Deep Learning of the Order Flow for Modelling Price Formation

Ye-Sheen Lim

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

January 24, 2022

I, Ye-Sheen Lim, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

The objective of this thesis is to apply deep learning to order flow data in novel ways, in order to improve price prediction models, and thus improve on current deep price formation models. A survey of previous work in the deep modelling of price formation revealed the importance of utilising the order flow for the deep learning of price formation had previously been over looked. Previous work in the statistical modelling of the price formation process in contrast has always focused on order flow data. To demonstrate the advantage of utilising order flow data for learning deep price formation models, the thesis first benchmarks order flow trained Recurrent Neural Networks (RNNs), against methods used in previous work for predicting directional mid-price movements. To further improve the price modelling capability of the RNN, a novel deep mixture model extension to the model architecture is then proposed. This extension provides a more realistically uncertain prediction of the mid-price, and also jointly models the direction and size of the mid-price movements. Experiments conducted showed that this novel architecture resulted in an improved model compared to common benchmarks. Lastly, a novel application of Generative Adversarial Networks (GANs) was introduced for generative modelling of the order flow sequences that induce the mid-price movements. Experiments are presented that show the GAN model is able to generate more realistic sequences than a well-known benchmark model. Also, the mid-price time-series resulting from the SeqGAN generated order flow is able to better reproduce the statistical behaviour of the real mid-price time-series.

Impact Statement

The modern financial securities market is a complex system comprising of networks of dependencies, competitions, relationships, and other types of interactions. The complexity of such systems make price prediction challenging since the emerging price formation processes are intrinsically difficult to model.

Though deep learning has been applied to learn price prediction, the first work in this thesis demonstrates predictions are improved when the deep price formation models are trained on order flow data. The improved predictions, would provide practitioners with a more reliable model for trading. Also, a method for interrogating the black-box model to study the price formation process is introduced here, providing researchers with an alternative data-driven view of the inner-working of the market microstructure.

The second work of the thesis further improves the price prediction capability of the deep order flow model. By introducing a novel architecture, the model is given the ability to produce probabilistic forecasts and jointly model the direction and size of price movements. Such probabilistic forecasts are a more realistic representation of uncertain risky markets, in addition to being necessary for the important aspect of risk management in trading strategies. The joint modelling of direction and size improves the prediction of both target variables. As opposed to previous work that only predicts the direction, the novel architecture here provides practitioners with an improved indicator of how to place trades.

The final work of the thesis opens up a new avenue of research by introducing the concept of deep generative modelling of order flow sequences.

Order flow sequences generated by the models can be used to predict prices in a bottom-up approach. However, generative models of the order flow are not limited to just price predictions. Practitioners can use the generated sequences as predictions of how the order flow will evolve, allowing the anticipation of order intensities, which can be used to compute market making or order execution policies. