus on F1 score performance as fair comparisons. We have compared our model to all existing methods including a naive baseline model (Naive) [4], Ridge Regression (RR) [124], Single-Layer-Feedforward Network (SLFN) [124], Linear Discriminant Analysis (LDA) [153], Multilinear Discriminant Analysis (MDA) [153], Multilinear Time-series Regression (MTR) [153], Weighted Multilinear Time-series Regression (WMTR) [153], Multilinear Class-specific Discriminant Analysis (MCSDA) [151], Bag-of-Feature (BoF) [128], Neural Bag-of-Feature (N-BoF) [128], and Attention-augmented-Bilinear-Network with one hidden layer (B(TABL)) and two hidden layers (C(TABL)) [152]. More methods such as PCA and Autoencoder (AE) are actually tested in their works but, for simplicity, we only report their best results and our model achieves better performance.

However, the Setup 1 is not ideal for training deep learning models as we mentioned

[4] A naive model takes the current observation as the future estimate and the model is based purely on temporal relationship.

that deep network often requires a large amount of data to calibrate weights. This anchored forward setup leads to only one or two days' training data for the first few folds and we observe worse performance in the first few days. As training data grows, we observe remarkably better results as shown in Table 3.3 which shows the results of our network compared to other methods in Setup 2. In particular, the important difference between our model and CNN-I [155] and CNN-II [157] is due to network architecture and we can see huge improvements on performance here.

In order to understand how the components of DeepLOB contribute to prediction performance, we present the results of DeepLOB without both the Inception Module and LSTM layer (CNN), DeepLOB without the Inception Module (CNN+LSTM) and DeepLOB without the LSTM layer (CNN+INCEP) in Table 3.3. We can observe that DeepLOB with just the convolutional component still delivers decent results, but adding the Inception Module and LSTM improves prediction performances. Also, CNN+LSTM performs better than CNN+INCEP, which could suggest that the LSTM layer is more effective than the Inception Module. This might be due to the ability of the LSTM to extract longer time-dependencies present in the input data, so the dynamics of limit order book can be better modelled.

For both setups, we observe that the best performances are obtained when the prediction horizon (k) equals to 10. This meets our expectation as, in general, the shorter the prediction horizon the better the model performance should be. However, we see that the model delivers better results when $k = 100$ compared to $k = 50$ for Setup 1 and $k = 50$ compared to $k = 20$ for Setup 2. This could be due to the small amounts of testing data in the FI-2010 dataset as it only has 10 days data in total. A longer testing period is necessary to verify how performance varies against different prediction horizons and this is one of the reasons why we include the LSE dataset in this thesis.

Table 3.2: Setup 1: Experiment Results for the FI-2010 Dataset.

Model	**Accuracy %**	**Precision %**	**Recall %**	**F1 %**
Prediction Horizon k = 10				
Naive	67.70	68.00	68.00	68.00
RR [124]	48.00	41.80	43.50	41.00
SLFN [124]	64.30	51.20	36.60	32.70
LDA [153]	63.83	37.93	45.80	36.28
MDA [153]	71.92	44.21	60.07	46.06
MCSDA [151]	83.66	46.11	48.00	46.72
MTR [153]	86.08	51.68	40.81	40.14
WMTR [153]	81.89	46.25	51.29	47.87
BoF [128]	57.59	39.26	51.44	36.28
N-BoF [128]	62.70	42.28	61.41	41.63
B(TABL) [152]	73.62	66.16	68.81	67.12
C(TABL) [152]	78.01	72.03	74.04	72.84
DeepLOB	78.91	78.47	78.91	**77.66**
Prediction Horizon k = 50				
Naive	48.76	49.00	49.00	49.00
RR [124]	43.90	43.60	43.30	42.70
SLFN [124]	47.30	46.80	46.40	45.90
BoF [128]	50.21	42.56	49.57	39.56
N-BoF [128]	56.52	47.20	58.17	46.15
B(TABL) [152]	69.54	69.12	68.84	68.84
C(TABL) [152]	74.81	74.58	74.27	74.32
DeepLOB	75.01	75.10	75.01	**74.96**
Prediction Horizon k = 100				
Naive	49.19	49.00	49.00	49.00
RR [124]	42.90	42.90	42.90	41.60
SLFN [124]	47.70	45.30	43.20	41.00
BoF [128]	50.97	42.48	47.84	40.84
N-BoF [128]	56.43	47.27	54.99	46.86
B(TABL) [152]	69.31	68.95	69.41	68.86
C(TABL) [152]	74.07	73.51	73.80	73.52
DeepLOB	76.66	76.77	76.66	**76.58**

Table 3.3: Setup 2: Experiment Results for the FI-2010 Dataset.

Model	**Accuracy %**	**Precision %**	**Recall %**	**F1 %**
Prediction Horizon k = 10				
Naive	72.10	73.00	73.00	73.00
SVM [156]	-	39.62	44.92	35.88
MLP [156]	-	47.81	60.78	48.27
CNN-I [155]	-	50.98	65.54	55.21
LSTM [156]	-	60.77	75.92	66.33
CNN-II [157]	-	56.00	45.00	44.00
B(TABL) [152]	78.91	68.04	71.21	69.20
C(TABL) [152]	84.70	76.95	78.44	77.63
CNN	76.99	79.99	80.00	78.12
CNN+INCEP	80.84	79.59	80.85	78.82
CNN+LSTM	82.03	80.42	82.93	80.01
DeepLOB	84.47	84.00	84.47	**83.40**
Prediction Horizon k = 20				
Naive	62.16	62.00	62.00	62.00
SVM [156]	-	45.08	47.77	43.20
MLP [156]	-	51.33	65.20	51.12
CNN-I [155]	-	54.79	67.38	59.17
LSTM [156]	-	59.60	70.52	62.37
CNN-II [157]	-	-	-	-
B(TABL) [152]	70.80	63.14	62.25	62.22
C(TABL) [152]	73.74	67.18	66.94	66.93
CNN	65.72	65.77	67.68	66.68
CNN+INCEP	69.38	65.37	69.38	68.70
CNN+LSTM	72.02	71.27	70.02	68.91
DeepLOB	74.85	74.06	74.85	**72.82**
Prediction Horizon k = 50				
Naive	51.50	52.00	52.00	52.00
SVM [156]	-	46.05	60.30	49.42
MLP [156]	-	55.21	67.14	55.95
CNN-I [155]	-	55.58	67.12	59.44
LSTM [156]	-	60.03	68.58	61.43
CNN-II [157]	-	56.00	47.00	47.00
B(TABL) [152]	75.58	74.58	73.09	73.64
C(TABL) [152]	79.87	79.05	77.04	78.44
CNN	72.39	73.13	73.39	74.19
CNN+INCEP	76.92	77.11	77.23	77.15
CNN+LSTM	78.45	78.00	78.51	78.01
DeepLOB	80.51	80.38	80.51	**80.35**

3.4.3 Experiments on the London Stock Exchange (LSE)

As we suggested, the FI-2010 dataset is not sufficient to verify a prediction model - it is far too short, downsampled and taken from less liquid markets. To perform a meaningful evaluation that can hold up to modern applications, we further test our method on stocks from the LSE of one year length with a testing period of three months. We train our model on five stocks: Lloyds Bank (LLOY), Barclays (BARC), Tesco (TSCO), BT and Vodafone (VOD). Recent work of [143] suggests that deep learning techniques can extract universal features for limit order book data. To test this universality, we directly apply our model to five more stocks that were not part of the training data set **(transfer learning** [5]**)**. We select HSBC, Glencore (GLEN), Centrica (CNA), BP and ITV for transfer learning because they are also among the most liquid stocks in the LSE. The testing period is the same three months as before, and the classes are roughly balanced.

Table 3.4 presents the results of two baseline models and DeepLOB for different prediction horizons. LM represents a simple linear model and CNN-I [155] is the benchmark convolutional network for LOB data. We observe that DeepLOB delivers the best performance and the longer the prediction horizon, the worse the model performance gets. This is rational behaviour that we expect as it is more difficult to look far into future. The testing period for the LSE dataset is three months, containing millions of samples to verify this observation. Conclusions drawn here should be more reliable than those we observed from the FI-2010 dataset, and better performances are obtained with shorter prediction horizons. In terms of transfer learning (Table 3.5), we see that deep learning models deliver better results than the simple linear model, suggesting that networks have the ability to extract universal features existed in limit order books and this observation is in line with the statement in [143].

[5] We use the phrase of "transfer learning" to mean that we apply our model to data that is not included in any way in the training set. The usage of the term might be different from the standard definition and we add this note to avoid confusion.

Table 3.4: Experiment Results for the LSE Dataset (LLOY, BARC, TSCO, BT and VOD).

Model	Accuracy %	Precision %	Recall %	F1%
Prediction Horizon k = 20				
LM	57.67	57.72	57.45	57.55
CNN-I [155]	62.97	63.03	62.97	62.99
DeepLOB	**70.17**	**70.17**	**70.17**	**70.15**
Prediction Horizon k = 50				
LM	56.63	56.75	56.64	56.39
CNN-I [155]	57.26	56.62	57.26	56.74
DeepLOB	**63.93**	**63.43**	**63.93**	**63.49**
Prediction Horizon k = 100				
LM	54.71	54.02	54.71	54.06
CNN-I [155]	54.95	55.88	54.95	55.24
DeepLOB	**61.52**	**60.73**	**61.52**	**60.65**

Table 3.5: Experiment Results for Transfer Learning on the LSE Dataset (GLEN, HSBC, CNA, BP and ITV).

Model	Accuracy %	Precision %	Recall %	F1%
Prediction Horizon k = 20				
LM	56.66	56.67	56.67	56.67
CNN-I [155]	62.23	62.21	62.23	62.20
DeepLOB	**68.62**	**68.64**	**68.63**	**68.48**
Prediction Horizon k = 50				
LM	56.45	56.49	56.45	56.46
CNN-I [155]	58.47	57.89	58.47	57.98
DeepLOB	**63.44**	**62.81**	**63.45**	**62.84**
Prediction Horizon k = 100				
LM	54.12	54.99	54.12	54.40
CNN-I [155]	55.43	54.66	55.43	54.80
DeepLOB	**61.46**	**60.68**	**61.46**	**60.77**

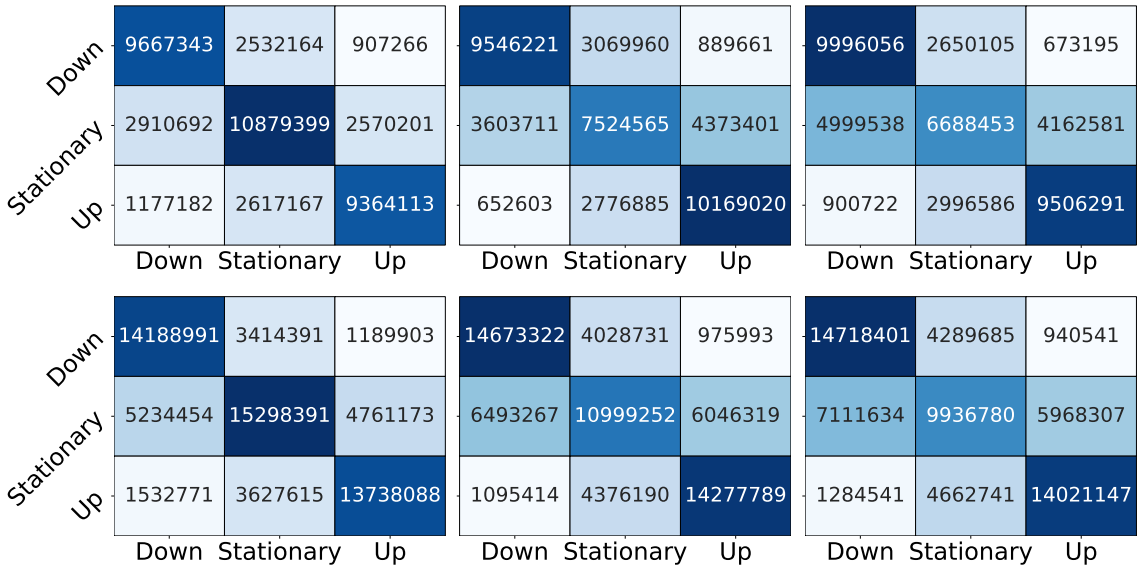

Figure 3.6: Confusion matrices for DeepLOB. **Top:** results on LLOY, BARC, TSCO, BT and VOD. From the left to right, prediction horizon (k) equals 20, 50 and 100; **Bottom:** results on transfer learning (GLEN, HSBC, CNA, BP, ITV).

To better investigate the results, we display the confusion matrices for our DeepLOB model in Figure 3.6 and calculate the accuracy for every day and for every stock across the testing period. In Figure 3.7, each point in the boxplot represents the accuracy score for one testing day. Since our testing period is three months, we have about 65 such accuracy scores for each stock and we make the boxplot from these scores. The body of the boxplot contains the scores that fall into the first and third quartiles of the observations and the line in the middle represents the median of the dataset. Lower and upper whiskers are 1.5 times interquartile range (IQR) away from the first and third quartiles. We use these boxplots to check how our model performs on a daily basis and to make sure that the performance is consistent across the testing period. We can observe robust performance, with narrow IQRs and few outliers, for all stocks across the testing period. Also, the ability of our model that generalises well to data not in the training set indicates that the CNN block in the algorithms, acting to extract features from the LOB, can capture universal patterns that relate to the price formation mechanism. We find this observation most interesting.

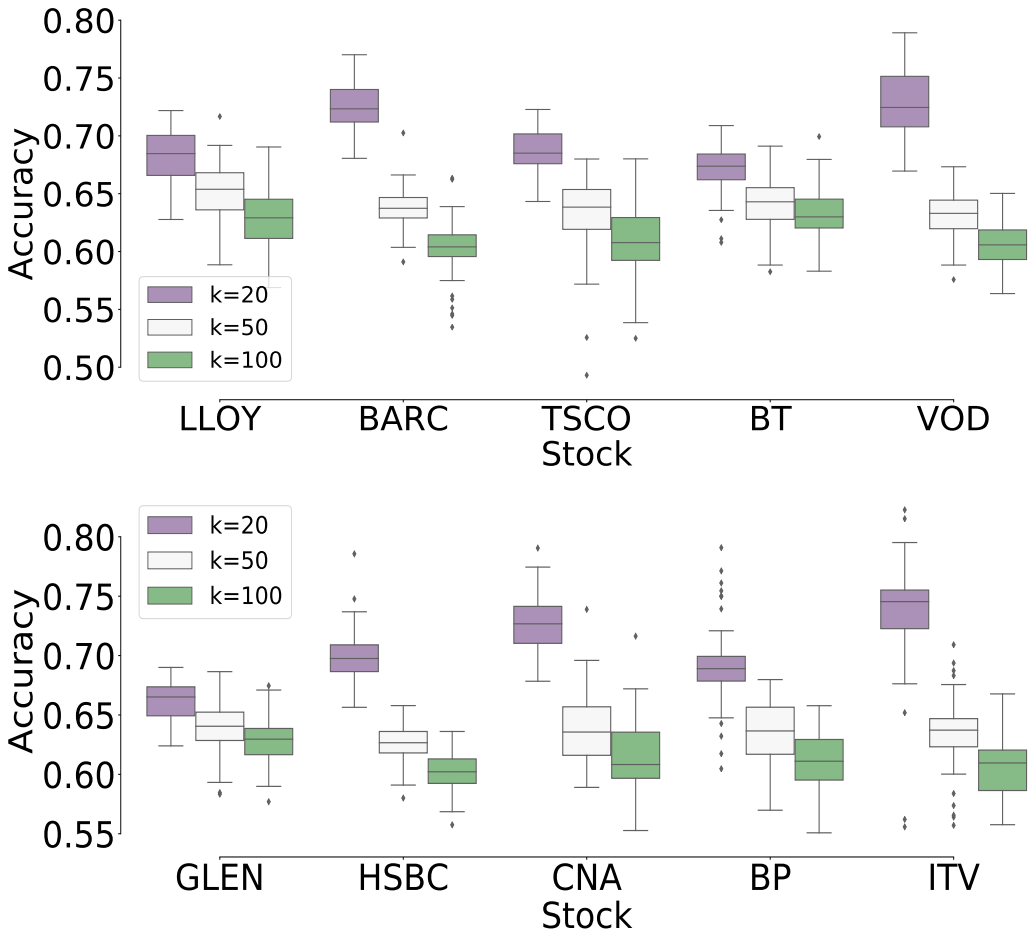

Figure 3.7: Boxplots of daily accuracy for the different prediction horizons for DeepLOB. **Top:** results on LLOY, BARC, TSCO, BT and VOD; **Bottom:** results on transfer learning (GLEN, HSBC, CNA, BP, ITV).

3.4.4 Performance of the Model in a Simple Trading Simulation

A simple trading simulation is designed to test the practicability of our results. We set the number of shares per trade, μ, to one both for simplicity and to minimise the market impact, ensuring orders to be executed at the best price. Although μ can be optimised to maximise the returns, for example, prediction probabilities are used to size the orders in [180], we would like to show that our algorithm can work even under this simple setup.

To reduce the number of trades, we use following rules to take actions. At each

time-step, our model generates a signal from the network outputs $(-1, 0, +1)$ that indicate the price movements in k steps. Signals $(-1, 0, +1)$ correspond to actions (short, wait and long). Suppose our model produces a prediction of $+1$ at time t, we then long μ shares at time $t + 5$ (taking slippage into account), and hold until -1 appears to sell all μ shares (we do nothing if 0 appears). We apply the same rule to short selling and repeat the process during a day. All positions are closed by the end of the day, so we hold no stocks overnight. We make sure that no trades take place at the time of auction, so no abnormal profits are generated.

As the focus of our work is on predictions, the above simple simulation is another way of accessing model predictions. In particular, our aim is not to present a fully developed, stand-alone trading strategy. Realistic high-frequency strategies often require a combination of various trading signals in particular to time the exact entry and exit points of the trade. For the purpose of the above simulation we use mid-prices without transaction costs. While in particular the second assumption is not a reasonable assumption for a stand-alone strategy, we argue that (i) it is enough for a relative comparison of the above models and (ii) it is a good indicator of the relative value of the above predictor to a more complex high-frequency trading model. Regarding the first assumption, a mid-mid simulation, we note that in high-frequency trading, many participants are involved in market making, as it is difficult to design profitable fully aggressive strategies with such short holding periods. If we assume that we are able to enter the trade passively, while we exit it aggressively, crossing the spread, then this is effectively equivalent to a mid-mid trade. Such a situation arises naturally for example in investment banks which are involved in client market making. Regarding the second assumption, careful timing of the entry points as well as more elaborate trading rules, such as position upsizing, should be able to account for additional profits to cover the transaction costs. In any case, as merely a metric of testing predictability of our model, the above simple simulation suffices.

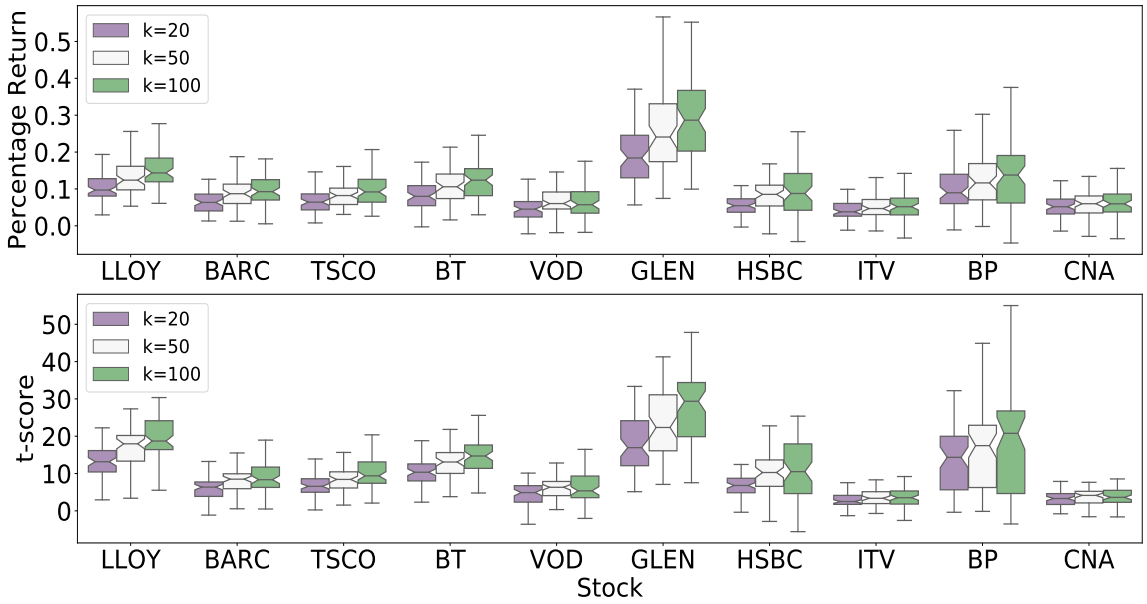

Figure 3.8: Boxplots for daily percentage returns (%) and t-statistics for daily precentage returns across different stocks and prediction horizons (k) for DeepLOB.

Figure 3.8 presents the boxplots for daily percentage return (%) for each stock. Each observation in the boxplots shows the daily percentage change of our wealth and we make the boxplots out of these percentage returns. With percentage returns, we can directly compare trading performance across different stocks and prediction horizons. The bottom of Figure 3.8 shows the t-scores for our daily percentage returns and we use t-test to check if these returns are statistically significant. Our null hypothesis is that the daily returns are equal to 0 and the alternative hypothesis is that the daily returns are greater than 0. We can also think of t-scores as a measure of risk-adjusted rewards as it measures rewards per unit of risk. In this case, the t-score is similar to Sharpe ratio but Sharpe ratio takes risk-free rate into calculation, which is inappropriate for high-frequency trading. We, therefore, report t-scores in this thesis and we can observe consistent and significant t-values over the testing period for all stocks. Although we obtain worse accuracy for longer prediction horizons, the cumulative returns in Figure 3.9 are actually higher as a more robust signal is generated.

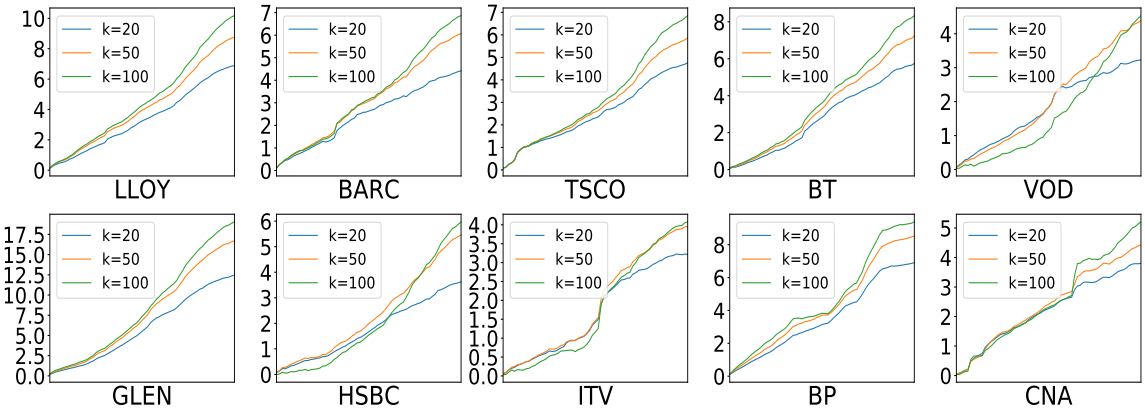

Figure 3.9: Cumulative returns for test periods for different stocks and prediction horizons (k) for DeepLOB.

3.4.5 Sensitivity Analysis

Trust and risk are fundamentals in any financial application. If we take actions based on predictions, it is always important to understand the reasons behind those predictions. Neural networks are often considered as "black boxes" which lack interpretability. However, if we understand the relationship between the inputs' components (e.g. words in text, patches in an image) and the model's prediction, we can compare those relationships with our domain knowledge to decide if we can accept or reject a prediction.

The work of [134] proposes a method, which they call LIME, to obtain such explanations. In our case, we use LIME to reveal components of LOBs that are most important for predictions and to understand why the proposed model DeepLOB works better than other network architectures such as CNN-I [155]. LIME uses an interpretable model to approximate the prediction of a complex model on a given input. It locally perturbs the input and observes variations in the model's predictions, thus providing some measure of information regarding input importance and sensitivity.

Figure 3.10 presents an example that shows how DeepLOB and CNN-I [155] react to a given input. In the figure we show the top 10 areas of pros (in green) and cons (in red) for the predicted class (yellow being the boundary). Not coloured areas

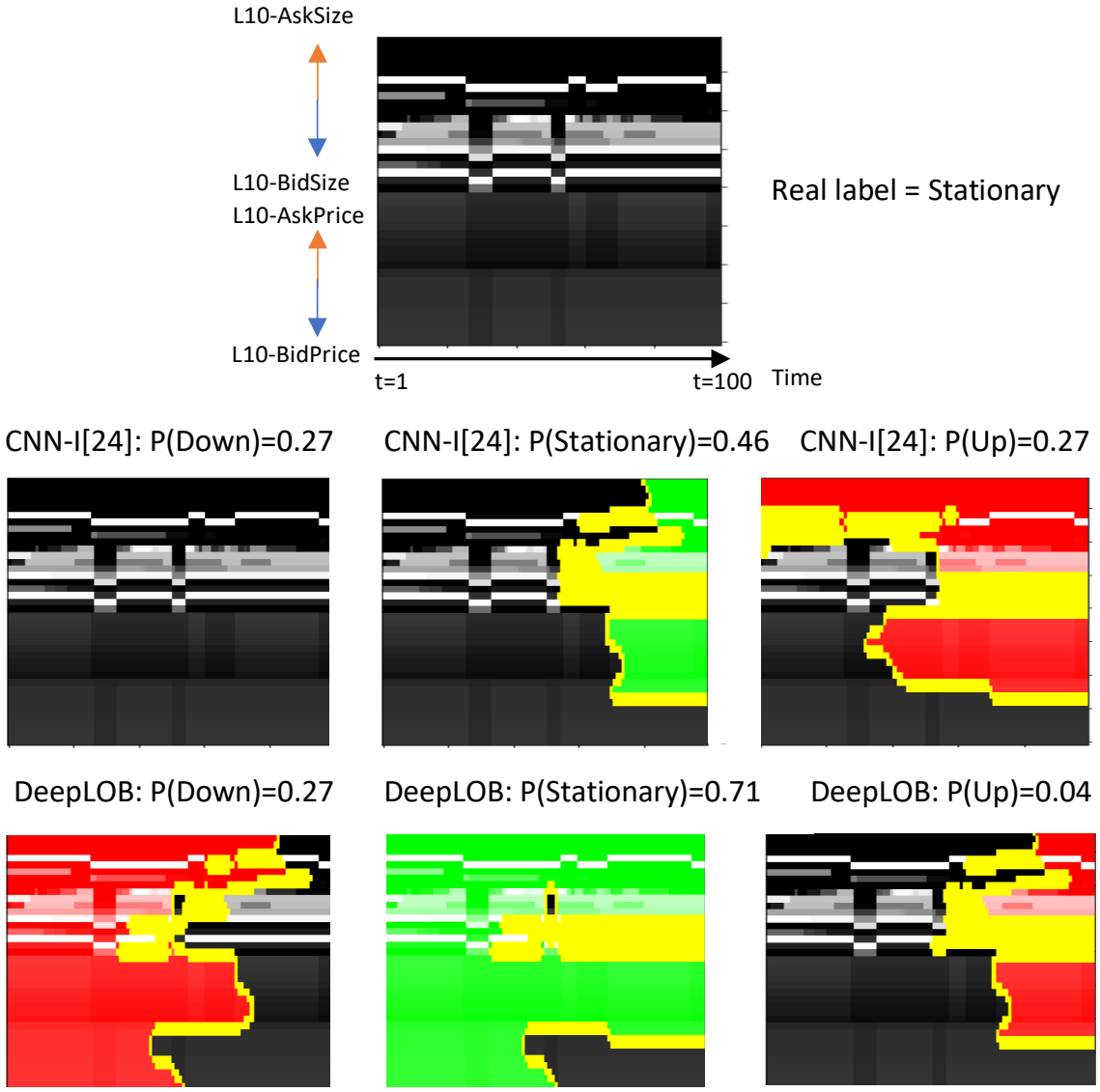

Figure 3.10: LIME plots. x-axis represents time stamps and y-axis represents levels of the LOB, as labelled in the top image. **Top:** original image. **Middle:** importance regions for CNN-I [155]. **Bottom:** importance regions for DeepLOB model. Regions supportive for predictions are shown in green, and regions against in red. The boundary is shown in yellow.

represent the components of inputs that are less influential on the predicted results or "unimportant". We note that most components of the input are inactive for CNN-I [155]. We believe that this is due to two max-pooling layers used in their architecture. Because [155] used large-size filters in the first convolutional layer, any representation deep in the network actually represents information gleaned from a large portion of

inputs. The two max-pooling layers would discard too much information from the inputs and cause underfitting problems. Our experiments applying LIME to many examples indicate this observation is a common feature.

However, LIME was initially designed for image recognition problems and we need to be cautious when drawing conclusions from this analysis. For example, if LIME detects that a classifier is looking at the face of an animal to discriminate between different species, we know that the model is making reasonable decisions. However, this is very different in financial time-series as researchers are still not sure what the key drivers are for the price formation mechanism. Even if LIME highlights certain components in our input, we can not tell if these are the "true" important features. As a result, interpretation drawn from LIME can be limited, and we mostly use this technique to perform a check of our network architecture. As mentioned above, the LIME analysis of the network in [155] shows that the pooling layers are detrimental for performance so we do not include pooling layers in our model. Recently, there has been more interest in explaining deep learning models for time-series applications. The work of [78] summarises and proposes benchmark algorithms for explaining deep learning models in time-series and following up on these works, to better understand the dynamics of financial time-series, is an avenue of future research.

3.5 Summary

We introduce the first hybrid deep neural network to predict stock price movements using high-frequency limit order book data. Unlike traditional models where features are carefully designed, we utilise a CNN and an Inception Module to automate feature extraction and use LSTM units to capture time dependencies.

The proposed method is evaluated against several baseline methods on the FI-2010 benchmark dataset and the results show that our model performs better than other

techniques in predicting short term price movements. We further test the robustness of our model by using one year of limit order data from the LSE with a testing period of three months. An interesting observation from our work is that the proposed model generalises well to instruments that did not form part of the training data. This suggests the existence of universal features that are informative for price formation and our model appears to capture these features by learning from a large data set that contains several instruments. A simple trading simulation is used to further test our model and we obtain good profits that are statistically significant.

To go beyond the often-criticised "black box" nature of deep learning models, we use LIME, a method for sensitivity analysis, to indicate the components of inputs that contribute to predictions. A good understanding of the relationship between the input's components and the model's prediction can help us decide if we can accept a prediction. In particular, we see how the information of prices and volumes on different levels of limit order books contribute to predictions.

Chapter 4

Financial Time Series Forecasting with Uncertainty

4.1 Introduction

The deep learning model presented in Chapter 3 is non-probabilistic and does not quantify uncertainty associated with predicted outputs. However, for any decision making process, especially for financial trading problems, trust and risk are primary concerns. In this chapter, we study two methods, Bayesian Neural Networks (BNNs) [16] and Quantile Regression (QR) [91], to obtain uncertainty estimates for outputs.

Uncertainty can be quantified through Bayesian inference. However, given the complexity of network models, such BNNs are often achieved by approximation such as variational inference [159]. We apply *dropout variational inference* proposed in [56] to our DeepLOB model from Chapter 3 to predict short term price movements from LOBs. We showcase that dropout techniques, as stochastic regularisers, achieve better predictive performance and we design trading strategies that can deliver improved performance using derived uncertainty information. Further, we show how uncertainty relates to position sizing, avoiding bad trades and amplifying profits.

In addition to BNNs, QR is another scalable method to measure risk exposure. QR is particularly useful for financial time-series as it models the conditional quantiles of a response. Standard financial forecasting methods result in a point estimate for financial returns, while returns are in general heterogeneous [34]. A point estimation is insufficient to describe the full distribution of returns and we can obtain considerably more information by estimating multiple quantiles. Doing this allows us to obtain an uncertainty bound on estimates that provides non-stationary information regarding risk exposure. We propose a network architecture that can simultaneously estimate multiple return quantiles from both long and short positions by training with different quantile loss functions. Further, we show how performance can be improved by combining estimates from different quantiles.

4.2 Bayesian Deep Convolutional Neural Networks for Limit Order Books [1]

4.2.1 Bayesian Inference

Bayesian inference utilises Bayes' theorem to update the probability for a hypothesis as data becomes available. Given training inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ and outputs $\boldsymbol{Y} = \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n\}$, we are interested in a function $\boldsymbol{y} = f^{\boldsymbol{w}}(\boldsymbol{x})$ with parameters $\boldsymbol{w}$. Following the Bayesian approach, we first need to put a *prior* distribution over the parameters, $p(\boldsymbol{w})$, which reflects our original belief about parameters distributions. In addition, we define a *likelihood* distribution $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$ that describes the process of generating outputs from inputs. For example, wde can use a Gaussian likelihood for regression or a softmax likelihood for classification problems [55]. Once data is available, we can derive the *posterior* distribution over the parameters by applying

[1] The content of this chapter has been published in [180].

Bayes' theorem to prior and likelihood distributions:

$$p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{Y}|\boldsymbol{X})}), \tag{4.1}$$

where $p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y})$ represents the most appropriate parameter distributions adjusted with given observations. With the posterior distribution, we can predict an output from a new input $\boldsymbol{x}^*$ by:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{X},\boldsymbol{Y}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y})d\boldsymbol{w}. \tag{4.2}$$

In many inference problems, we choose priors and likelihoods that lead to conjugate distributions because it makes the mathematics simple and tractable. For example, Gaussian distributions are often used because the multiplication of a Gaussian prior with a Gaussian likelihood function gives a posterior in the same distribution family. However, as we discussed, financial time-series are highly nonlinear and have fat tails. Distributions that better describe the dynamics of financial time-series do not lead to conjugate distributions that can be easily calculated. In these cases, we need to resort to approximation methods.

In addition, a key component in posterior evaluation is the *model evidence*:

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \int p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}, \tag{4.3}$$

where we need to integrate over the entire parameter space, known as marginalisation. Marginalisation can be done analytically for simple models, while modern networks can contain millions of parameters so the only feasible method of performing this high-dimensional integral is again to resort to approximation methods.

4.2.2 Bayesian Neural Networks

A Bayesian Neural Network (BNN) consists of a probabilistic model and a network that are intended to combine the strengths from both domains. A BNN has the advantage of being a universal function approximator and also benefits from the probabilistic guarantees on predictions, which allows models to generate posterior distributions for parameters and produce more than just a point estimate.

Early works by [16, 15, 120] laid out the foundations for BNNs and discuss the computation of posterior distributions for model parameters. There are two major families of approaches for model inference: sampling methods and variational inference methods. Sampling techniques include algorithms mostly originated from Markov Chain Monte Carlo (MCMC), such as the Metropolis Hastings Algorithm [69] and Hamiltonian Monte Carlo [43]. Sampling methods all draw sample parameters or output values from a probability distribution, rather from solving an intractable posterior distribution. However, sampling methods, mainly due to high sample variances, require long runs to obtain results. Such long runs are not scalable to high dimensional or large data problems.

Variational inference (VI) [158] , on the other hand, aims to overcome the computation difficulties of an intractable posterior by bounding it with an approximate distribution that is easy for inference. More often, the approximate distribution is factored, allowing for faster inference, even at scale. The work of [55] shows that networks using *stochastic regularisation techniques* (SRTs) such as dropout [71] can be considered, in effect, as performing variational inference to the posterior distribution. At the time of carrying out this project, we find that this is the only feasible approach that can be applied without difficulty to our problem. Indeed this technique has shown robust performance in many application domains [33]. In this thesis, we, therefore, adopt variational dropout to model limit order book data and, to the best of our knowledge, we are the first to introduce uncertainty information to such deep learning

financial applications. Further, we show how position sizing can be better decided with predictive uncertainty and trading performance improved.

In the following section, we include a short introduction to dropout induced variational inference and present our model architecture. Note that there has been active research in BNNs since the development of variational dropout. For example, recent works by [86, 87] propose gradient-based methods for approximating Bayesian inference in networks and these methods are easily scalable to large and complex problems. We think that further research comparing different approximation methods would be interesting. In particular, a study considering which method leads to better prediction uncertainty is necessary for financial applications, and this will be one of our future research directions.

4.2.3 Dropout Variational Inference

Beforing presenting the dropout variational inference [55], we briefly introduce the original Variational Inference (VI) [158]. In VI, we define a *variational* distribution $q_{\boldsymbol{\theta}}(\boldsymbol{w})$ parametrised by $\boldsymbol{\theta}$ that is easy to evaluate. We would like $q_{\boldsymbol{\theta}}(\boldsymbol{w})$ to be as close as possible to the posterior distribution, thereby, we minimise the Kullback-Leibler (KL) divergence [94] (a measure of similarity between two distributions) with respect to $\boldsymbol{\theta}$:

$$\mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})||p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y})) = \int q_{\boldsymbol{\theta}}(\boldsymbol{w})\log\frac{q_{\boldsymbol{\theta}}(\boldsymbol{w})}{p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y})}d\boldsymbol{w}. \tag{4.4}$$

If the minimum of KL is $q^*_{\boldsymbol{\theta}}(\boldsymbol{w})$, we can approximate the predictive distribution as:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{X},\boldsymbol{Y}) \approx \int p(\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{w})q^*_{\boldsymbol{\theta}}(\boldsymbol{w})d\boldsymbol{w} =: q^*_{\boldsymbol{\theta}}(\boldsymbol{y}^*|\boldsymbol{x}^*), \tag{4.5}$$

which is easier to evaluate. We can also consider KL divergence minimisation as

maximising the evidence lower bound (ELBO) with respect to $q^*_{\boldsymbol{\theta}}(\boldsymbol{w})$:

$$\begin{aligned}\mathcal{L}_{VI}(\boldsymbol{\theta}) &= \int q_{\boldsymbol{\theta}}(\boldsymbol{w}) \log p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}) d\boldsymbol{w} - \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})||p(\boldsymbol{w})) \leq \log p(\boldsymbol{Y}|\boldsymbol{X}), \\ &= \sum_{i=1}^{n} \int q_{\boldsymbol{\theta}}(\boldsymbol{w}) \log p(\boldsymbol{y}_i|f^{\boldsymbol{w}}(\boldsymbol{x}_i)) d\boldsymbol{w} - \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})||p(\boldsymbol{w})).\end{aligned} \tag{4.6}$$

However, the evaluation of $\mathcal{L}_{VI}(\boldsymbol{\theta})$ is not tractable for BNNs with more than a single hidden layer, so data sub-sampling methods are often used to approximate Equation 4.6:

$$\hat{\mathcal{L}}_{VI}(\boldsymbol{\theta}) = \frac{n}{m} \sum_{i \in S} \int q_{\boldsymbol{\theta}}(\boldsymbol{w}) \log p(\boldsymbol{y}_i|f^{\boldsymbol{w}}(\boldsymbol{x}_i)) d\boldsymbol{w} - \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})||p(\boldsymbol{w})), \tag{4.7}$$

with a random index set S of size m.

The work of [55] shows that we can consider deep learning models trained using *stochastic regularisation techniques* such as dropout as performing approximate variational inference to the posterior distribution in a BNN. Dropout is a regularisation method used in deep learning to solve overfitting problems by randomly dropping hidden units. Following [55], the weights obtained from the optimisation of a network using dropout are the same as the variational parameters of a network that has the same structure. We can then model uncertainty in Bayesian networks by a technique called *Monte-Carlo (MC) dropout*, which is essentially running a number of stochastic forward passes through our network and then averaging the results. A detailed derivation can be found in [55].

In this thesis, we adopt dropout probability to our model, DeepLOB, designed in Chapter 3; we call this new model BDLOB. Figure 4.1 shows the architecture of BDLOB. We apply a dropout layer with a rate of 0.2 after the Inception Module, chosen using grid-search methods. Note that the feature maps are still time-series, so we keep the dropout mask the same for all timesteps, which means that we drop the

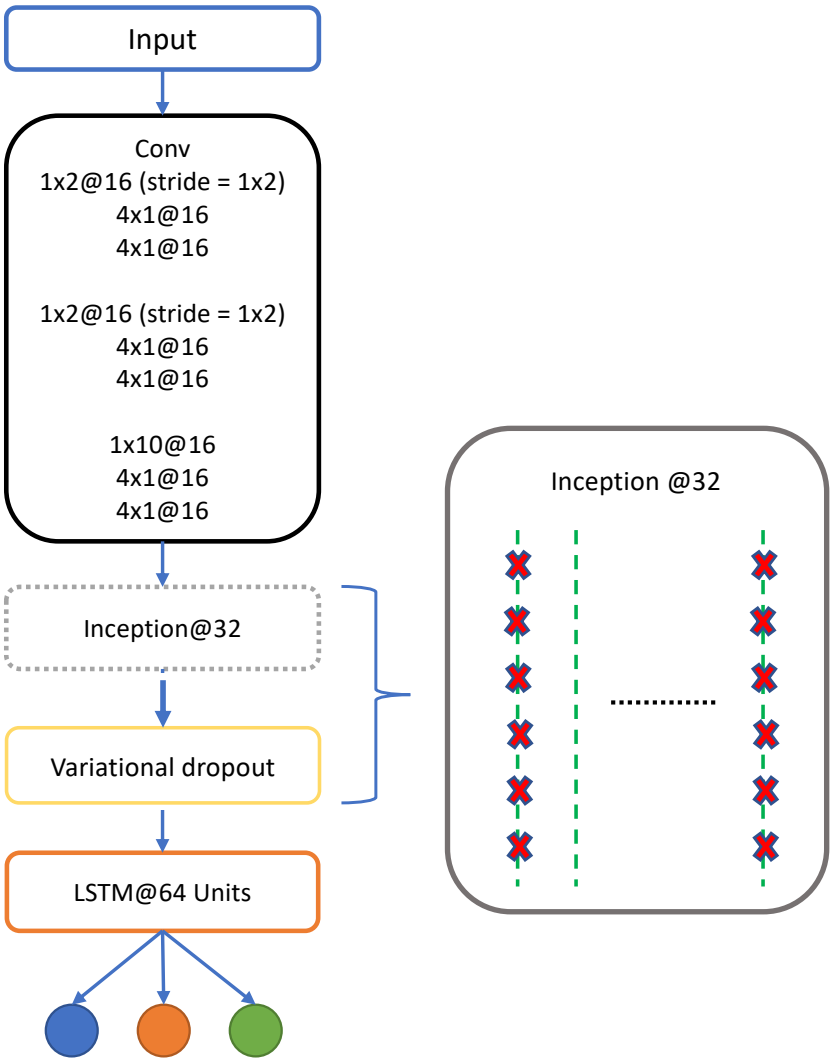

Figure 4.1: Model architecture schematic for BDLOB. Variational dropout is applied after the Inception Module and we drop the entire sequence of a feature map instead of single time stamps.

entire sequence of a feature map instead of dropping individual elements of feature maps. Doing this does not break time dependencies, but still regularises resulting features. We do not apply dropout to our convolutional blocks due to the small number of convolutional filters used. Indeed we find that adding dropout to these layers leads to underfitting problems. Similarly, LSTM units are not dropped as we only have a single LSTM layer with 64 units.

4.2.4 Experimental Results

We use the same LOB data (the LSE dataset) in Chapter 3.2.2, and follow the same procedures for normalisation, data labelling (prediction horizon $k = 100$) and train-validation-test split-off, thereby, a direct comparison can be made between DeepLOB and its Bayesian counterpart (BDLOB).

We compare it with two baseline models: the CNN model presented in [155] and DeepLOB in Chapter 3. We show each model's accuracy score, precision, recall and F1 score in Table 4.1 and we present confusion matrices and boxplots for daily accuracy in Figure 4.2. The construction of the boxplots is the same as in Figure 3.7. We calculate the accuracy score for each testing day and for each stock. We then construct the boxplots from these observations to check the consistency of model performance.

BDLOB achieves the best results, since dropout (as a set of stochastic regularisers) improves generalisation ability. Also, from the confusion matrices of Figure 4.2, we can see that BDLOB has better ability to classify the stationary class and leads to consistently better performance with narrow interquartile ranges (IQRs) and fewer outliers, as shown in the boxplots. Although the absolute difference in evaluation metrics seems small between DeepLOB and BDLOB, the results are still statistically significant under the Kolmogorov-Smirnov [107] test, indicating that they are not drawn from the same underlying distribution. The Pearson correlation of results between DeepLOB and BDLOB is about 0.6.

Table 4.1: Evaluation Metrics for Different Models and $k = 100$.

Model	**Accuracy %**	**Precision %**	**Recall %**	**F1 Score %**
CNN [155]	54.95	55.88	54.95	55.24
DeepLOB	61.52	60.73	61.52	60.65
BDLOB	62.03	61.24	62.03	61.23

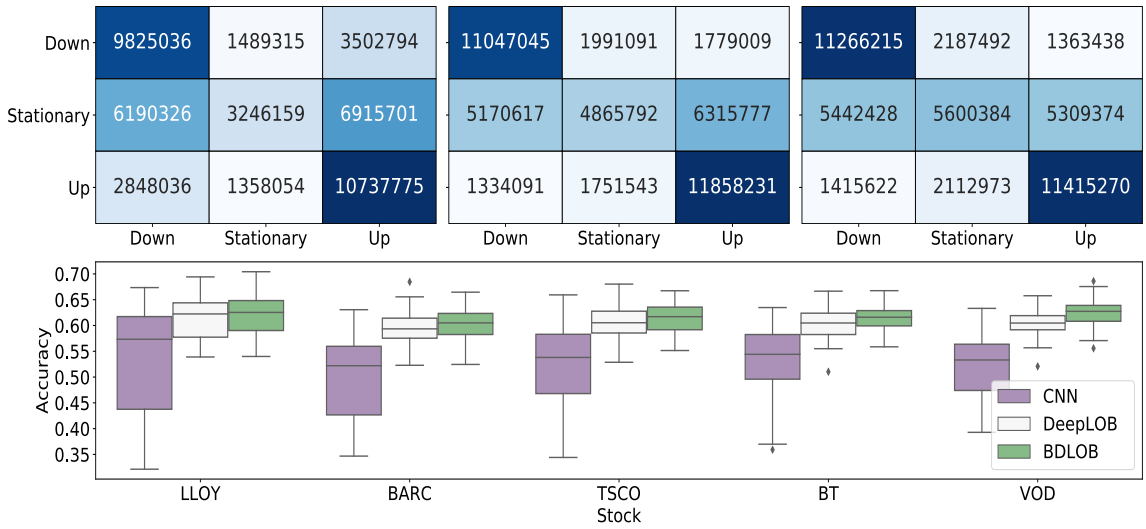

Figure 4.2: **Top:** confusion matrices for CNN [155], DeepLOB and BDLOB; **Bottom:** boxplots of daily accuracy for different models across 5 stocks.

4.2.5 Trading Strategies with Uncertainty Information

In this section, we demonstrate how uncertainty information obtained from BDLOB can be used to improve trading strategies. Recall that we classified future price movements into three classes (up=+1, neutral=0, down=-1) and used a softmax layer to give a probability vector at each time stamp $\mathbf{p}_t = [p_{+1,t}, p_{0,t}, p_{-1,t}]^T$. The basic trading strategy designed in Chapter 3 was based on discrete outputs and did not use the softmax probability vector. We briefly describe this strategy below and refer to it as **Normal trading strategy**. We then show how returns can be amplified by incorporating information from the softmax layer (**Softmax trading strategy**) and finally show how positions can be sized by using uncertainty information from the Bayesian networks (**Bayesian trading strategy**).

Normal trading strategy: The model delivers a softmax probability vector, $\mathbf{p}_t$, at each time. We enter the long position if $max(\mathbf{p}_t) = p_{+1,t}$ or the short position if $max(\mathbf{p}_t) = p_{-1,t}$. Only one position is allowed and we hold until the opposite movement is predicted (i.e. $max(\mathbf{p}_{t+m}) = p_{-1,t+m}$ in case of a long position where m

is the holding period), and we go long or short same number of shares μ at each time.

Softmax trading strategy: Same as above, but we use softmax probability vector to control our timing of entering positions. Instead of entering the trades when $max(\mathbf{p}_t)$ is equal to $p_{+1,t}$ or $p_{-1,t}$, we place a simple rule to only enter the trade if $max(\mathbf{p}_t)$ is greater than a threshold α (note that α here is not the label parameter defined in Chapter 3.2.3). The softmax probability vector contains the model confidence of each predicted class and this softmax trading strategy only allows us to enter positions if the model has a certain degree of confidence towards its prediction. Doing this largely reduces unnecessary trades and maintains our positions for relatively longer periods to profit from longer trends. We place no constraint on α to exit the trade because predictive signals always lag behind the real movements, which means that we are often too slow to exit trades, diminishing our profits. A strict exit rule will make the situation worse, so we do not use α to control for exits but we adapt uncertainty information below to exit positions.

Bayesian trading strategy: Although softmax outputs are used in the previous strategy, the work of [55] argues that softmax probability vectors are biased and could provide us with misleading information. For example, a large probability from a softmax vector could be overstated if the underlying process experiences high variance. Instead, three methods, variation ratios, predictive entropy and mutual information are suggested in [55] to summarise uncertainty in classification when using variational dropout. We test on all three quantities and discover that predictive entropy, $\mathbb{H}$, gives most consistent uncertainty information and we use this ratio to size our positions. Predictive entropy is defined as:

$$\mathbb{H}(y|\boldsymbol{x}, \mathcal{D}_{train}) = -\sum_{c} p(y = c|\boldsymbol{x}, \mathcal{D}_{train}) \log p(y = c|\boldsymbol{x}, \mathcal{D}_{train}), \tag{4.8}$$

where we sum over all possible classes c that an output y can take, and $\mathcal{D}_{train}$ represents training data. For a given input $\boldsymbol{x}$, $\mathbb{H}$ obtains its maximum value when all classes have equal probability, and its minimum value of zero is attained when one class has probability one. In other words, a small predictive entropy represents predictions with high confidence and vice versa.

We run $M(= 100)$ stochastic forward passes through our network and average the probabilities of each class to obtain the averaged probability vector $\bar{\mathbf{p}}_t$ for each input. As before, we still enter trades if $max(\bar{\mathbf{p}}_t) > \alpha$ at time t, but we upsize our positions to $1.5 \times \mu$ if $\mathbb{H} < \beta_1$ or keep original size μ if $\beta_1 < \mathbb{H} < \beta_2$ but downsize our positions to $0.5 \times \mu$ if $\mathbb{H} > \beta_2$. We would then hold this position until the opposite movement is predicted. Unlike previous strategies where we immediately exit the position, we calculate the predictive entropy for the opposite movement and only exit the position if the model is certain on this exit prediction ($\mathbb{H} < \beta_2$). Otherwise, we stay in the market to further reduce unnecessary actions.

4.2.6 Experimental Results for Different Trading Strategies

We follow the same assumptions, mid-price simulation and no transaction costs as in Chapter 3.4.4, and present how different trading strategies affect profitability. Our focus is to showcase how uncertainty information can be beneficial to decide enter and exit points and also position sizing, so the trading simulation aims to demonstrate the relative value of the here presented model when added to an existing market making strategy. We hold no stocks overnight and close all positions by the end of the day.

In Chapter 3.4.4, we use t-scores as a risk-adjusted reward measure for assessing different trading algorithms. High variance on trade returns is considered to be risky by investors because we often want smooth equity curves to ensure our confidence on trading a strategy over a long time. However, t-scores are based on standard deviation of both positive and negative trade returns. In practice, we would never consider large

positive returns as risky and that is actually what we are aiming for. Alternatively, we use the Downward Deviation [115] as a measure of risk when calculating t-scores. Downward Deviation only penalises negative returns and we call this new ratio DDR:

$$\text{DDR} = \frac{E(R_t)}{\text{DD}_T}, \quad \text{where} \quad \text{DD}_T = \Big(\frac{1}{T}\sum_{t=1}^{T}\min(R_t, 0)^2\Big)^{\frac{1}{2}}. \tag{4.9}$$

DDR is thus similar to t-scores, but only uses the standard deviation of negative returns in the calculation. The higher the DDR value, the better the strategy is. In the extreme case where all returns are positive, DDR would become indefinite.

Figure 4.3 presents the boxplots for daily percentage returns for different stocks and strategies. The construction of this figure is the same as in Figure 3.8. Any strategy using uncertainty information leads to substantial improvements in returns, and BDLOB generates the most profits and delivers good risk-adjusted rewards. Figure 4.4 shows the cumulative returns across the testing period and we observe consistent and better performance for BDLOB. We set $\alpha = 0.7$ for all strategies that use softmax probability vectors and set $\beta_1 = 0.1, \beta_2 = 0.9$ for Bayesian trading strategy. The value of these hyperparameters are optimised on our validation set and are selected to balance profits and risk. We discuss these parameters below.

Figure 4.5 shows how profit and risk change according to these hyperparameters. We can see that a higher α generally improves profits but $\alpha = 0.8$ or $\alpha = 0.9$ can lead to larger risk or lower profits – this is due to fewer trades taking place so any wrong prediction has a stronger impact on performance. As a result, we select $\alpha = 0.7$ as the best value and apply it to all Bayesian trading strategies in Figure 4.5. We set $\beta_1 = 0.1$ and find profit or risk is not hugely affected by this value because our model is very confident in its predictions when a high $\alpha = 0.7$ is used.

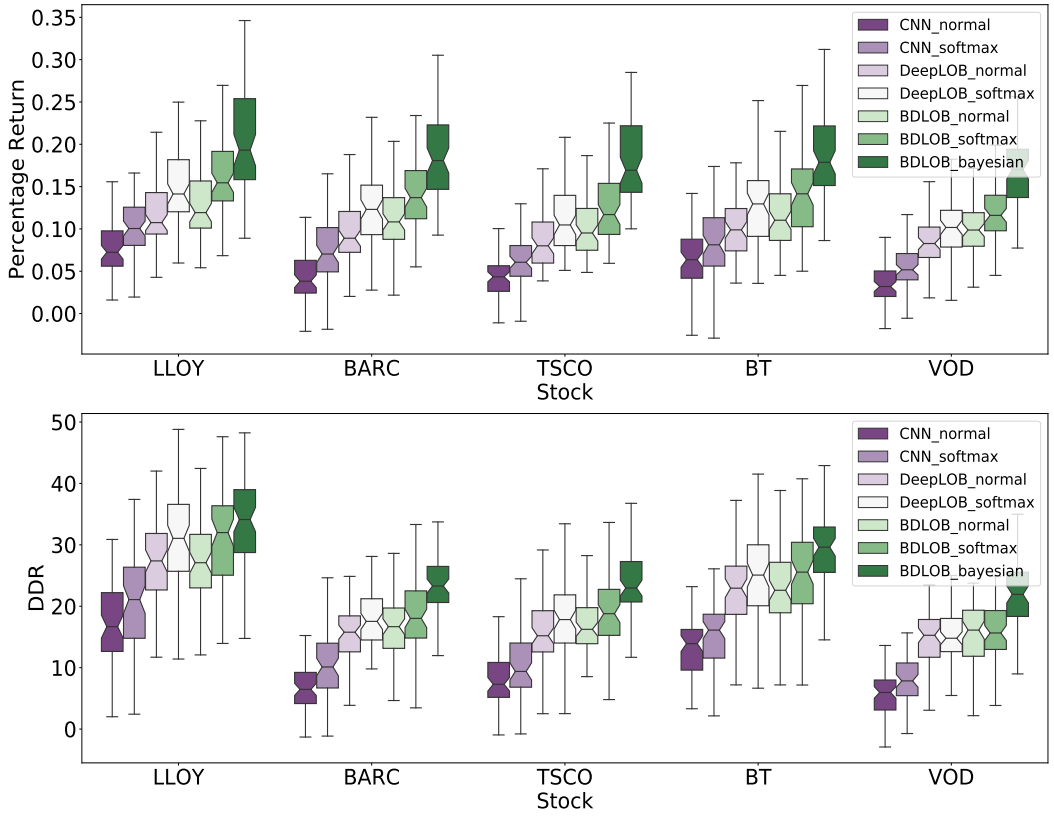

Figure 4.3: Boxplots for daily percentage returns (%) and DDR for daily percentage returns across different stocks.

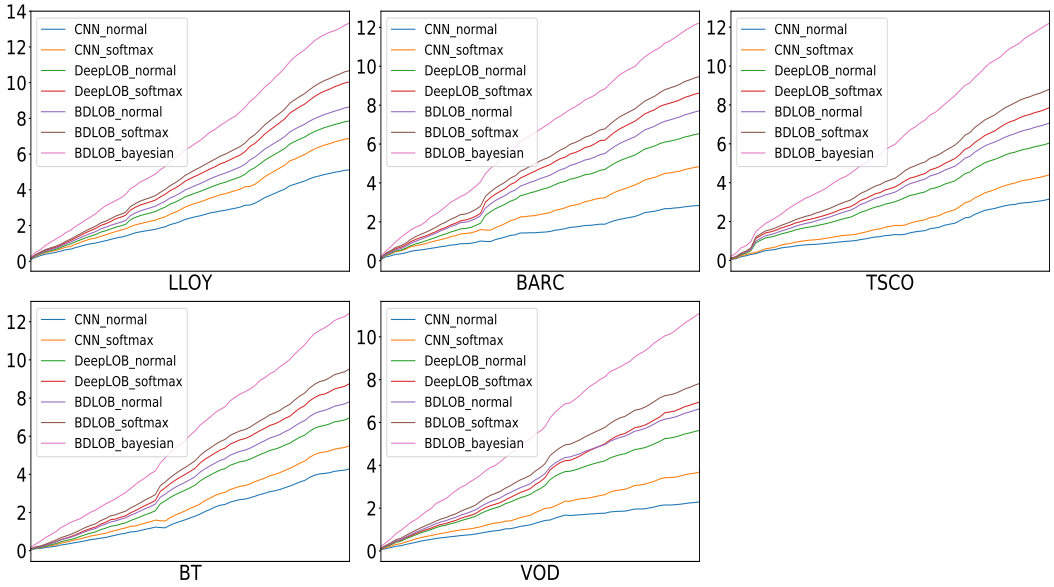

Figure 4.4: Cumulative returns for test periods for different stocks.

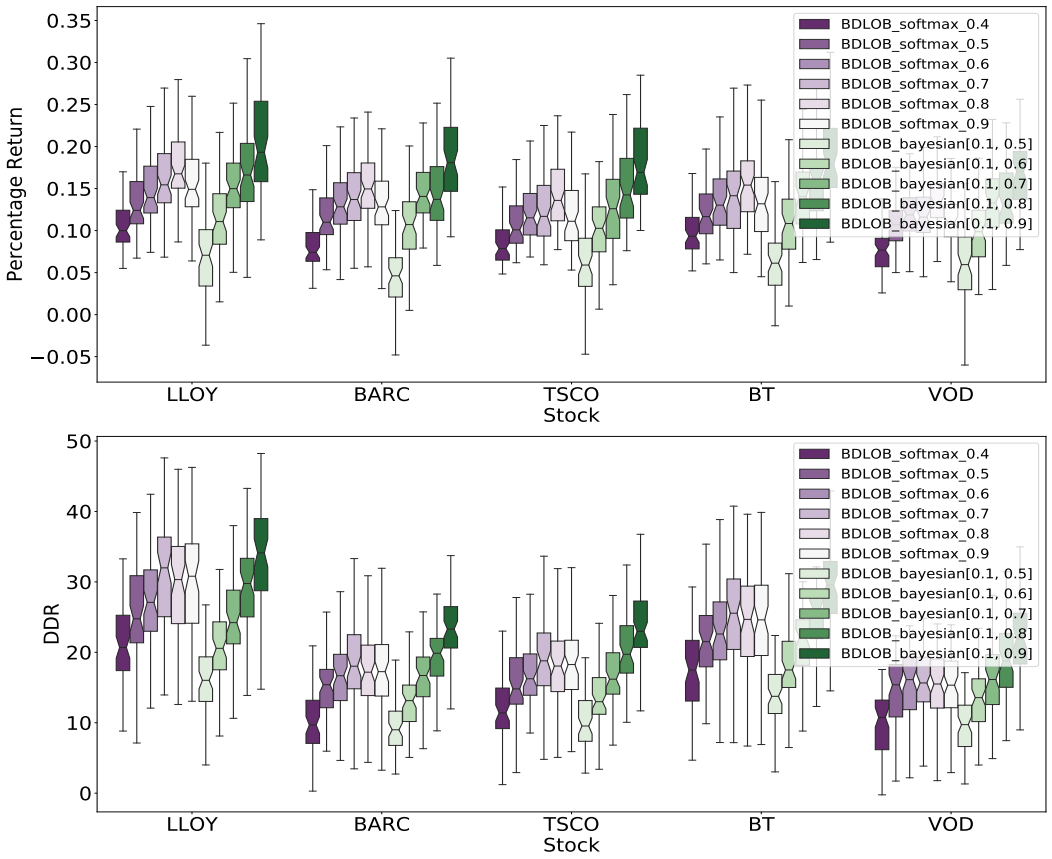

Figure 4.5: Boxplots for daily percentage returns (%) and DDR for different trading parameters. We denote a softmax strategy with rate α as BDLOB_softmax_α and Bayesian strategy as BDLOB_bayesian$[\beta_1, \beta_2]$. We set $\alpha = 0.7$ for all Bayesian trading strategies.

The effect of β_2 is more important as it not only decides the size of our positions but also controls exit time points. We can see that the smaller the β_2 becomes, the worse the returns and DDR scores we obtain. A lower β_2 means a stricter exit rule because we exit positions only if the model is very confident on this exiting signal (low β_2 means high certainty). As a result, a strict exit rule makes us to stay in a position for a longer time period, and we could hold a bad trade for too long without leaving it, leading to large variance. This is also why Bayesian trading strategies all have larger variance (measured by the distance between upper and lower whiskers of the boxplots) compared to other methods. If we stick to a position, we can profit from large trends, but also risk making big losses when our prediction is wrong.

4.3 Quantile Regression for Limit Order Books [2]

4.3.1 Task

In the previous chapter, we examined a classification setup where short-term price directions were predicted. We now estimate financial returns through a regression framework and demonstrate how Quantile Regression (QR) can be used to estimate multiple quantiles for returns from both long and short positions. Estimating returns not only provides us with the direction of markets but also indicates the strength of movements. In addition, the estimates of returns are often used in portfolio optimisation such as mean-variance analysis [106]. We define the return of an asset at time t as:

$$r_{mid}(t) = \frac{p_{mid}(t+1) - p_{mid}(t)}{p_{mid}(t)}, \tag{4.10}$$

where $p_{mid}(t)$ is the mid-price defined in Chapter 3.2.3. In the literature, mid-price is often used to represent financial time-series. Modelling mid-price is appropriate if we are using daily data, but it is improper for intraday analysis as we ignore spreads which are the differences between best ask and bid prices of a LOB.

The upper plot of Figure 4.6 illustrates the effects of spreads as transaction costs on returns. The return, $r_{mid}(t)$, from using mid-price is about three times higher than the actual return, $r_{long}(t)$, that we can obtain (we have to buy from ask sides and sell from bid sides for aggressively entering or exiting positions).

In addition, intraday trading strategies often involve both long and short positions to increase profitability. However, returns from these two positions are not symmetric at a given time stamp. We observe this at the bottom of Figure 4.6. A return, $r_{long}(t)$, is -0.23 from a long position, but it does not imply a profit of 0.23 by taking a short position ($r_{short}(t) = 0.01$). Indeed, the two return series are statistically different under Kolmogorov-Smirnov [107] and Wilcoxon signed-rank tests [167]. This discrepancy

[2] The content of this chapter has been published in [182].

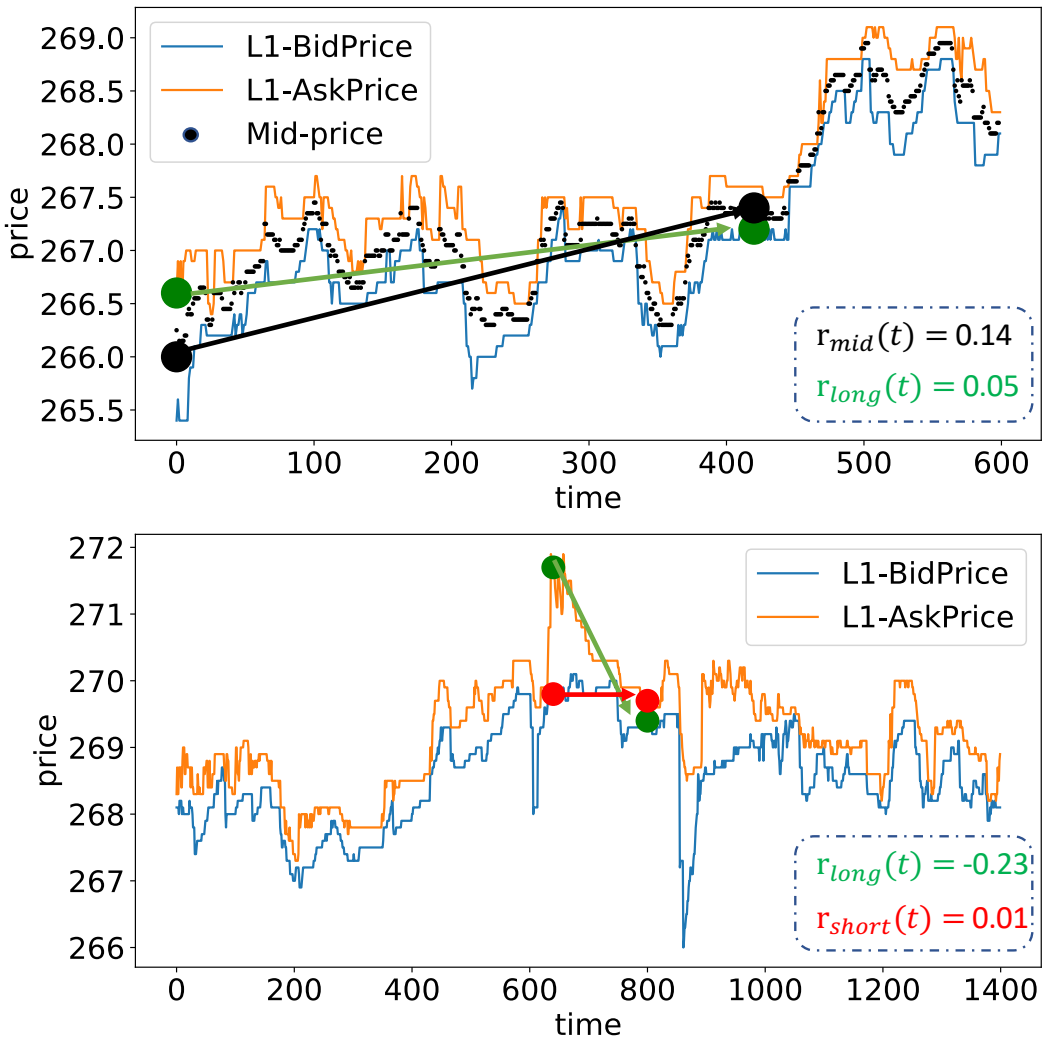

Figure 4.6: **Top:** returns obtained from using mid-prices or first level prices from LOBs. **Bottom:** returns obtained from long and short positions at a given time stamp.

comes from changing spreads and indicates that separate models are required to estimate returns for different positions.

We define returns by directly using first level prices from LOBs, thereby taking spreads into account. A return $r(t)$ at time t can be written as:

$$\begin{aligned} r_{long}(t) &= \frac{p^{(1)}_{bid}(t+k) - p^{(1)}_{ask}(t)}{p_{mid}(t)}, \\ r_{short}(t) &= -\frac{p^{(1)}_{ask}(t+k) - p^{(1)}_{bid}(t)}{p_{mid}(t)}, \end{aligned} \tag{4.11}$$

where k is the predicted horizon and $p_{mid}(t)$ is the mid-price at time t. We denote the first level ask and bid prices as $p^{(1)}_{ask}(t)$ and $p^{(1)}_{bid}(t)$. We can also write Equation 4.11 in terms of changes in mid-price and spread:

$$\begin{aligned} r_{long}(t) &= \frac{p_{mid}(t+k) - p_{mid}(t) - (s(t) + s(t+k))/2}{p_{mid}(t)}, \\ r_{short}(t) &= \frac{-(p_{mid}(t+k) - p_{mid}(t)) - (s(t) + s(t+k))/2}{p_{mid}(t)}. \end{aligned} \tag{4.12}$$

A schematic description of Equation 4.12 is given in Figure 4.7 – we cross half the spread now, follow the mid-price and cross the other half later:

$$\begin{aligned} p^{(1)}_{bid}(t+k) &= p_{mid}(t+k) - \frac{s(t+k)}{2}, \\ p^{(1)}_{ask}(t) &= p_{mid}(t) + \frac{s(t)}{2}. \end{aligned} \tag{4.13}$$

Equation 4.12 is mathematically equivalent to Equation 4.11, but this expression shows that, instead of modelling different quantiles of $(r_{long}(t), r_{short}(t))$, we can also estimate quantiles for mid-price change and spread change. Estimation of mid-price and spread could be useful for applications such as market making or trade execution where we do not need to cross spreads but to earn spreads. In this thesis, we stick to $(r_{long}(t), r_{short}(t))$.

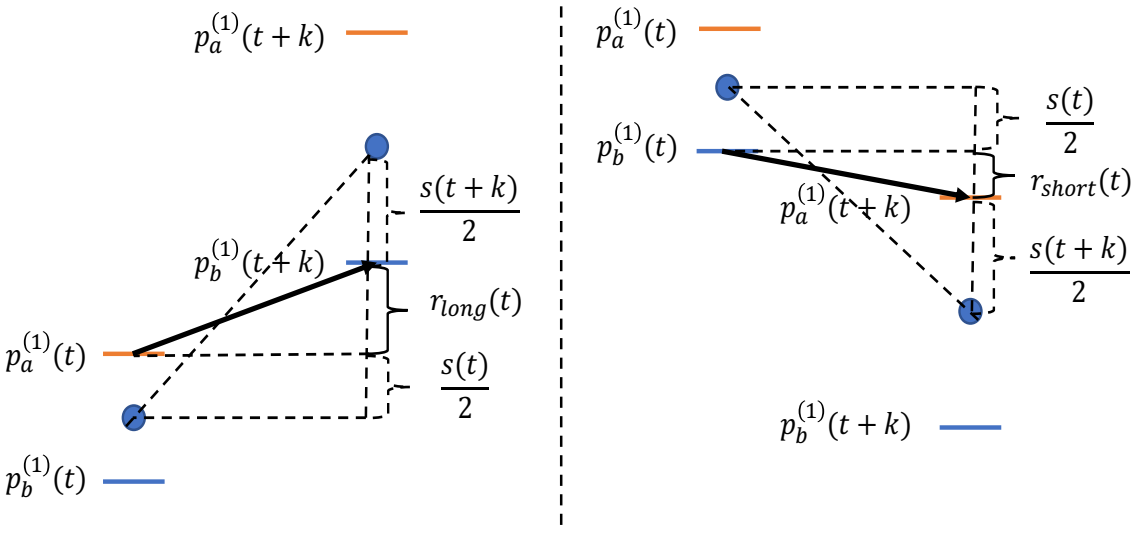

Figure 4.7: A schematic description of Equation 4.12. $p_a^{(1)}(t)$ and $p_b^{(1)}(t)$ represent best ask and bid prices at time t. **Left:** a return from a long position; **Right:** a return from a short position.

4.3.2 Methods

In Quantile Regression (QR), we predict the conditional quantile r^τ of the target distribution:

$$\mathbb{P}(r \leq r^\tau | x) = \tau, \tag{4.14}$$

where τ is the quantile of interest and x is the input. For example, if the 25th quantile of the return distributions were interested, we would find an estimate where the actual return has a 25% chance below it. We can obtain the estimates by minimizing the quantile loss (QL) with respect to $r^{(\tau)}$, leading to the τth quantile:

$$L_\tau(r, \hat{r}^\tau) = \sum_{t:r(t)<\hat{r}^\tau(t)} (\tau - 1)|r(t) - \hat{r}^\tau(t)| + \sum_{t:r(t)>\hat{r}^\tau(t)} \tau|r(t) - \hat{r}^\tau(t)|, \tag{4.15}$$

where $r(t)$ is the observation at time t and $\hat{r}^\tau(t)$ is the predicted value. Depending on the quantile chosen, asymmetric weights are given to the error in this loss function. The common used mean absolute error is equivalent to QL with $\tau = 0.5$. Note that each

quantile has its own QL function and, in order to obtain multiple quantiles, we need separate models to estimate each of them. We propose a network to solve this constraint by training with multiple QLs to model all quantiles of interest simultaneously.

However, QR has a common problem known as quantile crossing as quantile curves can cross each other, leading to an invalid distribution for the response, e.g. the predicted 90th percentile of the response is smaller than the 80th percentile which is impossible. In order to solve this problem, we follow the work of [28] to rearrange the original estimated non-monotone curve into a monotone rearranged curve.

4.3.3 Network Architecture

In this section, we present our network architecture which can simultaneously model different quantiles of returns. This model is an extension of DeepLOB presented in Chapter 3, and we denote this new architecture as DeepLOB-QR. We again use the same LSE dataset and follow the same input data preparation procedures as before.

As each quantile has its own loss function, if we are interested in 3 quantiles for returns from both long and short positions, we would need to estimate 6 separate models. This is computationally demanding for LOBs data as we have millions of them in a single day. Further, each model is essentially estimating the "same" underlying quantity (long or short returns), but just different quantiles of it. We would thus expect that there are common features that can contribute to the estimation of all models and it is wasteful to estimate them separately.

DeepLOB-QR is designed to solve the above constraints by using a common convolutional block and branching out several LSTM layers to model different interested quantiles. The "main input" takes raw LOBs data to extract features that can modulate relationships between demand and supply of an instrument. The two auxiliary inputs take past returns from long and short positions.We isolate LOBs data and past returns here because one important property of convolutional layer is parameter sharing and

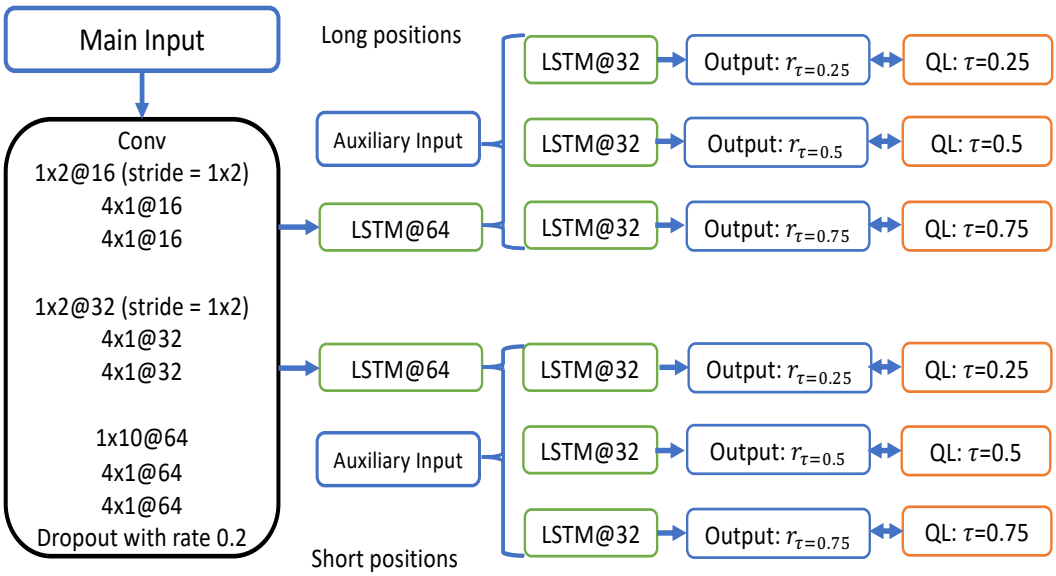

Figure 4.8: Model architecture for DeepLOB-QR. Here $1 \times 2@16$ represents a convolutional layer with 16 filters of size (1×2) . QL: τ represents the quantile loss at quantile τ.

price and volume series have different dynamics compared to returns. Overall, this is a multi-input and multi-output setup but trained using different loss functions (6 QLs in our case). The last parallel LSTM layers (LSTM@32) are only trained using their corresponding losses, while the two LSTM@64 layers are trained using 3 losses and the convolutional block is trained using all losses.

4.3.4 Forecast Combination

The works of [132, 108] suggest that the combination of individual quantile estimates can form a much more robust point estimation and help reduce prediction uncertainty. After obtaining quantile estimates $\hat{r}^{(\tau)}(t)$, $\tau \in \mathcal{S}$ where $\mathcal{S}$ denotes the set of considered quantiles, we can combine them as:

$$\hat{r}(t) = \sum_{\tau \in \mathcal{S}} \pi^{(\tau)} \hat{r}^{(\tau)}(t), \quad \sum_{\tau \in \mathcal{S}} \pi^{(\tau)} = 1, \tag{4.16}$$

where the weight $\pi^{(\tau)}$ represents the probability assigned to the prediction of quantile τ. This estimator, as a linear combination of order statistics, forms a point estimation of the central location of a distribution based on small sets of quantile estimates. We can reflect our beliefs on how each quantile estimate affects the central location by adjusting the corresponding weights.

The simplest method of assigning weights is to use a fixed weighting scheme, but we can also form it as a constrained optimisation problem [108] to find an optimal combination of quantile estimates:

$$\pi = \arg\min_{\pi} E_t[r(t-1) - \sum_{\tau \in \mathcal{S}} \pi^{(\tau)} \hat{r}^{(\tau)}(t-1)]^2, \tag{4.17}$$

where $\pi = [\pi^{(\tau)}]_{\tau \in \mathcal{S}}$ and $\sum_{\tau \in \mathcal{S}} \pi^{(\tau)} = 1$. This procedure hence treats the problem as constrained regression, where the weights are non-negative and sum to unity.

4.3.5 Experiments and Results

As in Figure 4.8, we predict three quantiles (0.25th, 0.5th and 0.75th) for returns from long positions and same three quantiles for short positions, totalling 6 estimates. Our prediction horizon k is 100 steps into the future. In order to compare with other methods, we assess how different models estimate the central location of a response. We denote the estimates of the 0.5th quantile from our model as DeepLOB-QR and estimates obtained from the combination scheme (Equation 4.17) as DeepLOB-QR(C). We compare to four other models: an autoregressive model (AR), a generalised linear model (GLR), a support vector regression (SVR) and a multilayer perceptron (MLP).

Table 4.2 shows the results of our model compared to other methods. Performance is measured by mean absolute error (MAE), mean squared error (MSE), median absolute error (MEAE) and R^2 score. All errors, except R^2 score, are divided by

Table 4.2: Experiment Results for the LSE Dataset where $\boldsymbol{k = 100}$.

Model	**MAE**	**MSE**	**MeAE**	$\boldsymbol{R^2}$
Metrics for returns from long positions				
AR	0.705	0.466	0.793	0.004
GLR	0.719	0.556	0.707	-0.188
SVR	0.802	0.498	0.755	-0.063
MLP	0.717	0.466	0.708	0.008
DeepLOB-QR	0.701	0.462	0.710	0.010
DeepLOB-QR(C)	**0.701**	**0.461**	**0.702**	**0.014**
Metrics for returns from short positions				
AR	0.708	0.498	0.791	0.004
GLR	0.736	0.528	0.709	-0.055
SVR	0.738	0.548	0.708	-0.096
MLP	0.726	0.510	0.708	0.002
DeepLOB-QR	0.705	0.501	0.720	0.013
DeepLOB-QR(C)	**0.702**	**0.494**	**0.699**	**0.015**

errors from a repetitive model[3], therefore, we understand how much better a model is compared to a zero-intelligence method. DeepLOB-QR achieves the smallest errors and the highest R^2 for modelling returns from both long and short positions. Although the absolute value of R^2 for DeepLOB-QR is small, this is kind of result we expect in practice after checking with industry practitioners. The inherent difficulty of predicting return series is high and caution should be taken if one sees superior performance in case of leaked future information.

In addition, our results suggest that better prediction results can be obtained from forecast combinations DeepLOB-QR(C) (with a fixed weighting scheme where $[\pi^{(\tau=0.25)} = 0.25, \pi^{(\tau=0.5)} = 0.5, \pi^{(\tau=0.75)} = 0.25]$). Our experiments suggest that a fixed weighting scheme delivers robust results and we have not seen improvements by deriving quantile weights from constrained optimisations (therefore omitted here). To visualise how quantile estimates from DeepLOB-QR(C) form a prediction interval,

[3]A repetitive model means using current observations as future predictions, a zero-intelligence model.

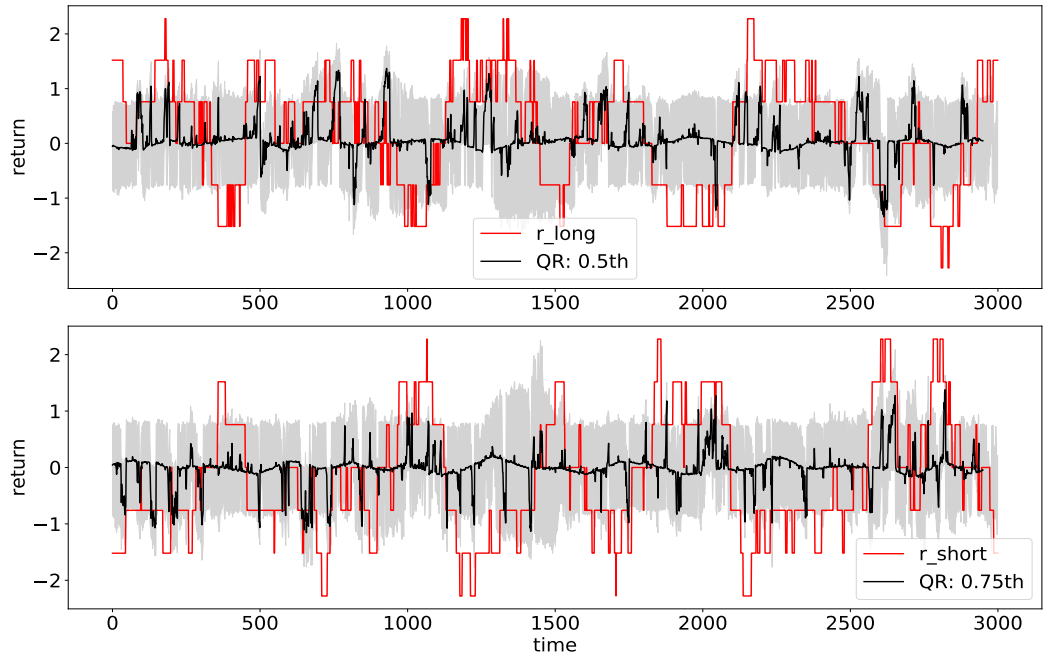

Figure 4.9: Predictions from DeepLOB-QR(C)$_{k=100}$. **Top:** real returns from long positions are in red and 0.5th quantile estimates are in black. The upper boundary of grey shading represents 0.75th quantile estimate and the lower boundary is for 0.25th quantile estimate; **Bottom:** real returns and quantile estimates for short positions.

we plot them against the true observations in Figure 4.9. We can observe that true observations are more volatile than our estimates and the central estimate (0.5th quantile estimate) is close to 0. Since R^2 measures the proportion of the variance in the target that is explained from our estimates, this could be also the reason why we have small R^2 values because our model outputs estimates with small variation.

In Table 4.3, we show how DeepLOB-QR performs at different prediction horizons (k). We test three cases: $k = 50, 100$ and 200. These prediction horizons are different from the experiments in Chapter 3.4.3, in particular, we remove $k = 20$ and include a longer horizon ($k = 200$). This is because we observe that longer prediction horizons lead to better trading performance, as in Figures 3.8 and 3.9, so we are interested in models that predict far into the future. The experiment suggests that the longer the prediction horizon, the worse the model performance we obtain. This observation

Table 4.3: Experiment Results for Different Prediction Horizons k.

Model	**MAE**	**MSE**	**MeAE**	$\boldsymbol{R}^2$
Metrics for Returns from Long Positions				
DeepLOB-QR$_{k=50}$	0.626	0.524	0.521	0.025
DeepLOB-QR(C)$_{k=50}$	0.634	0.514	0.513	0.042
DeepLOB-QR$_{k=100}$	0.701	0.462	0.710	0.010
DeepLOB-QR(C)$_{k=100}$	0.701	0.461	0.702	0.014
DeepLOB-QR$_{k=200}$	0.701	0.417	0.864	0.010
DeepLOB-QR(C)$_{k=200}$	0.719	0.419	0.841	0.011
Metrics for Returns from Short Positions				
DeepLOB-QR$_{k=50}$	0.624	0.493	0.494	0.024
DeepLOB-QR(C)$_{k=50}$	0.644	0.483	0.501	0.044
DeepLOB-QR$_{k=100}$	0.705	0.501	0.720	0.013
DeepLOB-QR(C)$_{k=100}$	0.702	0.494	0.699	0.015
DeepLOB-QR$_{k=200}$	0.700	0.428	0.864	0.012
DeepLOB-QR(C)$_{k=200}$	0.718	0.430	0.837	0.011

agrees with the results shown in Table 3.4 and meets our expectation, as the prediction problem generally becomes more different with longer prediction horizons.

4.4 Summary

In this chapter, we study two methods, Bayesian Neural Networks and Quantile Regression, to obtain uncertainty estimates for target outputs. We showcase that dropout variational inference improves performance for predicting stock price movements and demonstrate that uncertainty information can be utilised to design better trading strategies. For a regression framework, we show that QR can provide us with prediction uncertainty by forming a confidence interval and better predictive performance can be obtained by combining multiple quantile estimates.

Chapter 5

Deep Reinforcement Learning for Trading [1]

5.1 Introduction

In previous chapters, we designed trading strategies by first predicting market movements over some (fixed) horizons. However, the trading rules are discretionary as we apply some thresholds to predictions, deciding positions and aiming to optimise profits. Such a process forms a two-step optimisation problem and there is no guarantee that the first minimisation between targets and estimates would improve trading performance. Furthermore, the mapping from predictive signals to trade positions is non-trivial to construct and requires extensive hyperparameters tuning that might lead to severe overfitting problems.

In this chapter, we research Reinforcement Learning (RL) [146] algorithms to tackle above problems. Instead of making predictions - followed by a trade decision based on predictions, we train our models to directly output trade positions, bypassing the explicit forecasting step. Modern portfolio theory [4, 131, 77] implies that, given

[1] The content of this chapter has been published in [183].

a finite time horizon, an investor chooses actions to maximise some expected utility of final wealth:

$$\mathbb{E}[U(W_T)] = \mathbb{E}\left[U\left(W_0 + \sum_{t=1}^{T} \delta W_t\right)\right], \tag{5.1}$$

where U is the utility function, W_T is the final wealth over a finite time horizon T and δW_t represents the change in wealth. The performance of the final wealth measure depends upon sequences of interdependent actions where optimal trading decisions do not just decide immediate trade returns but also affect subsequent future returns. As mentioned in [109, 135], this falls under the framework of optimal control theory [90] and forms a classical sequential decision making process. If the investor is risk-neutral, the utility function becomes linear and we only need to maximise the expected cumulative trade returns, $\mathbb{E}(\sum_{t=1}^{T} \delta W_t)$, and we observe that the problem fits exactly with the framework of RL, the goal of which is to maximise some expected cumulative rewards via an agent interacting with an uncertain environment. Under the RL framework, we can directly map different market situations to trade positions and conveniently include market frictions, such as commissions, to our reward functions, allowing trading performance to be directly optimised.

5.2 Markov Decision Process Formalisation

We can formulate the trading problem as a Markov Decision Process (MDP) where an agent interacts with the environment at discrete time steps. At each time step t, the agent receives some representation of the environment denoted as a state S_t. Given this state, an agent chooses an action A_t, and based on this action, a numerical reward R_{t+1} is given to the agent at the next time step, and the agent finds itself in a new state S_{t+1}. The interaction between the agent and the environment produces a trajectory $\tau = [S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots]$. At any time step t, the goal of RL is to maximise the expected return (essentially the expected discounted cumulative

rewards) denoted as G_t at time t:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{5.2}$$

where γ is the discounting factor. If the utility function in Equation 5.1 has a linear form and we use R_t to represent trade returns, we can see that optimising $\mathbb{E}(G)$ is equivalent to optimising our expected wealth.

State Space: In the literature, many different features have been used to represent state spaces. Among these features, a security's past price is always included and the related technical indicators are often used [54]. In this thesis, we take past prices, returns (r_t) over different horizons and technical indicators including Moving Average Convergence Divergence (MACD) [10] and the Relative Strength Index (RSI) [169] to represent states. At a given time step, we take the past 60 observations of each feature to form a single state. A list of our features is below:

- Normalised close price series;
- Returns over the past month, 2-month, 3-month and 1-year periods are used. Following [100], we normalise them by daily volatility adjusted to a reasonable time scale. As an example, we normalise annual returns as $r_{t-252,t}/(\sigma_t\sqrt{252})$ where σ_t is computed using an exponentially weighted moving standard deviation of r_t with a 60-day span;
- MACD indicators are proposed in [10] where:

$$\begin{aligned} \text{MACD}_t &= \frac{q_t}{\text{std}(q_{t-252:t})}, \\ q_t &= (m(S) - m(L))/\text{std}(p_{t-63:t}), \end{aligned} \tag{5.3}$$

  where $\text{std}(p_{t-63:t})$ is the 63-day rolling standard deviation of prices p_t and $m(S)$

is the exponentially weighted moving average of prices with a time scale S;

- The RSI is an oscillating indicator moving between 0 and 100. It indicates the oversold (a reading below 20) or overbought (above 80) conditions of an asset by measuring the magnitude of recent price changes. We include this indicator with a look back window of 30 days in our state representations.

Action Space: We study both discrete and continuous action spaces. For discrete action spaces, a simple action set of $\{-1, 0, 1\}$ is used and each value represents the position directly, i.e. -1 corresponds to a maximally short position, 0 to no holdings and 1 to a maximally long position. This representation of action space is also known as target orders [74] where a trade position is the output instead of the trading decision. Note that if the current action and next action are the same, no transaction cost will occur and we just maintain our positions. If we move from a fully long position to a short position, transaction cost will be doubled. The design of continuous action spaces is very similar to discrete case where we still output trade positions but allow actions to be any value between -1 and 1 ($A_t \in [-1, 1]$).

Reward Function: The design of the reward function depends on the utility function in Equation 5.1. In this work, we let the utility function be profits representing a risk-insensitive trader, and the reward R_t at time t is:

$$R_t = \mu\Big[A_{t-1}\frac{\sigma_{tgt}}{\sigma_{t-1}}r_t - C \cdot \ p_{t-1}\ \Big|\frac{\sigma_{tgt}}{\sigma_{t-1}}A_{t-1} - \frac{\sigma_{tgt}}{\sigma_{t-2}}A_{t-2}\Big|\Big], \tag{5.4}$$

where σ_{tgt} is the volatility target and σ_{t-1} is an ex-ante volatility estimate calculated using an exponentially weighted moving standard deviation with a 60-day window on r_t. This expression forms the volatility scaling in [117, 67, 100] where our positions are scaled up when market volatility is low and scaled down vice versa. In addition, given a volatility target, our reward R_t is mostly driven by our actions instead of

being heavily affected by market volatility. We can also consider the volatility scaling as normalising rewards from different contracts to a same scale. Since our dataset consists of 50 futures contracts with different price ranges, we need to normalise different rewards to a same scale for training and also for portfolio construction. μ is a fixed number per contract at each trade and we set it to 1. Note that our transaction cost term also includes a price term p_{t-1}. This is necessary as again we work with many contracts and each contract has different costs, so we represent transaction cost as a fraction of traded value. The constant C ($= 1\text{bp} = 0.0001$) is the cost rate, for example, if the cost rate is 1bp, we need to pay \$0.1 to buy one unit of a contract priced at \$1000.

We define $r_t = p_t - p_{t-1}$ and this expression represents additive profits. Additive profits are often used if a fixed number of contracts is traded at each time. If we want to trade a fraction of our accumulated wealth at each time, multiplicative profits should be used and $r_t = p_t/p_{t-1} - 1$. In this case, we also need to change the expression of R_t as R_t represents the percentage of our wealth. The exact form can be found in [116]. We stick to additive profits in our work as logarithmic transformation needs to be taken for multiplicative profits to have cumulative rewards required by the RL setup, but logarithmic transformation penalises large wealth growths.

5.3 Reinforcement Learning Algorithms

We adopt state-of-art RL algorithms, including Deep Q-learning Networks (DQN) [113, 160], Policy Gradients (PG) [171] and Advantage Actor-Critic (A2C) [112].

Deep Q-learning Networks (DQN): A DQN approximates the state-action value function (Q function) to estimate how good it is for the agent to perform a given action in a given state by adopting a neural network. Suppose our Q function is parameterised by some θ. We minimise the mean squared error between the current

and target Q to derive the optimal state-action value function:

$$
\begin{aligned}
L(\theta) &= \mathbb{E}[(Q_\theta(S, A) - Q'_\theta(S, A))^2], \\
Q'_\theta(S_t, A_t) &= r + \gamma \ \text{argmax}_{A'} Q_\theta(S_{t+1}, A_{t+1}),
\end{aligned}
\tag{5.5}
$$

where $L(\theta)$ is the objective function. A problem is that the training of a vanilla DQN is not stable and suffers from variability. Many improvements have been made to stabilise the training process, and we adopt the following three techniques, fixed Q-targets [160], Double DQN [68] and Dueling DQN [163], to improve the training in our work.

In Equation 5.5, we minimise the error between our current $Q_\theta(S, A)$ and target $Q'_\theta(S, A)$. However, we do not have values for real targets so we use our network to estimate $Q'_\theta(S, A)$. As a result, there is a high correlation between our current and target Q-values because we use the same network to estimate both of them and, at every step of training, $Q_\theta(S, A)$ shifts and the target $Q'_\theta(S, A)$ changes due to parameter updates, leading to oscillations in training. Fixed Q-targets and Double DQN are used to reduce policy variances and to solve this problem of "chasing tails" by using a separate network (target network) with the same architecture as our DQN. We fix parameters for our target network to produce constant targets and, at every τ step, we update the target network by copying parameters from our DQN.

Recall that Q-values represent how beneficial it is for an agent to be in a given state and take an action in that state, so we can decompose the Q-value as the sum of the value of being in a given state ($V(s)$) and the advantage of taking an action in that state ($A(s, a)$). Instead of estimating $Q(s, a)$ directly, Dueling DQN separately estimates $V(s)$ and $A(s, a)$, and then combines them to produce $Q(s, a)$. A schematic illustration of Dueling DQN is shown in Figure 5.1. By decoupling the estimation, the work of [163] suggests that the network can learn a better representation of the state values because the state stream gets more updates. This is useful for situations

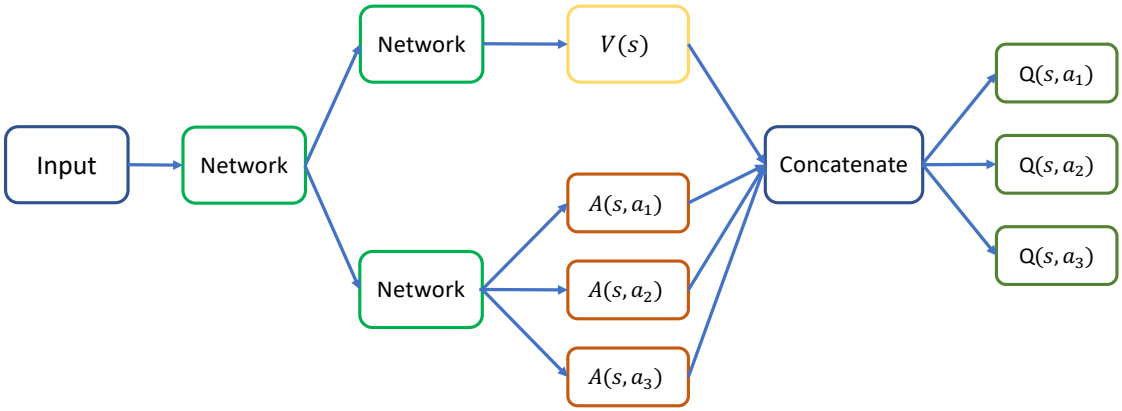

Figure 5.1: A schematic description of Dueling DQN.

where actions do not affect the environment in a relevant way and it helps us find more reliable Q-values for each action.

Policy Gradients (PG): The PG aims to maximise the expected cumulative rewards by optimising the policy directly. If we use a neural network with parameters θ to represent the policy $\pi_\theta(A|S)$, we can generate a trajectory $\tau = [S_0, A_0, R_1, S_1, \cdots, S_t, A_t]$ from the environment and obtain a sequence of rewards. We maximise the expected cumulative rewards $J(\theta)$ by using gradient ascent to adjust θ:

$$\begin{aligned} J(\theta) &= \mathbb{E}[\sum_{t=0}^{T-1} R_{t+1}|\pi_\theta], \\ \nabla_\theta J(\theta) &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) G_t, \end{aligned} \tag{5.6}$$

where G_t is defined in Equation 5.2. Compared with DQN, PG learns a policy directly and can output a probability distribution over actions. This is useful if we want to design stochastic policies or work with continuous action spaces. However, the training of PG uses Monte Carlo methods to sample trajectories from the environment and the update is done only when the episode finishes. This often results in slow training and it can get stuck at a (sub-optimal) local maximum.

Advantage Actor-Critic (A2C): The A2C is proposed to solve the training problem of PG by updating the policy in real time. It consists of two models: one is an actor network that outputs the policy and the other is a critic network that measures how good the chosen action is in the given state. We can update the policy network $\pi(A|S,\theta)$ by maximising the objective function:

$$J(\theta) = \mathbb{E}[\log \pi(A|S,\theta) A_{adv}(S,A)], \tag{5.7}$$

where $A_{adv}(S,A)$ is the advantage function defined as:

$$A_{adv}(S_t, A_t) = R_t + \gamma V(S_{t+1}|w) - V(S_t|w). \tag{5.8}$$

In order to calculate advantages, we use another network, the critic network, with parameters w to model the state value function $V(s|w)$, and we can update the critic network using gradient descent to minimise the TD-error:

$$J(w) = \left(R_t + \gamma V(S_{t+1}|w) - V(S_t|w)\right)^2. \tag{5.9}$$

The A2C is most useful if we are interested in continuous action spaces as we reduce the policy variance by using the advantage function and update the policy in real time. The training of A2C can be done synchronously or asynchronously (A3C). We adopt the synchronous approach and execute agents in parallel on multiple environments.

5.4 Experiments

5.4.1 Description of Dataset

We use daily financial data on 50 ratio-adjusted continuous futures contracts from the Pinnacle Data Corp CLC Database [31]. Our dataset ranges from 2005 to 2019, and consists of a variety of asset classes including commodity, equity index, fixed income and Foreign Exchange (FX). A full breakdown of the dataset can be found in Appendix A. We retrain our model at every 5 years, using all data available up to that point to optimise the parameters. Model parameters are then fixed for the next 5 years to produce out-of-sample results. In total, our testing period is from 2011 to 2019.

5.4.2 Baseline Algorithms

We compare our methods to the following baseline models including classical time series momentum strategies:

- Long Only;
- Sign(R) [117, 100]:

$$A_t = \text{sign}(r_{t-252:t}); \tag{5.10}$$

- MACD Signal [10]:

$$\begin{aligned} A_t &= \phi(\text{MACD}_t), \\ \phi(\text{MACD}_t) &= \frac{\text{MACD}_t \exp(-\text{MACD}_t^2/4)}{0.89}, \end{aligned} \tag{5.11}$$

where MACD_t is defined in Equation 5.3. We can also take multiple signals with

different time-scales and average them to give a final indicator:

$$\widetilde{\text{MACD}}_t = \sum_k \text{MACD}_t(S_k, L_k), \tag{5.12}$$

where S_k and L_k define short and long time-scales, namely $S_k \in \{8, 16, 32\}$ and $L_k \in \{24, 48, 96\}$ as in [100].

We compare the above baseline models with our RL algorithms, DQN, PG and A2C. DQN and PG have discrete action spaces $\{-1, 0, 1\}$, and A2C has a continuous action space $[-1, 1]$.

5.4.3 Training Schemes for RL

In our work, we use Long Short-term Memory (LSTM) [73] neural networks to model both actor and critic networks. LSTMs are traditionally used in natural language processing, but many recent works have applied them to financial time-series [53, 156, 9, 39], in particular, the work of [100] shows that LSTMs deliver superior performance on modelling daily financial data. We use two-layer LSTM networks with 64 and 32 units in all models, and Leaky Rectifying Linear Units (Leaky-ReLU) [103] are used as activation functions. As our dataset consists of different asset classes, we train a separate model for each asset class. It is a common practice to train models by grouping contracts within the same asset class, and we find it also improves our performance.

We list the value of hyperparameters for different RL algorithms in Table 5.1. We denote the learning rates for critic and actor networks as α_{critic} and α_{actor}. The Adam optimiser [89] is used for training all networks, and batch size means the size of mini-batch used in gradient descent. As previously introduced, γ is the discounting factor and C (in bp) is the cost rate used in training. We can treat C as a regularising term as a large C penalises turnovers and lets agents maintain current trade positions.

Table 5.1: Values of Hyperparameters for Different RL Algorithms.

Model	$\boldsymbol{\alpha}_{\text{critic}}$	$\boldsymbol{\alpha}_{\text{actor}}$	**Optimiser**	**Batch size**	$\boldsymbol{\gamma}$	$\boldsymbol{C}$	**Memory**	$\boldsymbol{\tau}$
DQN	0.0001	-	Adam	64	0.3	0.0020	5000	1000
PG	-	0.0001	Adam	-	0.3	0.0020	-	-
A2C	0.001	0.0001	Adam	128	0.3	0.0020	-	-

The memory shows the size of the buffer for experience replay, and we update the parameters of our target network in DQN at every τ steps.

5.4.4 Experimental Results

We test both baseline models and our methods between 2011 and 2019, and we calculate the trade returns net of transaction costs as in Equation 5.4 for each contract. We then form a simple portfolio by giving equal weights to each contract, and the trade return of a portfolio is:

$$R_t^{\text{port}} = \frac{1}{N} \sum_{i=1}^{N} R_t^i, \tag{5.13}$$

where N represents the number of contracts considered and R_t^i is the trade return for contract i at time t. We evaluate the performance of this portfolio using following metrics as suggested in [100]:

1. $E(R)$: annualised expected trade return,
2. std(R): annualised standard deviation of trade return,
3. Downside Deviation (DD): annualised standard deviation of trade returns that are negative, also known as downside risk,
4. Sharpe: annualised Sharpe ratio ($E(R)$/std(R)),
5. Sortino: a variant of Sharpe ratio that uses downside deviation as risk measures ($E(R)$/Downside Deviation),

6. MDD: maximum drawdown shows the maximum observed loss from any peak of a portfolio,

7. % +ve Returns: percentage of positive trade returns,

8. $\frac{\text{Ave. P}}{\text{Ave. L}}$: the ratio between positive and negative trade returns.

We present our results in Table 5.2 where an additional layer of portfolio-level volatility scaling is applied for each model. Equation 5.4 ensures that the trading volatility of each contract meets our target volatility, but when we form a portfolio as shown in Equation 5.13, the portfolio volatility will be always smaller than our target because of diversification effect. We, therefore, apply another volatility scaling to R_t^{port} to meet our target volatility. Also, doing this brings the volatility of different methods to a same target so we can directly compare metrics such as expected and cumulative trade returns. We also include the results without the volatility scaling for reference in Table B.1 in Appendix B. Table 5.2 is split into five parts based on different asset classes. The results show the performance of a portfolio by only using contracts from that specific asset class. An exception is the last part of the table where we form a portfolio using all contracts from our dataset. The cumulative trade returns for different models and asset classes are presented in Figure 5.2.

We can see that RL algorithms deliver better performance for most asset classes except for the equity index where a long-only strategy is better. This can be explained by the fact that most equity indices were dominated by large upward trends in the testing period. Similarly, fixed incomes had a growing trend until 2016 and entered into consolidation periods until now. Arguably, the most reasonable strategy we should have in these cases might be just to hold our positions if large trends persist. However, the long-only strategy performs the worst in commodity and FX markets where prices are more volatile. RL algorithms perform better in these markets being able to go long or short at reasonable time points. Overall, DQN obtains the best performance

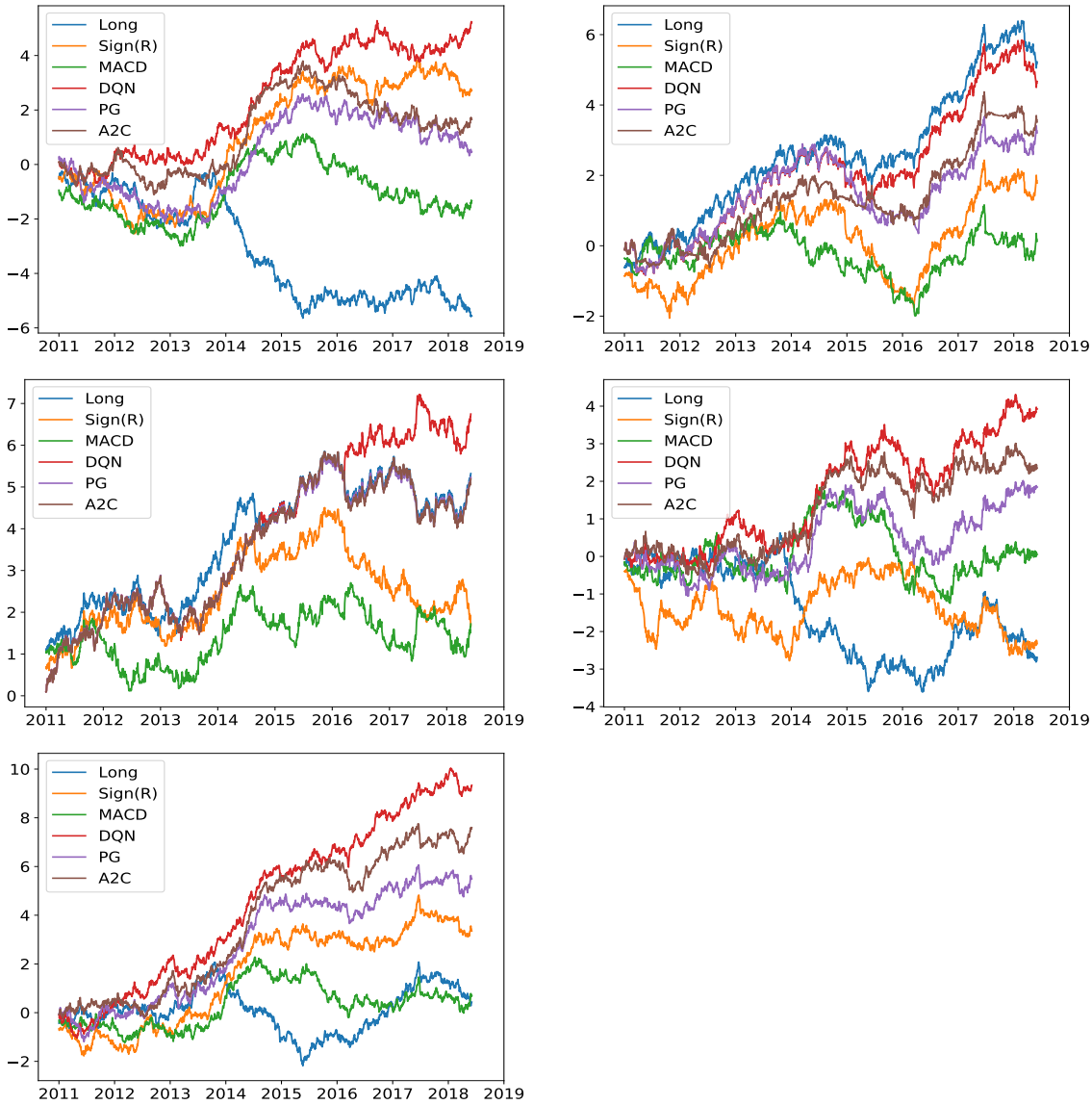

Figure 5.2: Cumulative trade returns for **First row:** commodity and equity index; **Second row:** fixed income and FX; **Third row:** the portfolio of using all contracts.

among all models and the second best is the A2C approach. We investigate the cause of this observation and find that A2C generates larger turnovers, leading to smaller average returns per turnover as shown in Figure 5.4.

We also investigate the performance of our methods under different transaction costs. In the left of Figure 5.3, we plot the annualised Sharpe ratio for the portfolio using all contracts at different cost rates. We can see that RL algorithms can tolerate

Table 5.2: Experiment Results for the Portfolio-level Volatility Targeting.

	E(R)	**Std(R)**	**DD**	**Sharpe**	**Sortino**	**MDD**	**% of + Ret**	**$\frac{\text{Ave. P}}{\text{Ave. L}}$**
				Commdity				
Long	-0.710	0.979	0.604	-0.726	-1.177	0.350	0.473	0.989
Sign(R)	0.347	0.980	0.572	0.354	0.606	0.116	0.494	1.084
MACD	-0.171	0.978	0.584	-0.175	-0.293	0.190	0.486	1.026
DQN	**0.703**	0.973	**0.552**	**0.723**	**1.275**	**0.066**	**0.498**	**1.135**
PG	0.062	0.982	0.585	0.063	0.106	0.039	0.495	1.029
A2C	0.223	0.955	0.559	0.234	0.399	0.141	0.487	1.093
				Equity Index				
Long	**0.668**	0.970	0.606	**0.688**	**1.102**	0.132	**0.542**	0.948
Sign(R)	0.228	0.966	0.610	0.236	0.374	0.344	0.528	0.930
MACD	0.016	0.962	0.618	0.017	0.027	0.311	0.519	0.927
DQN	0.629	0.970	0.606	0.648	1.038	0.161	0.541	0.944
PG	0.432	0.967	0.605	0.447	0.714	0.242	0.529	0.960
A2C	0.473	0.929	**0.593**	0.510	0.798	**0.124**	0.533	**0.962**
				Fixed Income				
Long	0.680	0.975	0.576	0.698	1.180	**0.061**	0.515	1.054
Sign(R)	0.214	0.972	0.592	0.221	0.363	0.080	0.504	1.019
MACD	0.219	0.967	0.579	0.228	0.380	0.065	0.486	**1.101**
DQN	**0.908**	0.972	**0.562**	**0.935**	**1.617**	0.062	0.515	1.098
PG	0.705	0.974	0.576	0.724	1.225	**0.061**	**0.517**	1.052
A2C	0.699	0.979	0.582	0.714	1.203	0.067	**0.517**	1.048
				FX				
Long	-0.344	0.973	0.583	-0.353	-0.590	0.423	0.491	0.979
Sign(R)	-0.297	0.973	0.592	-0.306	-0.502	0.434	0.499	0.954
MACD	0.006	0.970	0.582	0.007	0.011	0.329	0.493	1.029
DQN	**0.528**	0.967	**0.553**	**0.546**	**0.955**	0.183	**0.510**	**1.051**
PG	0.248	0.967	0.566	0.257	0.438	0.240	0.506	1.021
A2C	0.316	0.963	0.563	0.328	0.561	**0.165**	0.507	1.026
				All				
Long	0.055	0.975	0.598	0.058	0.092	0.071	0.520	0.933
Sign(R)	0.429	0.972	0.582	0.441	0.737	0.038	0.510	1.031
MACD	0.089	0.978	0.582	0.091	0.153	0.008	0.493	1.043
DQN	**1.258**	0.976	**0.567**	**1.288**	**2.220**	**0.002**	**0.543**	**1.043**
PG	0.740	0.980	0.593	0.754	1.247	0.012	0.533	0.990
A2C	1.024	0.975	0.573	1.050	1.785	0.007	0.538	1.021

Table 5.3: Market Performances over the Testing Period.

	E(R)	**Std(R)**	**Sharpe**
Stock	0.114	0.146	0.776
Bond	0.007	0.033	0.223
Commodity	-0.049	0.152	-0.324

larger cost rates and, in particular, DQN and A2C can still generate positive profits with cost rate $C = 25$bp. To understand how cost rates translate into monetary values, we plot the average cost per contract at the right of Figure 5.3 and we can see that 25bp represents roughly \$3.50 per contract. This is a realistic cost for a retail trader to pay but institutional traders have a different fee structure based on trading volumes and often have cheaper cost. In any case, this shows the validity of our methods in a realistic setup.

In order to compare our method with general market performance, we present the annualised return, standard deviation and Sharpe ratio for market indices of stock, bond and commodity in Table 5.3. We see that the stock market delivers better results compared to the bond and commodity markets, but the Sharpe ratio is still lower than our reinforcement learning models (DQN and A2C).

The performance of a portfolio is generally better than the performance of an individual contract as risks are diversified across a range of assets, so the return per risk is higher. In order to investigate the raw quality of our methods, we investigate the performance of individual contracts. We use the boxplots in Figure 5.4 to present the annualised Sharpe ratio and average trade return per turnover for each futures contract. Overall, these results reinforce our previous findings that RL algorithms generally work better, and the performance of our method is not driven by a single contract that shows superior performance, reassuring the consistency of our model.

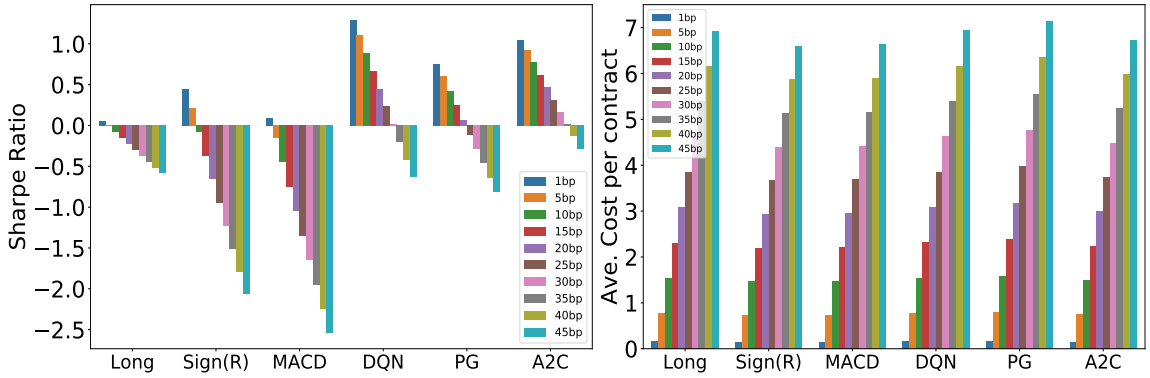

Figure 5.3: Sharpe ratio (**Left**) and average cost per contract (**Right**) under different cost rates.

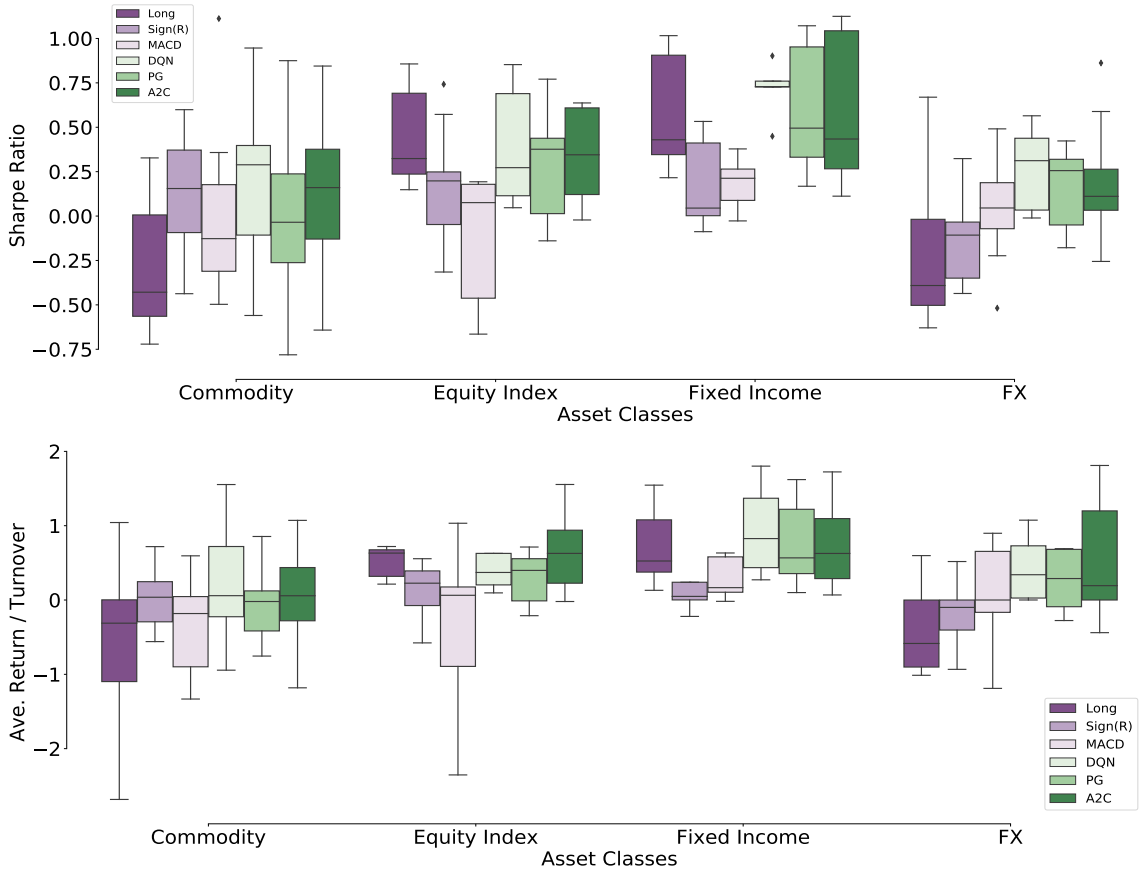

Figure 5.4: Sharpe ratio (**Top**) and average trade return per turnover (**Bottom**) for individual contracts.

5.5 Summary

We adopt RL algorithms to learn trading strategies for continuous futures contracts, and discuss the connection between modern portfolio theory and the RL reward hypothesis, showing that they are equivalent if a linear utility function is used. Our analysis focuses on Deep Q-learning Networks, Policy Gradients and Advantage Actor-Critic, and both discrete and continuous action spaces are investigated. We utilise features from time series momentum and technical indicators to form state representations, and introduce volatility scaling to improve reward functions. Our experiments show that RL algorithms outperform baseline models and deliver profits even under heavy transaction costs over a testing period of 10 years from 2011 to 2019 on 50 liquid futures contracts.

Chapter 6

Portfolio Optimisation [1]

6.1 Introduction

Following Chapter 5 where our goal was to directly maximise profits for trading individual assets, we now extend this idea to portfolio optimisation. Portfolio optimisation aims to select the best asset distribution within a portfolio in order to maximise returns at a given risk level. This theory was pioneered in Markowitz's key work [106] and is widely applied by institutional funds.

The main benefit of constructing such a portfolio comes from the promotion of diversification that smoothes out the equity curve, leading to a higher return per risk than trading an individual asset. This observation has been proven (see e.g. [185]) showing that the risk (volatility) of a long-only portfolio is always lower than that of an individual asset, for a given expected return, as long as assets are not perfectly correlated. We note that this is a natural consequence of Jensen's inequality [79].

Despite the undeniable power of such diversification, it is not straightforward to select the "right" asset allocations in a portfolio, as the dynamics of financial markets change significantly over time. Assets that exhibit, for example, strong negative correlations in the past could be positively correlated in the future. This adds extra

[1] The content of this chapter has been published in [179].

risk to the portfolio and degrades subsequent performance. Further, the universe of available assets for constructing a portfolio is enormous. Taking the US stock markets as a single example, more than 5000 stocks are available to choose from [168]. Indeed, a well-rounded portfolio not only consists of stocks, but also is typically supplemented with bonds and commodities, further expanding the spectrum of choices.

We consider directly optimising a portfolio by utilising deep learning models. Unlike classical methods [106] where expected returns are first predicted (typically through econometric models), we bypass this forecasting step to directly obtain asset allocations. Several works [116, 115, 183] have shown that the return forecasting approach is not guaranteed to maximise the performance of a portfolio, as the prediction steps attempt to minimise a prediction loss which is not the overall reward from the portfolio. In contrast, our approach is to directly optimise the Sharpe ratio [141], thus maximising return per unit of risk. Our framework starts by concatenating multiple features from different assets to form a single observation and then uses a neural network to extract salient information and output portfolio weights so as to maximise the Sharpe ratio.

Instead of choosing individual assets, Exchange-Traded Funds (ETFs) [57] of market indices are selected to form a portfolio. We use four market indices: US total stock index (VTI), US aggregate bond index (AGG), US commodity index (DBC) and Volatility Index (VIX). All of these indices are popularly traded ETFs that offer high liquidity and relatively small expense ratios. Trading indices substantially reduces the possible universe of asset choices and gains exposure to most securities. Further, these indices are generally uncorrelated, or even negatively correlated, as shown in Figure 6.1. Individual instruments in the same asset class, however, often exhibit strong positive correlations. For example, more than 75% stocks are highly correlated with the market index [168], thereby adding them to a portfolio helps less with diversification.

We are aware that subsector indices can be included in a portfolio, rather than using the total market index, since sub-industries perform at different levels and a weighting

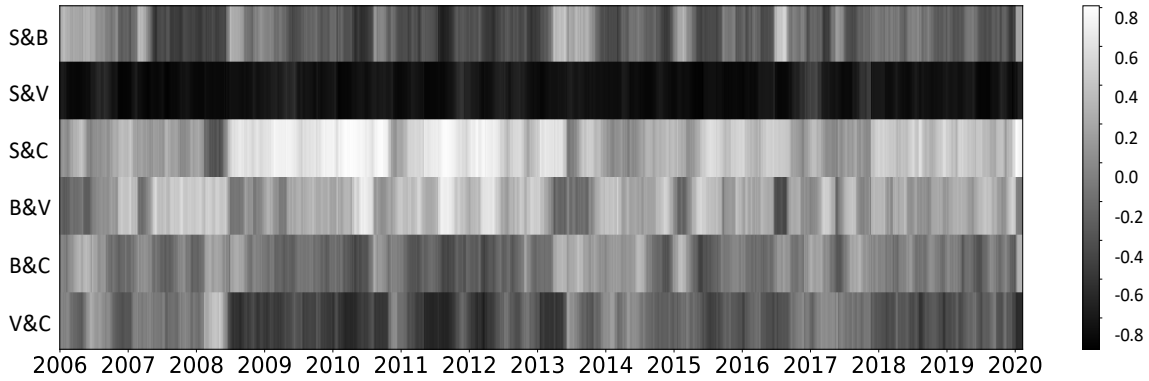

Figure 6.1: Heatmap for rolling correlations of different index pairs (S: stock index, B: bond index, C: commodity index and V: volatility index.).

on good performance in a sector would therefore deliver extra returns. However, we see subsector indices as highly correlated, thus adding them again provides minimal diversification for the portfolio and risks lowering return per unit risk. If higher returns are desired, we can use (e.g.) volatility scaling to upweight our positions and amplify returns. We therefore do not believe there is a need to find the best performing sector. Instead, we aim to provide a portfolio that delivers high return per unit risk, and allows for volatility scaling [117, 67, 100] to achieve desired return levels.

6.2 Methodology

In this section, we introduce our framework and discuss how Sharpe ratio can be optimised through gradient ascent. We discuss the types of neural networks used and detail the functionality of each component in our method.

6.2.1 Objective Function

The Sharpe ratio is used to gauge the return per risk of a portfolio and is defined as expected return over volatility (excluding risk-free rate for simplicity):

$$L = \frac{E(R_p)}{\text{Std}(R_p)}, \tag{6.1}$$

where $E(R_p)$ and $\text{Std}(R_p)$ are the estimates of the mean and standard deviation of portfolio returns. Specifically, for a trading period of $t = \{1, \cdots, T\}$, we can maximise the following objective function:

$$\begin{aligned} L_T &= \frac{E(R_{p,t})}{\sqrt{E(R_{p,t}^2) - (E(R_{p,t}))^2}}, \\ E(R_{p,t}) &= \frac{1}{T}\sum_{t=1}^{T} R_{p,t}, \end{aligned} \tag{6.2}$$

where $R_{p,t}$ is the realised portfolio return over n assets at time t denoted as:

$$R_{p,t} = \sum_{i=1}^{n} w_{i,t-1} \cdot r_{i,t}, \tag{6.3}$$

where $r_{i,t}$ is the return of asset i with $r_{i,t} = (p_{i,t}/p_{i,t-1} - 1)$. We represent the allocation ratio (position) of asset i as $w_{i,t} \in [0, 1]$ and $\sum_i^n w_{i,t} = 1$. In our approach, a neural network f with parameters θ is adopted to model $w_{i,t}$ for a long-only portfolio:

$$w_{i,t} = f(\theta | x_t), \tag{6.4}$$

where x_t represents the current market information and we bypass the classical forecasting step by linking the inputs with positions to maximise the Sharpe ratio over trading period T, namely L_T. However, a long-only portfolio imposes constraints that require weights to be positive and summed to one, we use softmax outputs to fulfil these requirements:

$$w_{i,t} = \frac{\exp(\tilde{w}_{i,t})}{\sum_j^n \exp(\tilde{w}_{j,t})}, \quad \text{where } \tilde{w}_{i,t} \text{ are the raw weights.} \tag{6.5}$$

Such a framework can be optimised using unconstrained optimisation methods. Particularly, we use gradient ascent to maximise the Sharpe ratio. The gradient of L_T with respect to parameters θ is readily calculable, and we show the derivation in

Appendix C. Once we obtain $\partial L_T/\partial\theta$, we can repeatedly compute this value from training data and update the parameters by using gradient ascent:

$$\theta_{new} := \theta_{old} + \alpha\frac{\partial L_T}{\partial\theta}, \tag{6.6}$$

where α is the learning rate and the process can be repeated for many epochs until the convergence of Sharpe ratio or the optimisation of validation performance is achieved.

6.2.2 Model Architecture

We depict our network architecture in Figure 6.2. Our model consists of three main building blocks: input layer, neural layer and output layer. The idea of this design is to use neural networks to extract cross-sectional features from input assets. Features extracted from deep learning models have been suggested to perform better than traditional hand-crafted features [183]. Once features have been extracted, the model outputs portfolio weights and we obtain realised returns to maximise Sharpe ratio. The following details each component of our method.

Input layer: We denote each asset as A_i and we have n assets to form a portfolio. A single input is prepared by concatenating information from all assets. For example, the input features of one asset can be its past prices and returns with a dimension of $(k, 2)$ where k represents the lookback window. By stacking features across all assets, the dimension of the resulting input would be $(k, 2 \times n)$. We can then feed this input to the network and expect nonlinear features being extracted.

Neural layer: A series of hidden layers can be stacked to form a network, however, in practice, this part requires lots of experiments as there are plentiful ways of combining hidden layers and the performance often depends on the design of architecture. We have tested deep learning models including fully connected neural network (FCN),

convolutional neural network (CNN) and Long Short-Term Memory (LSTM) [73]. Overall, LSTMs deliver the best performance for modelling daily financial data and a number of works [156, 100, 183] support this observation.

We note the problem of FCN is its problem of severe overfitting. As it assigns parameters to each input feature, this results in an excess number of parameters. The LSTM operates with a cell structure that has gate mechanisms to summarise and filter information from its long history, so the model ends up with fewer trainable parameters and achieves better generalisation results. In contrast, CNNs with a strong smoothing (typical of large convolutional filters) tend to have underfitting problems, such that oversmooth solutions are obtained. Due to the design of parameter sharing and the convolution operations, we experience CNNs to overfilter the inputs. However, we have shown that CNNs appear to be excellent candidates for modelling high-frequency limit order books data in Chapter 3.

Output layer: In order to construct a long-only portfolio, we use the *softmax* activation function for the output layer, which naturally imposes constraints to keep portfolio weights positive and summing to one. The number of output nodes $(w_1, \cdots, w_n)$ is equal to the number of assets in our portfolio, and we can multiply these portfolio weights with associated assets' returns $(r_1, \cdots, r_n)$ to calculate realised portfolio returns (R_p). Once realised returns are obtained, we can derive the Sharpe ratio and calculate the gradients of the Sharpe ratio with respect to the model parameters and use gradient ascent to update the parameters.

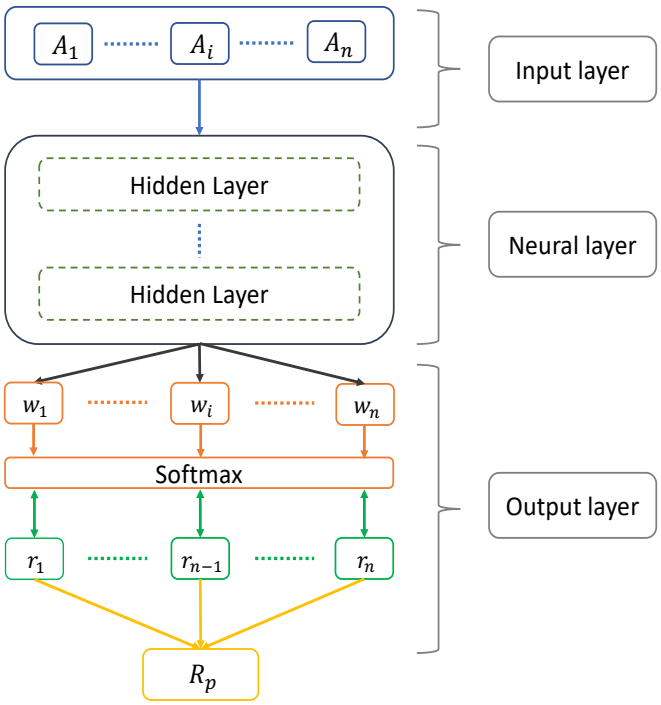

Figure 6.2: Model architecture schematic. Overall, our model contains three main building blocks: input layer, neural layer and output layer.

6.3 Experiments

6.3.1 Description of Dataset

We use four market indices: US total stock index (VTI), US aggregate bond index (AGG), US commodity index (DBC) and Volatility Index (VIX). These are popular Exchange-Traded Funds (ETFs) [57] that have existed for more than 15 years. As discussed before, trading indices offers advantages over trading individual assets because these indices are generally uncorrelated resulting in diversification. A diversified portfolio delivers a higher return per risk and the idea of our strategy is to have a system that delivers good reward-to-risk ratio. Our dataset ranges from 2006 to 2020 and contains daily observations. We retrain our model at every 2 years and use all data available up to that point to update parameters. Overall, our testing period is from 2011 to the end of April 2020, including the most recent crisis due to COVID-19.

6.3.2 Baseline Algorithms

We compare our method with a group of baseline algorithms. The first set of baseline models are reallocation strategies adopted by many pension funds. These strategies assign a fixed allocation ratio to relevant assets and rebalance portfolios annually to maintain these ratios. Investors can select a portfolio based on their risk preferences. In general, portfolios weighted more on equities would deliver better performance at the expense of larger volatility. In this work, we consider four such strategies: Allocation 1 (25% shares, 25% bonds, 25% commodities and 25% volatility index), Allocation 2 (50% shares, 10% bonds, 20% commodities, and 20% volatility index), Allocation 3 (10% shares, 50% bonds, 20% commodities, and 20% volatility index), and Allocation 4 (40% shares, 40% bonds, 10% commodities and 10% volatility index).

The second set of comparison models are mean-variance optimisation (MV) [106] and maximum diversification (MD) [150]. We use moving averages with a rolling window of 50 days to estimate the expected returns and covariance matrix. The portfolio weights are updated at a daily basis and we select weights that maximise Sharpe ratio for MV. The last baseline algorithm is the diversity-weighted portfolio (DWP) from Stochastic Portfolio Theory presented in [139]. The DWP relates portfolio weights to assets' market capitalisation and it has been suggested to be able to outperform the market index with certainty [51].

6.3.3 Training Scheme

In this thesis, we use a single layer of LSTM with 64 units to model the portfolio weights and thence to optimise the Sharpe ratio. We purposely keep our network simple to indicate the effectiveness of this end-to-end training pipeline instead of carefully fine-tuning the "right" hyperparameters. Our input contains close prices and daily returns for each market index and we take the past 50 days of these observations to form a single input. We are aware that returns can be derived from prices, but

keeping returns help with the evaluation of Equation 6.7 and we can also treat them as momentum features in [117]. As our focus is not on feature selection, we choose these commonly used features in our work. The Adam optimiser [89] is used for training our network, and the mini-batch size is 64. We take 10% of any training data as a separate validation set to optimise hyperparameters and control overfitting problems. Any hyperparameter optimisation is done on the validation set, leaving the test data for the final performance evaluation and ensuring the validity of our results. In general, our training process stops after 100 epochs.

6.3.4 Experimental Results

When reporting the test performance, we include transaction costs and use volatility scaling [117, 100, 183] to scale our positions based on market volatility. We can set our volatility target and meet expectations of investors with different risk preferences. Once volatilities are adjusted, our investment performances are mainly driven by strategies instead of being heavily affected by markets. The modified portfolio return can be defined as:

$$R_{p,t} = \sum_{i}^{n} \frac{\sigma_{tgt}}{\sigma_{i,t-1}} w_{i,t-1} \cdot r_{i,t} - C \cdot \sum_{i}^{n} \left| \frac{\sigma_{tgt}}{\sigma_{i,t-1}} w_{i,t-1} - \frac{\sigma_{tgt}}{\sigma_{i,t-2}} w_{i,t-2} \right|, \tag{6.7}$$

where σ_{tgt} is the volatility target and $\sigma_{i,t-1}$ is an ex-ante volatility estimate of asset i calculated using an exponentially weighted moving standard deviation with a 50-day window on $r_{i,t}$. We use daily changes of traded value of an asset to represent transaction costs, which is calculated by the second term in Equation 6.7. C (=1bp=0.0001) is the cost rate and we change it to reflect how our model performs under different transaction costs.

To evaluate the performance of our methods, we utilise following metrics defined in Chapter 5.4.4: expected return ($E(R)$), standard deviation of return (Std(R)), Sharpe

ratio, downside deviation of return (DD), and Sortino ratio. All of these metrics are annualised, and we also report on maximum drawdown (MDD), percentage of positive return (% of + Ret) and the ratio between positive and negative returns (Ave. P / Ave. L).

Table 6.1 presents the results of our model (DLS) compared to other baseline algorithms. The top of the table shows the performances for market indices (VTI, AGG, DBC and VIX). These indices represent the general market performance for stock, bond, commodity and volatility. We record these values to compare our method with general market performance. The second block of the table shows the results without using volatility scaling, and we can see that our model (DLS) achieves the best Sharpe ratio and Sortino ratio, delivering the highest return per risk. However, given the large differences in volatilities, we can not directly compare expected and cumulative returns for different methods, thereby volatility scaling also helps to make fair comparisons.

Once volatilities are scaled (shown in the middle of Table 6.1), DLS delivers the best performance across all evaluation metrics except for a slightly larger drawdown. If we look at the cumulative returns in Figure 6.3, DLS exhibits outstanding performance over the long haul and the maximum drawdown is reasonable, ensuring the confidence of investors to hold through hard times. Further, if we look at the bottom of Table 6.1 where a large cost rate ($C = 0.1\%$) is used, our model (DLS) still delivers the best expected return and achieves the highest Sharpe and Sortino ratios.

However, with a higher cost rate, we can see that reallocation strategies work well and, in particular, Allocations 3 and 4 achieve comparable results to our method. In order to investigate why performance gap diminishes with a higher cost rate, we present the boxplots for annual realised trade returns and accumulated costs for different assets in Figure 6.4. Overall, our model delivers better realised returns than reallocation strategies, but we also accumulate much larger transaction costs since our

Table 6.1: Experiment Results for Different Algorithms.

	E(R)	**Std(R)**	**Sharpe**	**DD**	**Sortino**	**MDD**	**% of + Ret**	**$\frac{\text{Ave. P}}{\text{Ave. L}}$**
Performance for different market indices								
VTI	0.091	0.175	0.586	0.127	0.806	0.350	0.546	1.124
AGG	0.011	0.042	0.284	0.032	0.374	0.096	0.528	1.057
DBC	-0.095	0.159	-0.545	0.120	-0.722	0.671	0.491	0.910
VIX	0.073	1.348	0.667	0.751	1.199	0.810	0.463	1.134
No volatility scaling and $C = 0.01\%$								
Allo 1	0.282	0.303	0.929	0.136	2.065	0.142	0.479	1.193
Allo 2	0.249	0.212	1.173	0.095	2.616	0.097	0.483	1.254
Allo 3	0.228	0.256	0.890	0.116	1.962	0.122	0.476	1.183
Allo 4	0.152	0.123	1.228	0.052	2.932	**0.081**	0.505	1.349
MV	0.082	0.108	0.759	0.069	1.192	0.195	**0.562**	1.199
MD	0.462	0.523	0.882	0.239	1.931	0.273	0.473	1.182
DWP	0.051	**0.102**	0.493	**0.067**	0.740	0.179	0.549	1.107
DLS	**0.313**	0.168	**1.858**	0.099	**3.135**	0.102	0.537	**1.518**
Volatility scaling ($\sigma_{tgt} = 0.10$) and $C = 0.01\%$								
Allo 1	0.160	0.105	1.526	0.061	2.629	0.111	0.554	1.289
Allo 2	0.123	0.106	1.146	0.065	1.861	0.127	0.549	1.211
Allo 3	0.145	0.105	1.383	0.061	2.396	**0.105**	0.542	1.259
Allo 4	0.164	0.104	1.579	0.064	2.588	0.112	0.565	1.303
MV	0.112	0.100	1.120	0.063	1.767	0.211	0.561	1.213
MD	0.157	0.106	1.484	0.065	2.414	0.125	0.565	1.297
DWP	0.089	0.109	0.818	0.069	1.291	0.115	0.556	1.148
DLS	**0.206**	0.105	**1.962**	0.062	**3.322**	0.123	**0.559**	**1.375**
Volatility scaling ($\sigma_{tgt} = 0.10$) and $C = 0.1\%$								
Allo 1	0.133	0.105	1.274	0.061	2.172	0.113	0.548	1.236
Allo 2	0.105	0.107	0.986	0.066	1.590	0.244	0.547	1.179
Allo 3	0.117	0.105	1.110	0.061	1.903	**0.107**	0.538	1.203
Allo 4	0.135	0.104	1.299	0.064	2.108	0.114	**0.559**	1.244
MV	0.019	0.101	0.191	0.066	0.293	0.324	0.537	1.033
MD	0.095	0.106	0.899	0.066	1.431	0.145	0.549	1.171
DWP	-0.083	0.110	-0.753	0.074	-1.129	0.627	0.508	0.880
DLS	**0.148**	0.105	**1.403**	0.063	**2.327**	0.125	0.547	**1.272**

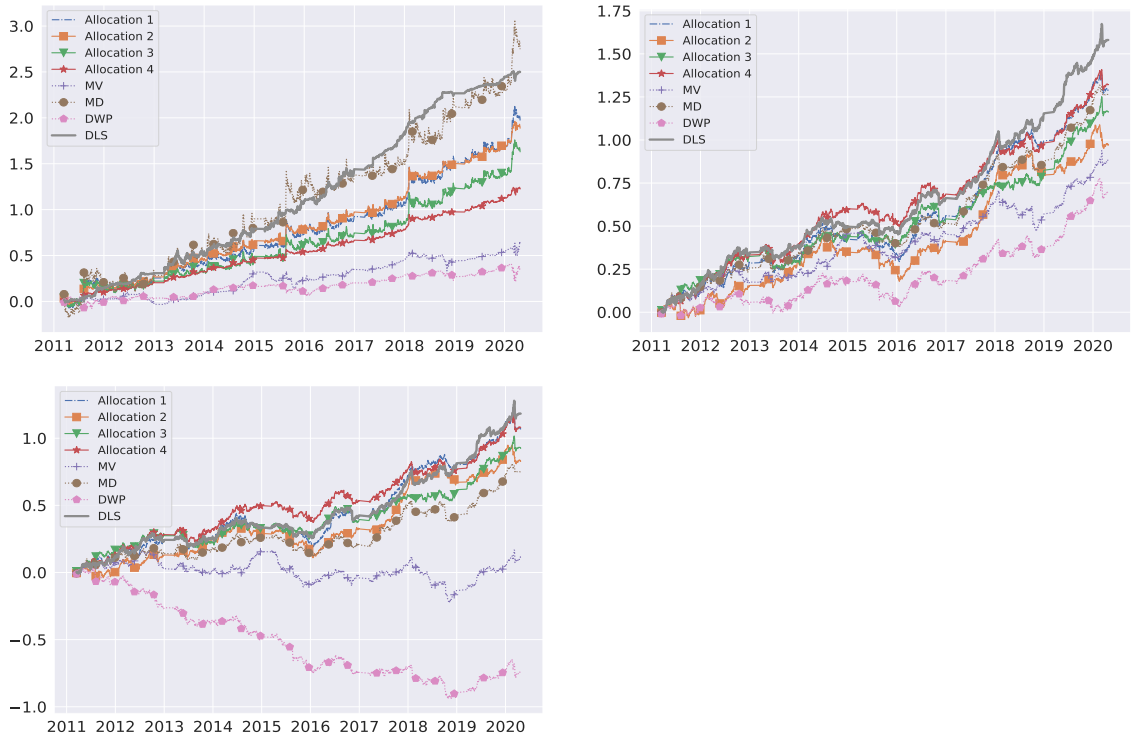

Figure 6.3: Cumulative returns (logarithmic scale) for **Top left**: no volatility scaling and $C = 0.01\%$; **Top right**: volatility scaling ($\sigma_{tgt} = 0.10$) and $C = 0.01\%$; **Bottom left**: volatility scaling ($\sigma_{tgt} = 0.10$) and $C = 0.1\%$.

positions are adjusted on a daily basis, leading to higher turnovers.

For reallocation strategies, daily position changes are only updated for volatility scaling. Otherwise, we only actively change positions once a year to rebalance and maintain the allocation ratio. As a result, reallocation strategies deliver minimal transaction costs. This analysis aims to indicate the validity of our results and show that our method can work under unfavourable conditions.

6.3.5 Model Performance during 2020 Crisis

Due to the recent COVID-19 pandemic, global stock markets fell dramatically and experienced extreme volatility. The crash started on the 24th February 2020 where markets reported their largest one-week declines since the 2008 financial crisis. Later on, with an oil price war between Russia and the OPEC countries, markets further

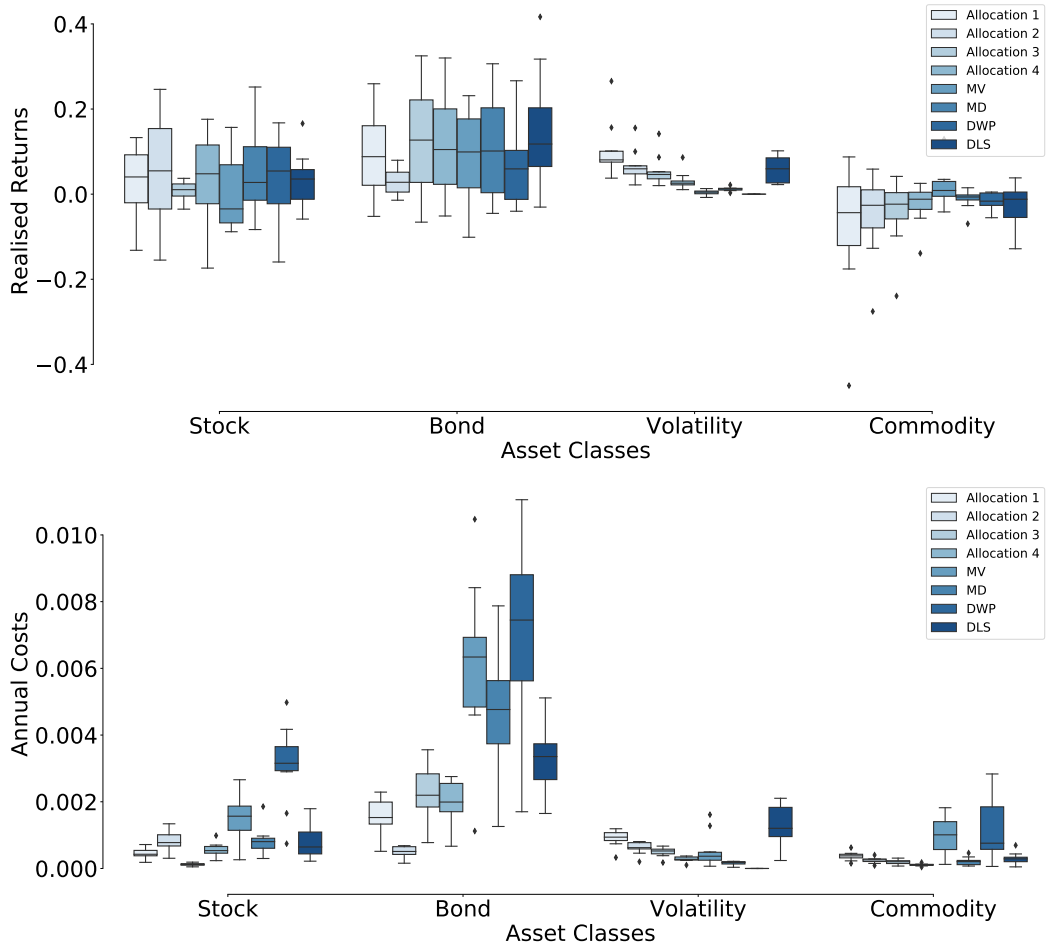

Figure 6.4: Boxplots for **Top**: annual realised trade returns; **Bottom**: annual accumulated costs for different assets with volatility scaling ($\sigma_{tgt} = 0.10$) and $C = 0.01\%$.

dampened and encountered the largest single-day percentage drop since Black Monday in 1987. As of March 2020, we have seen a downturn of at least 25% in the US markets and 30% in most G20 countries. The crisis shattered many investors' confidence and resulted in a great loss of their wealth. However, it also provides us with a great opportunity to stress test our method and to understand how our model performs during the crisis.

Table 6.2 shows the performance of different algorithms during the crisis (from January to April 2020) and we set $\sigma_{tgt} = 0.10$ and $C = 0.01\%$. We can observe that DLS generates the highest returns and delivers the second best Sharpe and Sortino

Table 6.2: Experiment Results from January to April 2020 ($\sigma_{tgt} = 0.10$ and $C = 0.01\%$).

	E(R)	**Std(R)**	**Sharpe**	**DD**	**Sortino**	**MDD**	**% of + Ret**	**$\frac{\text{Ave. P}}{\text{Ave. L}}$**
Allo 1	-0.057	0.138	-0.411	0.106	-0.539	0.111	0.594	0.924
Allo 2	-0.195	0.141	-1.382	0.103	-1.905	0.128	0.519	0.783
Allo 3	0.192	0.137	1.407	0.095	2.018	0.105	0.595	1.309
Allo 4	0.093	0.148	0.633	0.112	0.831	0.112	0.608	1.123
MV	0.279	0.138	**2.025**	0.097	2.895	0.088	**0.708**	1.525
MD	0.175	**0.097**	1.809	**0.059**	**2.961**	**0.057**	0.658	1.405
DWP	0.092	0.148	0.620	0.114	0.807	0.115	0.646	1.124
DLS_{raw}	0.209	0.147	1.418	0.112	1.868	0.102	0.633	1.377
DLS	**0.368**	0.183	2.009	0.125	2.935	0.123	0.658	**1.531**

ratios. We also report the results of our model without using volatility scaling (DLS_{raw}) to make sure that the good performances are not just made by the scaling technique, and we see that DLS_{raw} delivers decent results with a reasonable drawdown that can be tolerated during bad times.

In order to study our model behaviours, we plot how our algorithm allocated the assets from January to April 2020 in Figure 6.5. At the beginning of 2020, we can see that our model had a quite diverse holding. However, after a small dip in stock index in early February, we almost had only bonds in our portfolio. There were some equity positions left but very small positions for volatility and commodity indices. When the crash started on 24th February, our holdings were concentrated on the bond index which is considered to be the safe asset during the crisis. Interestingly, the bond index also fell this time (in the middle of March) although it rebounded quite quickly. During the bond falling, our original positions did not change much but the scaled positions decreased a lot for the bond index due to a spiking volatility, therefore our drawdown was small. Overall, we can see that our model delivers reasonable allocations during the crisis and our positions are protected through volatility scaling.

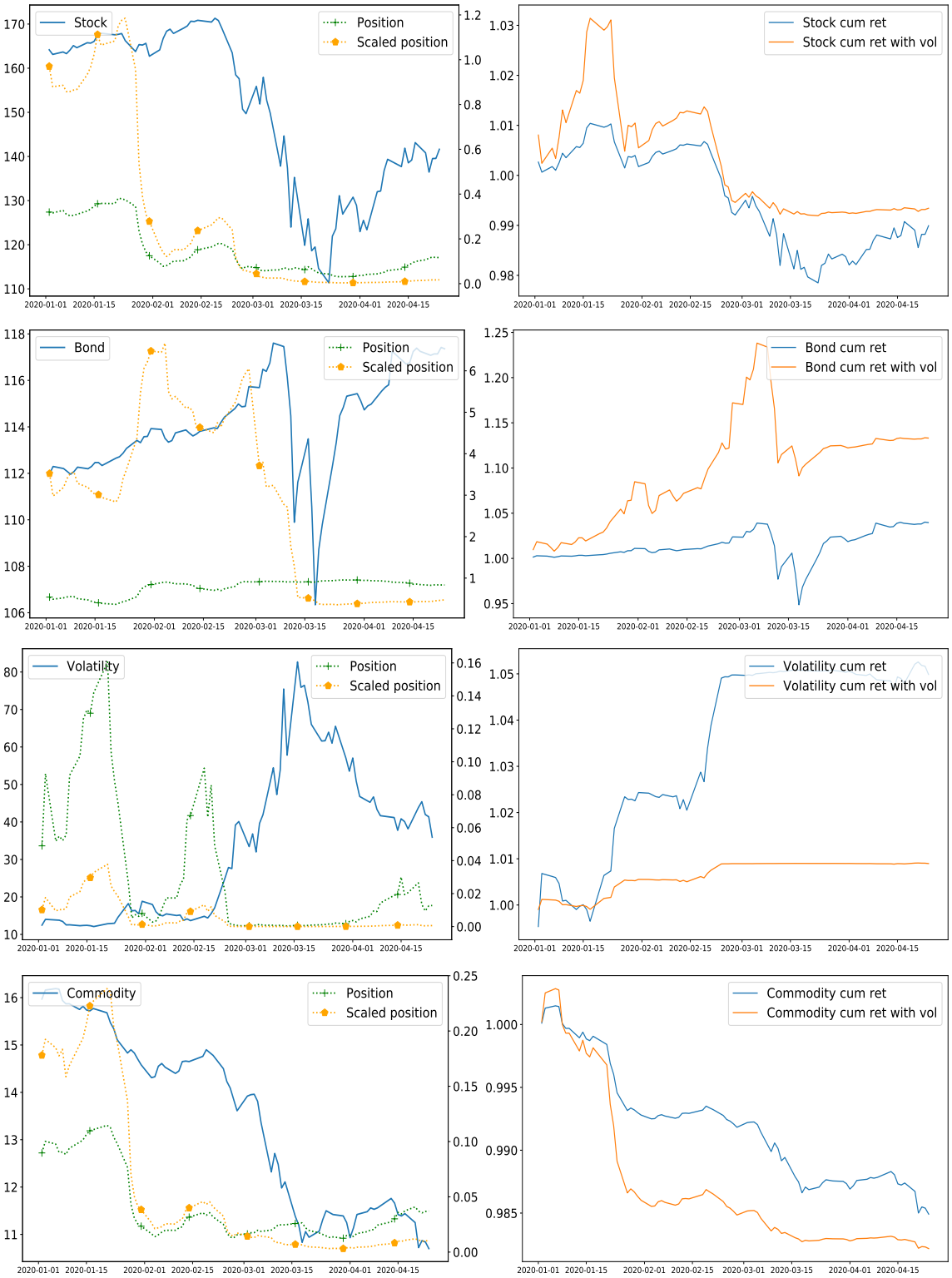

Figure 6.5: Shifts of portfolio weights (left column) and cumulative returns (**cum ret** on the right column) for our model (DLS) during COVID-19 with $\sigma_{tgt} = 0.10$.

6.3.6 Sensitivity Analysis

In order to understand how input features affect our decisions, we study the sensitivity analysis presented in [115] for our method. The absolute normalised sensitivity of feature x_i is defined as:

$$S_i = \frac{\frac{dL}{dx_i}}{\max_j \left| \frac{dL}{dx_j} \right|}, \tag{6.8}$$

where L represents the objective function and S_i captures the relative sensitivity for feature x_i compared to other features. We plot the time-varying sensitivities for all features in Figure 6.6. The y-axis indicates the 400 features we have because we use 4 indices (each with prices and returns) and we take past 50 observations to form a single input so there are 400 features in total. The row labelled "Sprice" represents price features for the stock index and the bottom of row "Sprice" means the most recent price for that observation. The same convention is used for all other features.

The importance of features varies over the time, but the most recent features always make the biggest contributions as we can see that the bottom of each feature row has the highest weight. This observation meets our understanding as, for time-series, recent observations carry more information. The further away from the current observation point, the less importance of features show and we could adjust features used based on this observation.

However, this sensitivity analysis shares the same limitations as we discussed for the LIME method in Chapter 3. We mostly use this technique to select features that are active to the objective function and inactive features could be dropped to reduce the dimension of input. While very limited information can be drawn from this method to understand how the model decides portfolio weights based on input data. This lack of understanding could be problematic for industrial deployment as it affects our confidence in holding onto this strategy when abnormal decisions are produced since we can not tell if these decisions are reasonable.

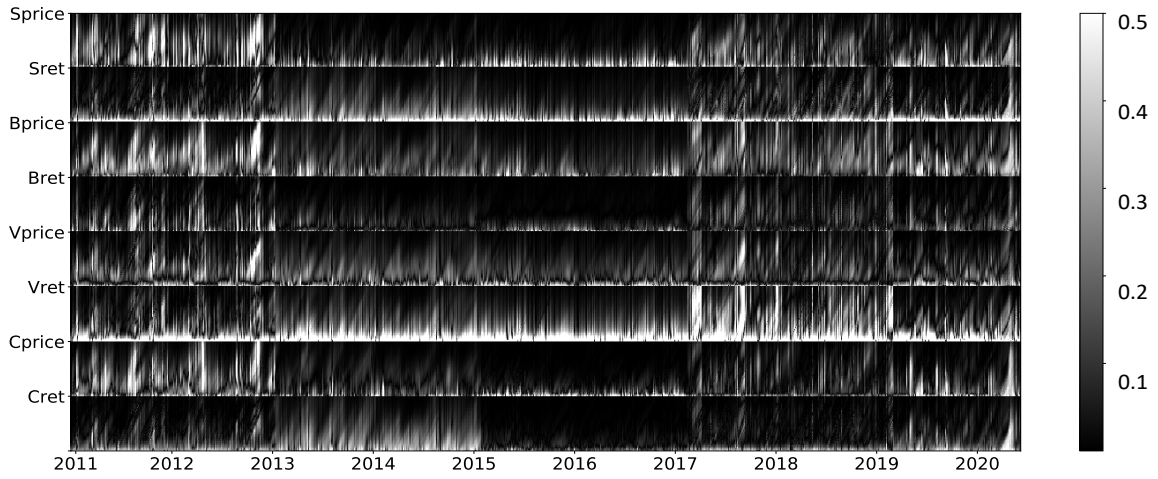

Figure 6.6: Sensitivity analysis for input features over the time.

6.4 Summary

In this chapter, we adopt deep learning models to directly optimise a portfolio's Sharpe ratio. This pipeline bypasses the traditional forecasting step and allows us to optimise portfolio weights by updating model parameters through gradient ascent. Instead of using individual assets, we focus on ETFs of market indices to form a portfolio. Doing this substantially reduces the scope of possible assets to choose from, and these indices have shown robust correlations.

We compare our method with a wide range of popular algorithms including reallocation strategies, classical mean-variance optimisation, maximum diversification and stochastic portfolio theory model. Our testing period is from 2011 to the April of 2020, and include the recent crisis due to COVID-19. The results show that our model delivers the best performance and a detailed study of our model performance during the crisis shows the rationality and practicability of our method. A sensitivity analysis is included to understand how input features contribute to outputs and the observations meet our econometric understanding, showing the most recent features are most relevant.

Chapter 7

Conclusions

This chapter concludes the thesis by restating our contributions and presenting potential directions for future work. A discussion is also included to reflect our thoughts on the usage of deep learning models in financial markets.

7.1 Summary of Contributions

We introduce a hybrid deep neural network and show its ability to predict stock price movements on high-frequency limit order book data. Unlike traditional hand-crafted models, where features from the order book are carefully designed, we utilise a CNN and an Inception Module to automate such feature extraction and use LSTM units to capture time dependencies. The proposed method is evaluated against several baseline methods and we show that our model delivers superior performance with feature extraction implicit as part of the training process. An interesting observation from our work is that the proposed model generalises well not just to out-of-sample data, but also to instruments that did not form part of the training data corpus. This suggests the existence of universal features across order books that are informative for price formation and the ability of models to capture these features. To ensure potential practicality of our model, a simple trading simulation is used, leading to

positive profits that are statistically significant.

In order to obtain uncertainty measures on the variables and decisions, we apply variational dropout. Using variational dropout, an approximation to Bayesian Neural Networks, we show how uncertainty relates to position sizing and how trading performance can be improved by its inclusion, so avoiding bad trades. In addition to such approximate Bayesian networks, we study Quantile Regression to obtain uncertainty bounds for financial return forecasts. We propose a network architecture that can simultaneously estimate multiple return quantiles from both long and short positions by training with different quantile loss functions. We show that returns from long and short positions are statistically different, due to changing spread, and a separate modelling approach for each provides us with valuable, non-stationary, extra information regarding risk exposure.

Due to the difficulty of transforming predictive signals to trade positions, we adopt state-of-art reinforcement learning algorithms including the Deep Q-learning Networks, Policy Gradients and Advantage Actor-Critic to design trading strategies for continuous futures contracts. We study both discrete and continuous action spaces, and utilise features from both time-series momentum and technical indicators to form state representations. Volatility scaling is applied to improve reward functions, allowing us to adjust risk levels based on investors' preferences. We test our model on the 50 most liquid futures contracts from 2011 to 2019, and investigate how performance varies across different asset classes including commodities, equity indices, fixed incomes and FX markets. The experiments show that the proposed algorithms can follow large market trends without changing positions and can also scale down, or hold, through consolidation periods.

The reinforcement learning algorithms are used to trade individual assets, while, in practice, trading a single asset is risky and portfolios are preferred as they lead to higher Sharpe ratios. A well constructed portfolio benefits from diversification

and smoothes out the equity curve, leading to higher return per risk than trading an individual asset. In Chapter 6, we study deep learning models to directly optimise the portfolio Shape ratio. The framework we present circumvents the requirements of forecasting expected returns and allows us to directly optimise portfolio weights using gradient methods. We select Exchange-Traded Funds (ETFs) of market indices to form the portfolio, and compare our method with a variety of baseline algorithms. The testing period is from 2011 to the end of April 2020, including the financial instabilities due to COVID-19. The results show that our model delivers solid performance under different cost rates and risk levels. Finally, we conduct a sensitivity analysis to understand the relevance of input features.

7.2 Discussion

In this section, we include a discussion to reflect our thoughts on the usage of deep learning models in financial markets. We discuss topics including data availability, diversification, model interpretability and ethnics and fairness for machine learning in the financial industry.

In this thesis, we design large-scale deep neural networks for high-frequency limit order book data. This dataset, with millions of observations and high dimensionality, provides us with great flexibility to test large and complex networks with enough samples to calibrate model parameters. Deep learning models are, in general, still very data-hungry and a large dataset also prevents the problem of overfitting to some extent. However, such a reliance on sample size could pose great problems for modelling middle-to-low frequency financial time-series. In terms of data at daily frequency, this problem is mild as we can group securities from different asset classes to form a large training set as long as a proper normalisation scheme is used. However, even if we group securities together, deep networks are inappropriate for modelling

weekly, monthly or quarterly time-series because the resulting dataset will still be very small, without enough observations to calibrate weights. Models that are built from such low-frequency data are often based on economic understanding of markets and features used are derived from the fundamentals of securities. In this case, machine learning algorithms that are data-efficient should be considered, for example, Gaussian process (GP), as a Bayesian non-parametric approach, could be a better alternative and it could be interesting to adapt the GP to classical econometric models such as the Fama and French Model to test model performance.

In Chapters 3 and 4, we place emphasis on designing models that can achieve good prediction performance. These chapters are based on early work of the PhD and, since then, there has been a shift in research focus, from optimising prediction accuracy to optimising trading performance directly (reflected in Chapters 5 and 6). This shift of interest comes from a better understanding of financial markets and, thanks to the work of [149, 76], we realise that the core of systematic trading is to have a model that can deliver good positive expected return and we use proper risk management techniques to explore this positive expectancy while staying in the market. Literature is filled with work on designing prediction models, but a good predictive signal only forms a part of a complete trading strategy and we can equivalently benefit from broad diversification, smart portfolio construction, risk control and cost-effective techniques. For example, volatility scaling is used in our thesis to control for volatility and maximum drawdown, and the work of [145] discusses machine learning models to protect tail-risk. Instead of emphasising the importance of prediction models, we encourage more research in these directions.

In addition, research on model interpretability should be encouraged, both in academia and industry. In this thesis, we briefly tackle this problem by using LIME and sensitivity analysis to understand components of input that are active to objective functions. However, we use these methods mostly for overview purposes and very

limited knowledge can be drawn from these analysis to better understand the decision making process of deep neural networks or the dynamics of financial markets. Such an understanding would be imperative as the deployment of deep learning models is rapidly increasing in financial industry. Investors need to be better informed of model behaviours to prevent events such as "flash clash", where models take aggressive short-selling positions that can cause great damage to vulnerable retail traders without enough capital support.

Lastly, we include a broad discussion on the ethics and fairness of using machine learning algorithms in the financial industry. Machine learning models have not just been used for trading but also been deployed for commercial banking services. For example, there has been active usage of models for credit rating and lending, such applications have unique requirements for fairness and there are strong ethical reasons to make sure that models adopted are unbiased and fair. Also, regulation needs to be made to check model bias and assessment of fairness should be undertaken to make sure that models do not discriminate against groups that are vulnerable due to historical prejudice and injustices. Fortunately, we have seen progress being made on these topics in recent years. At NeurIPS 2020, a workshop on Fair AI in Finance was held to discuss fairness and ethics for deploying machine learning algorithms in the financial industry, and the works of [178, 36] propose measures for fairness in finance. We hope to see more of these discussions in future.

7.3 Future Work

In this section, we discuss some directions for future research work.

7.3.1 A Convolutional Layer for Time Series

We note that convolution filters used in any CNN architecture are actually finite impulse response (FIR) filters in linear time-invariant (LTI) systems [126]. A FIR filter is an operation where the value of the output signal $y(n)$ at any time depends only on a finite number of observed values of the input signal $x(n)$. We can write any FIR filter in the following form:

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k), \tag{7.1}$$

where M is the filter order, $x(n)$ is the observation and b_k is the filter coefficient. For FIR systems, the filter coefficients are the values of the impulse response signals. As we discussed in Chapter 3, this operation has several appealing properties for processing time-series data, e.g. offering both smoothing and equivariance to translation.

However, standard CNNs are mostly designed for image-processing tasks rather than time-series. For example, key features of an image, such as the nose or tail of a dog, are located separately in an image without path-dependencies, while features from time-series data are clearly time dependent with information propagating from the past to present. In other words, the existence of representative features is more important than their locations for image recognition problems as the tail or nose of a dog can be arranged in a number of ways in an image. The transition between these features is to some extent irrelevant, while for time-series data, features are time-ordered and information from all points in the past can, in principle, contribute to inference at the current time instance. FIR filters, by definition, do not consider information pertaining to past observations or responses outside the filter order (the width). As a result, a standard convolutional layer is not ideal for processing time-series data and there is a resultant need for a neural layer that not only inherits properties from convolution, but also considers time dependencies among feature maps.

Another class of LTI system is the infinite impulse response (IIR) filter. The IIR filter is generally represented in recursive form, such that $y(n)$ is a linear combination of past responses as well as observations:

$$y(n) = \sum_{l=1}^{N} a_l y(n-l) + \sum_{k=0}^{M-1} b_k x(n-k), \tag{7.2}$$

where a_l are the recursive coefficients, b_k the non-recursive coefficients and (M, N) are the filter orders. Note that we can obtain FIR filters from this general form by setting the recursive coefficients to zero. With an IIR component, past output responses contribute to current estimates, capturing some element of the "infinite memory" of a signal.

One possible extension would be replace the FIR filters used in CNNs with IIR filters. We note that IIR filters usually require fewer coefficients than FIR filters to execute similar filtering performance, leading to fast operation and less memory requirements. However, we do need to ensure Bounded Input Bounded Output (BIBO) stability for IIR systems. There is an interest to design such a convolutional layer for processing time-series data and we can study how stability conditions can be met during the training process.

7.3.2 Sequential Adaptation of Modelling

In this thesis, we adopt a static training scheme implying that, once a model is fully trained, we do not update model parameters during testing periods. However, financial time-series typically experience regime-shifts. The non-stationary and nonlinear nature of such data makes the static training approach inadequate. A natural extension of future work would be to research online learning models where the parameters are updated in real time.

A good starting point of doing this would be linear state space models [118] where

an observed time series is a linear function of a hidden state vector and the hidden state vector operates first-order vector autoregression:

$$\begin{aligned} \boldsymbol{y}_t &= \boldsymbol{F}_t \boldsymbol{x}_t + \boldsymbol{v}_t, \quad \boldsymbol{v}_t \sim N(0, \boldsymbol{V}_t), \\ \boldsymbol{x}_t &= \boldsymbol{G}_t \boldsymbol{x}_{t-1} + \boldsymbol{w}_t, \quad \boldsymbol{w}_t \sim N(0, \boldsymbol{W}_t), \end{aligned} \tag{7.3}$$

where $\boldsymbol{y}_t \in \mathbb{R}^p$ represents p observations at time t and $\boldsymbol{x}_t \in \mathbb{R}^m$ contains the unobserved states of the system that depends on previous states through a linear operation $\boldsymbol{G}_t \in \mathbb{R}^{m \times m}$. For time-series data, we can think of $\boldsymbol{x}_t$ containing different components of the process such as trend or seasonality, and an observation operator $\boldsymbol{F}_t \in \mathbb{R}^{p \times m}$ transforms the states to observations. Both observation and transition equations have Gaussian errors with covariance matrices $\boldsymbol{V}_t$ and $\boldsymbol{W}_t$.

The Kalman filter [82] can be utilised to compute the above models with exact Gaussian likelihood functions. While, we would like to extend the state space framework to nonlinear models and non Gaussian errors. The nonlinearity can be explored in two ways: the extended Kalman filter (EKF) [37] and the Unscented Kalman filter (UKF) [161]. In EKF, both the state transition and observation models can be nonlinear as long as these functions are differentiable since a matrix of partial derivatives is needed. However, EKF can give particularly poor performance if transition and observation functions are highly nonlinear [81]. The UKF operates a sampling method known as the unscented transformation (UT) that selects a set of sample points (sigma points) to calculate the mean and covariance. This method does not require the calculation of Jacobians that are difficult for complex systems and the work of [64] suggests that the UKF yields better results than the EKF in certain systems.

In addition to nonlinearity, we can relax the assumptions of Gaussian distributions by using Dynamic Generalised Linear Model (DGLM) [166, 165]. As an example, the work of [97] proposes sequential dynamic classification algorithms that are based on Bayesian Kalman processes and DGLM. Their results suggest the ability of the

model to track changes in the underlying decision surfaces, and we can potentially incorporate these techniques to model financial time-series and make predictions in a dynamic way so the parameters of the model will not be fixed during the testing time.

Appendix A

Pinnacle Dataset

The dataset consists of 50 futures contracts, and there are 25 commodity contracts, 11 equity index contracts, 5 fixed income contracts and 9 Foreign Exchange (FX) contracts. A detail description of each contract is below.

Ticker	Contract Details	Ticker	Contract Details
Commodities		**Equity Indexes**	
CC	COCOA	CA	CAC40 INDEX
DA	MILK III, Comp	EN	NASDAQ, MINI
GI	ORANGE JUICE	ER	RUSSELL 2000, MINI
JO	COFFEE	ES	S & P 500, MINI
KW	WHEAT, KC	LX	FTSE 100 INDEX
LB	LUMBER	MD	S&P 400 (Mini Electronic)
NR	ROUGH RICE	SC	S & P 500, Composite
SB	SUGAR #11	SP	S & P 500, Day Session
ZA	PALLADIUM, Electronic	XU	DOW JONES EURO
ZC	CORN, Electronic	XX	DOW JONES STOXX 50
ZF	FEEDER CATTLE, Electronic	YM	Mini Dow Jones ($5.00)
ZG	GOLD, Electronic	**Fixed Incomes**	
ZH	HEATING OIL, Electronic	DT	EURO BOND (BUND)
ZI	SILVER, Electronic	FB	T-NOTE, 5-year Composite
ZK	COPPER, Electronic	TY	T-NOTE, 10-year Composite
ZL	SOYBEAN OIL, Electronic	UB	EURO BOBL
ZN	NATURAL GAS, Electronic	US	T-BONDS, Composite
ZO	OATS, Electronic	**Foreign Exchange**	
ZP	PLATINUM, Electronic	AN	AUSTRALIAN, Day Session
ZR	ROUGH RICE, Electronic	BN	BRITISH POUND
ZT	LIVE CATTLE, Electronic	CN	CANADIAN, Composite
ZU	CRUDE OIL, Electronic	DX	US DOLLAR INDEX
ZW	WHEAT, Electronic	FN	EURO, Composite
ZZ	LEAN HOGS, Electronic	JN	JAPANESE YEN
		MP	MEXICAN PESO
		NK	NIKKEI INDEX
		SN	SWISS FRANC

Appendix B

Additional Results

Table B.1 presents the performance metrics for portfolios without additional layer of portfolio-level volatility scaling in Chapter 5.4.4.

Table B.1: Experiment Results for the Raw Signal in Chapter 5.4.4.

	E(R)	**Std(R)**	**DD**	**Sharpe**	**Sortino**	**MDD**	**% of + Ret**	**$\frac{\text{Ave. P}}{\text{Ave. L}}$**
				Commdity				
Long	-0.298	0.412	0.258	-0.723	-1.152	0.248	0.473	0.987
Sign(R)	0.101	0.312	0.185	0.325	0.548	0.082	0.494	1.081
MACD	-0.039	0.227	0.136	-0.174	-0.290	0.132	0.486	1.024
DQN	0.187	0.301	0.173	0.623	1.085	0.039	0.498	1.119
PG	0.013	0.287	0.172	0.047	0.078	0.072	0.495	1.026
A2C	0.072	0.163	0.098	0.440	0.729	0.099	0.487	1.151
				Equity Indexes				
Long	0.504	0.928	0.606	0.543	0.831	0.127	0.541	0.928
Sign(R)	0.168	0.799	0.526	0.211	0.319	0.299	0.528	0.928
MACD	-0.068	0.586	0.385	-0.117	-0.178	0.351	0.519	0.904
DQN	0.461	0.933	0.611	0.494	0.754	0.170	0.541	0.922
PG	0.320	0.875	0.574	0.366	0.558	0.211	0.529	0.949
A2C	0.293	0.629	0.427	0.466	0.686	0.193	0.533	0.965
				Fixed Income				
Long	0.605	0.939	0.561	0.645	1.081	0.108	0.515	1.048
Sign(R)	0.189	0.795	0.496	0.237	0.381	0.165	0.504	1.024
MACD	0.136	0.609	0.367	0.224	0.371	0.124	0.485	1.102
DQN	0.734	0.862	0.508	0.851	1.445	0.118	0.515	1.086
PG	0.624	0.938	0.561	0.665	1.113	0.109	0.517	1.043
A2C	0.852	1.345	0.806	0.633	1.057	0.128	0.517	1.039
				FX				
Long	-0.198	0.472	0.285	-0.420	-0.696	0.219	0.491	0.966
Sign(R)	-0.113	0.551	0.341	-0.207	-0.332	0.170	0.499	0.968
MACD	0.016	0.424	0.259	0.037	0.061	0.156	0.493	1.034
DQN	0.272	0.487	0.280	0.560	0.972	0.085	0.510	1.058
PG	0.157	0.533	0.312	0.295	0.503	0.098	0.506	1.029
A2C	0.159	0.455	0.267	0.349	0.592	0.081	0.507	1.034
				All				
Long	-0.013	0.363	0.230	-0.036	-0.057	0.037	0.519	0.919
Sign(R)	0.086	0.296	0.186	0.291	0.461	0.016	0.510	1.008
MACD	-0.018	0.230	0.143	-0.080	-0.129	0.026	0.493	1.013
DQN	0.318	0.252	0.150	1.258	2.111	0.008	0.543	1.041
PG	0.168	0.279	0.174	0.602	0.968	0.011	0.533	0.968
A2C	0.214	0.221	0.134	0.969	1.601	0.009	0.538	1.014

Appendix C

Additional Derivations

We derive how the Sharpe ratio can be updated using gradient ascent in Chapter 6. Recall that the Sharpe ratio, over a trading period of T, is defined as:

$$\begin{aligned} L_T &= \frac{E(R_{p,t})}{\sqrt{E(R_{p,t}^2) - (E(R_{p,t}))^2}}, \\ E(R_{p,t}) &= \frac{1}{T}\sum_{t=1}^{T} R_{p,t}, \\ R_{p,t} &= \sum_{i=1}^{n} w_{i,t-1}(\theta) \cdot r_{i,t}, \end{aligned} \tag{C.1}$$

where $R_{p,t}$ is the realised portfolio return over n assets at time t and $r_{i,t}$ is the return of asset i. We use a neural network with parameters θ to model the allocation ratios and we denote L_T as:

$$L_T = \frac{A}{\sqrt{B - A^2}}, \tag{C.2}$$

where

$$A = \frac{1}{T}\sum_{t=1}^{T} R_{p,t} \quad \text{and} \quad B = \frac{1}{T}\sum_{t=1}^{T} R_{p,t}^2. \tag{C.3}$$

The gradient of L_T with respect to parameters θ can be derived using chain rule:

$$\begin{aligned}
\frac{\partial L_T}{\partial \theta} &= \frac{\partial}{\partial \theta}\Big\{\frac{A}{\sqrt{B-A^2}}\Big\} = \frac{\partial L_T}{\partial A}\frac{\partial A}{\partial \theta} + \frac{\partial L_T}{\partial B}\frac{\partial B}{\partial \theta}, \\
&= \sum_{t=1}^{T}\Big\{\frac{\partial L_T}{\partial A}\frac{\partial A}{\partial R_{p,t}}\frac{\partial R_{p,t}}{\partial \theta} + \frac{\partial L_T}{\partial B}\frac{\partial B}{\partial R_{p,t}}\frac{\partial R_{p,t}}{\partial \theta}\Big\}, \\
&= \sum_{t=1}^{T}\Big\{\frac{\partial L_T}{\partial A}\frac{\partial A}{\partial R_{p,t}} + \frac{\partial L_T}{\partial B}\frac{\partial B}{\partial R_{p,t}}\Big\}\cdot\frac{\partial R_{p,t}}{\partial \theta}, \\
&= \sum_{t=1}^{T}\Big\{\frac{\partial L_T}{\partial A}\frac{\partial A}{\partial R_{p,t}} + \frac{\partial L_T}{\partial B}\frac{\partial B}{\partial R_{p,t}}\Big\}\cdot\Big\{\sum_{i=1}^{n}\frac{\partial R_{p,t}}{\partial w_{i,t-1}}\frac{\partial w_{i,t-1}}{\partial \theta}\Big\},
\end{aligned} \tag{C.4}$$

where $\frac{\partial L_T}{\partial A}$, $\frac{\partial A}{\partial R_{p,t}}$, $\frac{\partial L_T}{\partial B}$, $\frac{\partial B}{\partial R_{p,t}}$ and $\frac{\partial R_{p,t}}{\partial w_{i,t-1}}$ can be easily derived. The form of $\frac{\partial w_{i,t-1}}{\partial \theta}$ depends on the neural network used and it is the gradient of network output with respect to parameters.

Bibliography

[1] JG Agrawal, VS Chourasia, and AK Mittra. State-of-the-art in stock prediction techniques. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2(4):1360–1366, 2013.

[2] Andrew Ang and Geert Bekaert. Stock return predictability: Is it there? *The Review of Financial Studies*, 20(3):651–707, 2007.

[3] Adebiyi A Ariyo, Adewumi O Adewumi, and Charles K Ayo. Stock price prediction using the ARIMA model. In *Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on*, pages 106–112. IEEE, 2014.

[4] Kenneth J Arrow. The theory of risk aversion. *Essays in the theory of risk-bearing*, pages 90–120, 1971.

[5] George S Atsalakis and Kimon P Valavanis. Surveying stock market forecasting techniques–Part II: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941, 2009.

[6] Marco Avellaneda, Josh Reed, and Sasha Stoikov. Forecasting prices from Level-I quotes in the presence of hidden liquidity. *Algorithmic Finance*, 1(1):35–43, 2011.

[7] Philippe Bacchetta, Elmar Mertens, and Eric Van Wincoop. Predictability in financial markets: What do survey expectations tell us? *Journal of International Money and Finance*, 28(3):406–426, 2009.

[8] Michel Ballings, Dirk Van den Poel, Nathalie Hespeels, and Ruben Gryp. Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications*, 42(20):7046–7056, 2015.

[9] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[10] Jamil Baz, Nicolas Granger, Campbell R Harvey, Nicolas Le Roux, and Sandy Rattray. Dissecting investment strategies in the cross section and time series. *Available at SSRN 2695101*, 2015.

[11] Stelios D Bekiros. Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach. *Journal of Economic Dynamics and Control*, 34(6):1153–1170, 2010.

[12] Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Francesco Bertoluzzo and Marco Corazza. Testing different reinforcement learning configurations for financial trading: Introduction and applications. *Procedia Economics and Finance*, 3:68–77, 2012.

[15] Christopher M Bishop. Bayesian neural networks. *Journal of the Brazilian Computer Society*, 4(1), 1997.

[16] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[17] Tim Bollerslev, James Marrone, Lai Xu, and Hao Zhou. Stock return predictability and variance risk premia: Statistical inference and international evidence. *Journal of Financial and Quantitative Analysis*, 49(3):633–661, 2014.

[18] Y Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in vision algorithms. In *Proc. International Conference on Machine learning (ICML'10)*, volume 345, 2010.

[19] Antonio Briola, Jeremy Turiel, and Tomaso Aste. Deep learning modeling of limit order book: A comparative perspective. *arXiv preprint arXiv:2007.07319*, 2020.

[20] Yuri Burlakov, Michael Kamal, and Michele Salvadore. Optimal limit order execution in a simple model for market microstructure dynamics. 2012.

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

[22] Qing Cao, Karyl B Leggio, and Marc J Schniederjans. A comparison between Fama and French's model and artificial neural networks in predicting the Chinese stock market. *Computers & Operations Research*, 32(10):2499–2512, 2005.

[23] Rodolfo C Cavalcante, Rodrigo C Brasileiro, Victor LF Souza, Jarley P Nobrega, and Adriano LI Oliveira. Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*, 55:194–211, 2016.

[24] Tolga Cenesizoglu and Allan Timmermann. Predictability of stock returns: A quantile regression approach. Technical report, Technical report, Technical Report, Cirano, 2007.

[25] Ernest P Chan. *Machine trading: Deploying computer algorithms to conquer the markets*. John Wiley & Sons, 2017.

[26] Ernie Chan. *Quantitative trading: How to build your own algorithmic trading business*, volume 430. John Wiley & Sons, 2009.

[27] Jou-Fan Chen, Wei-Lun Chen, Chun-Ping Huang, Szu-Hao Huang, and An-Pin Chen. Financial time-series data analysis using deep convolutional neural networks. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*, pages 87–92. IEEE, 2016.

[28] Victor Chernozhukov, Iván Fernández-Val, and Alfred Galichon. Quantile and probability curves without crossing. *Econometrica*, 78(3):1093–1125, 2010.

[29] Thomas C Chiang and Jiandong Li. Stock returns and risk: Evidence from quantile. *Journal of Risk and Financial Management*, 5(1):20–58, 2012.

[30] Eunsuk Chong, Chulwoo Han, and Frank C Park. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187–205, 2017.

[31] Pinnacle Data Corp. CLC Database. `https://pinnacledata2.com/clc.html`, Accessed: 2019-08-11.

[32] An example of convolutional network. `https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac`, Accessed: 2020-08-11.

[33] Adam D Cobb, Stephen J Roberts, and Yarin Gal. Loss-calibrated approximate inference in Bayesian neural networks. *arXiv preprint arXiv:1805.03901*, 2018.

[34] Rama Cont and De Nitions. Statistical properties of financial time series. 1999.

[35] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[36] Sanjiv Das, Michele Donini, Jason Gelman, Kevin Haas, Mila Hardt, Jared Katzman, Krishnaram Kenthapadi, Pedro Larroy, Pinar Yilmaz, and Bilal Zafar. Fairness measures for machine learning in finance.

[37] Frederick E. Daum. *Extended Kalman Filters*, pages 411–413. Springer London, London, 2015.

[38] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[39] Luca Di Persio and Oleksandr Honchar. Artificial neural networks architectures for stock price prediction: Comparisons and applications. *International Journal of Circuits, Systems and Signal Processing*, 10:403–413, 2016.

[40] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[41] Matthew Dixon. Sequence classification of the limit order book using recurrent neural networks. *Journal of computational science*, 24:277–286, 2018.

[42] Jonathan Doering, Michael Fairbank, and Sheri Markose. Convolutional neural networks applied to high-frequency market microstructure forecasting. In *Com-*

puter Science and Electronic Engineering (CEEC), 2017, pages 31–36. IEEE, 2017.

[43] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[44] Eugene F Fama and Kenneth R French. Common risk factors in the returns on stocks and bonds. *Journal of*, 1993.

[45] M Todd Farrell and Andrew Correa. Gaussian process regression models for predicting stock trends. *Relation*, 10:3414, 2007.

[46] Daniel Fernholz and Ioannis Karatzas. On optimal arbitrage. *The Annals of Applied Probability*, pages 1179–1204, 2010.

[47] Daniel Fernholz, Ioannis Karatzas, et al. Optimal arbitrage under model uncertainty. *The Annals of Applied Probability*, 21(6):2191–2225, 2011.

[48] E Robert Fernholz. Stochastic portfolio theory. In *Stochastic Portfolio Theory*, pages 1–24. Springer, 2002.

[49] Robert Fernholz. Portfolio generating functions. In *Quantitative Analysis in Financial Markets: Collected Papers of the New York University Mathematical Finance Seminar*, pages 344–367. World Scientific, 1999.

[50] Robert Fernholz and Ioannis Karatzas. Stochastic portfolio theory: An overview. *Handbook of numerical analysis*, 15:89–167, 2009.

[51] Robert Fernholz, Ioannis Karatzas, and Constantinos Kardaras. Diversity and relative arbitrage in equity markets. *Finance and Stochastics*, 9(1):1–27, 2005.

[52] Miguel A Ferreira and Pedro Santa-Clara. Forecasting stock market returns: The sum of the parts is more than the whole. *Journal of Financial Economics*, 100(3):514–537, 2011.

[53] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[54] Thomas G Fischer. Reinforcement learning in financial markets - A survey. Technical report, FAU Discussion Papers in Economics, 2018.

[55] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

[56] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[57] Gary L Gastineau. Exchange-traded funds. *Handbook of finance*, 1, 2008.

[58] Jim Gatheral and Roel CA Oomen. Zero-intelligence realized variance estimation. *Finance and Stochastics*, 14(2):249–283, 2010.

[59] Siddartha Ghoshal and Steve Roberts. Extracting predictive information from heterogeneous data streams using Gaussian processes. *Algorithmic Finance*, 5(1-2):21–30, 2016.

[60] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[61] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[64] Fredrik Gustafsson and Gustaf Hendeby. Some relations between extended and unscented kalman filters. *IEEE Transactions on Signal Processing*, 60(2):545–555, 2011.

[65] Larry Harris. *Trading and exchanges: Market microstructure for practitioners*. Oxford University Press, USA, 2003.

[66] Larry Harris. Maker-taker pricing effects on market quotations. *USC Marshall School of Business Working Paper. Avalable at http://bschool. huji. ac. il/. upload/hujibusiness/Maker-taker. pdf*, 2013.

[67] Campbell R Harvey, Edward Hoyle, Russell Korgaonkar, Sandy Rattray, Matthew Sargaison, and Otto Van Hemert. The impact of volatility targeting. *The Journal of Portfolio Management*, 45(1):14–33, 2018.

[68] Hado V Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[69] W Keith Hastings. Monte carlo sampling methods using Markov chains and their applications. 1970.

[70] Terrence Hendershott, Charles M Jones, and Albert J Menkveld. Does algorithmic trading improve liquidity? *The Journal of Finance*, 66(1):1–33, 2011.

[71] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[72] Nils Lid Hjort, Chris Holmes, Peter Müller, and Stephen G Walker. *Bayesian nonparametrics*, volume 28. Cambridge University Press, 2010.

[73] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[74] Chien Yi Huang. Financial Trading as a Game: A Deep Reinforcement Learning Approach. *arXiv preprint arXiv:1807.02787*, 2018.

[75] John C Hull. *Options futures and other derivatives*. Pearson Education India, 2003.

[76] Antti Ilmanen. *Expected returns: An investor's guide to harvesting market rewards*. John Wiley & Sons, 2011.

[77] Jonathan E Ingersoll. *Theory of financial decision making*, volume 3. Rowman & Littlefield, 1987.

[78] Aya Abdelsalam Ismail, Mohamed Gunady, Héctor Corrada Bravo, and Soheil Feizi. Benchmarking Deep Learning Interpretability in Time Series Predictions. *arXiv preprint arXiv:2010.13924*, 2020.

[79] Johan Ludwig William Valdemar Jensen et al. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30:175–193, 1906.

[80] Olivier Jin and Hamza El-Saawy. Portfolio management using reinforcement learning. Technical report, Working paper, Stanford University, 2016.

[81] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[82] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[83] Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert systems with Applications*, 38(5):5311–5319, 2011.

[84] Alec N Kercheval and Yuan Zhang. Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329, 2015.

[85] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

[86] Samuel Kessler, Arnold Salas, Vincent WC Tan, Stefan Zohren, and Stephen Roberts. Practical Bayesian Learning of Neural Networks via Adaptive Optimisation Methods. *arXiv preprint arXiv:1811.03679*, 2018.

[87] Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam. *arXiv preprint: 1806.04854*, 2018.

[88] Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

[89] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations 2015*, 2015.

[90] Donald E Kirk. *Optimal control theory: An introduction*. Courier Corporation, 2012.

[91] Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of Economic Perspectives*, 15(4):143–156, 2001.

[92] Gary Koop. *Bayesian econometrics*. John Wiley & Sons, 2003.

[93] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[94] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[96] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[97] Seung Min Lee and Stephen J Roberts. Sequential dynamic classification using latent variable models. *The Computer Journal*, 53(9):1415–1429, 2010.

[98] Hailin Li, Cihan H Dagli, and David Enke. Short-term stock market timing prediction under reinforcement learning schemes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 233–240. IEEE, 2007.

[99] Bryan Lim and Stefan Zohren. Time Series Forecasting With Deep Learning: A Survey. *arXiv preprint arXiv:2004.13408*, 2020.

[100] Bryan Lim, Stefan Zohren, and Stephen Roberts. Enhancing Time-Series Momentum Strategies Using Deep Neural Networks. *The Journal of Financial Data Science*, 2019.

[101] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *International Conference on Learning Representations*, 2014.

[102] Alexander Lipton, Umberto Pesavento, and Michael G Sotiropoulos. Trade arrival dynamics and quote imbalance in a limit order book. *arXiv preprint arXiv:1312.0514*, 2013.

[103] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[104] Benoit Mandelbrot and Richard L Hudson. *The Misbehavior of Markets: A fractal view of financial turbulence*. Basic books, 2007.

[105] Benoit B Mandelbrot. How fractals can explain what's wrong with Wall Street. *Scientific American*, 15(9):2008, 2008.

[106] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[107] Frank J Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[108] Loukia Meligkotsidou, Ekaterini Panopoulou, Ioannis D Vrontos, and Spyridon D Vrontos. A quantile regression approach to equity premium prediction. *Journal of Forecasting*, 33(7):558–576, 2014.

[109] Robert C Merton. Lifetime portfolio selection under uncertainty: The continuous-time case. *The review of Economics and Statistics*, pages 247–257, 1969.

[110] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[111] Hrushikesh Narhar Mhaskar and Charles A Micchelli. How to choose an activation function. In *Advances in Neural Information Processing Systems*, pages 319–326, 1994.

[112] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[113] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *NIPS Deep Learning Workshop 2013*, 2013.

[114] Mohammad Mojaddady, Moin Nabi, and Shahram Khadivi. Stock market prediction using twin Gaussian process regression. *International Journal for Advances in Computer Research (JACR) preprint*, 2011.

[115] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[116] John Moody, Lizhong Wu, Yuansong Liao, and Matthew Saffell. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6):441–470, 1998.

[117] Tobias J Moskowitz, Yao Hua Ooi, and Lasse Heje Pedersen. Time series momentum. *Journal of financial economics*, 104(2):228–250, 2012.

[118] Howard Musoff and Paul Zarchan. *Fundamentals of Kalman filtering: A practical approach*. American Institute of Aeronautics and Astronautics, 2009.

[119] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[120] Radford M Neal. Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer, 1992.

[121] Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

[122] David MQ Nelson, Adriano CM Pereira, and Renato A de Oliveira. Stock market's price movement prediction with LSTM neural networks. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1419–1426. IEEE, 2017.

[123] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[124] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37(8):852–866, 2018.

[125] Maureen O'Hara. *Market microstructure theory*, volume 108. Blackwell Publishers Cambridge, MA, 1995.

[126] Sophocles J Orfanidis. *Introduction to signal processing*. Prentice-Hall, Inc., 1995.

[127] Christine A Parlour and Duane J Seppi. Limit order markets: A survey. *Handbook of financial intermediation and banking*, 5:63–95, 2008.

[128] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Temporal bag-of-features learning for predicting mid price movements using high frequency limit order book data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

[129] Jigar Patel, Sahil Shah, Priyank Thakkar, and Ketan Kotecha. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert systems with applications*, 42(1):259–268, 2015.

[130] Dejan Petelin, Jan Šindelář, Jan Přikryl, and Juš Kocijan. Financial modeling using Gaussian process models. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 2, pages 672–677. IEEE, 2011.

[131] John W Pratt. Risk aversion in the small and in the large. In *Uncertainty in Economics*, pages 59–79. Elsevier, 1978.

[132] David E Rapach, Jack K Strauss, and Guofu Zhou. Out-of-sample equity premium prediction: Combination forecasts and links to the real economy. *The Review of Financial Studies*, 23(2):821–862, 2010.

[133] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

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

[135] Gordon Ritter. Machine learning for trading. 2017.

[136] Stephen Roberts, Michael Osborne, Mark Ebden, Steven Reece, Neale Gibson, and Suzanne Aigrain. Gaussian processes for time-series modelling. *Phil. Trans. R. Soc. A*, 371(1984):20110550, 2013.

[137] Ioanid Rosu et al. Liquidity and information in order driven markets. Technical report, 2010.

[138] Johannes Ruf. Hedging under arbitrage. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 23(2):297–317, 2013.

[139] Yves-Laurent Kom Samo and Alexander Vervuurt. Stochastic portfolio theory: A machine learning perspective. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 657–665, 2016.

[140] Nonita Sharma and Akanksha Juneja. Combining of random forest estimates using lsboost for stock market index prediction. In *2017 2nd International Conference for Convergence in Technology (I2CT)*, pages 1199–1202. IEEE, 2017.

[141] William F Sharpe. The Sharpe Ratio. *Journal of Portfolio Management*, 21(1):49–58, 1994.

[142] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[143] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: Perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, 2019.

[144] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

[145] Bruno Spilak and Wolfgang K Härdle. Tail-risk protection: Machine Learning meets modern Econometrics. *Available at SSRN 3714632*, 2020.

[146] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[147] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[148] Zhiyong Tan, Chai Quek, and Philip YK Cheng. Stock trading with cycles: A financial application of ANFIS and reinforcement learning. *Expert Systems with Applications*, 38(5):4741–4755, 2011.

[149] Van K Tharp, Christian Chabot, and K Tharp. *Trade your way to financial freedom*. McGraw-Hill, 2007.

[150] Ludan Theron and Gary Van Vuuren. The maximum diversification investment strategy: A portfolio performance comparison. *Cogent Economics & Finance*, 6(1):1427533, 2018.

[151] Dat Thanh Tran, Moncef Gabbouj, and Alexandros Iosifidis. Multilinear class-specific discriminant analysis. *Pattern Recognition Letters*, 100:131–136, 2017.

[152] Dat Thanh Tran, Alexandros Iosifidis, Juho Kanniainen, and Moncef Gabbouj. Temporal attention-augmented bilinear network for financial time-series data analysis. *IEEE transactions on neural networks and learning systems*, 2018.

[153] Dat Thanh Tran, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Tensor representation in high-frequency financial data for price change prediction. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on*, pages 1–7. IEEE, 2017.

[154] I-Chun Tsai. The relationship between stock price index and exchange rate in asian markets: A quantile regression approach. *Journal of International Financial Markets, Institutions and Money*, 22(3):609–621, 2012.

[155] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *Business Informatics (CBI), 2017 IEEE 19th Conference on*, volume 1, pages 7–12. IEEE, 2017.

[156] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning to detect price change indications in financial markets. In *Signal Processing Conference (EUSIPCO), 2017 25th European*, pages 2511–2515. IEEE, 2017.

[157] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing*, page 106401, 2020.

[158] Dimitris G Tzikas, Aristidis C Likas, and Nikolaos P Galatsanos. The variational approximation for Bayesian inference. *IEEE Signal Processing Magazine*, 25(6):131–146, 2008.

[159] Dimitris G. Tzikas, Aristidis C. Likas, and Nikolaos P. Galatsanos. The variational approximation for Bayesian inference. *IEEE Signal Processing Magazine*, 25(31), 2008.

[160] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[161] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[162] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013.

[163] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 1995–2003. JMLR. org, 2016.

[164] Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker. Model-based reinforcement learning for predictions and control for limit order books. *arXiv preprint arXiv:1910.03743*, 2019.

[165] Mike West and Jeff Harrison. *Bayesian forecasting and dynamic models*. Springer Science & Business Media, 2006.

[166] Mike West, P Jeff Harrison, and Helio S Migon. Dynamic generalized linear models and Bayesian forecasting. *Journal of the American Statistical Association*, 80(389):73–83, 1985.

[167] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[168] Russell Wild. *Index Investing for Dummies*. John Wiley & Sons, 2008.

[169] J Welles Wilder. *New concepts in technical trading systems*. Trend Research, 1978.

[170] Christopher KI Williams. Computing with infinite networks. In *Advances in neural information processing systems*, pages 295–301, 1997.

[171] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[172] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[173] Ting-Kam Leonard Wong. Optimization of relative arbitrage. *Annals of Finance*, 11(3-4):345–382, 2015.

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

[175] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[176] Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*, 2019.

[177] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[178] Yukun Zhang and Longsheng Zhou. Fairness assessment for artificial intelligence in financial industry. *arXiv preprint arXiv:1912.07211*, 2019.

[179] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep learning for portfolio optimisation. *The Journal of Financial Data Science*.

[180] Zihao Zhang, Stefan Zohren, and Stephen Roberts. BDLOB: Bayesian Deep Convolutional Neural Networks for Limit Order Books. In *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*, 2018.

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

[182] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Extending Deep Learning Models for Limit Order Books to Quantile Regression. *Proceedings of Time Series Workshop of the 36 th International Conference on Machine Learning, Long Beach, California, PMLR 97*, 2019.

[183] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40, 2020.

[184] Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78, 1988.

[185] Eric Zivot. *Introduction to computational finance and financial econometrics*. Chapman & Hall Crc, 2017.

[186] Eric Zivot and Jiahui Wang. Vector autoregressive models for multivariate time series. *Modeling Financial Time Series with S-PLUS®*, pages 385–429, 2006.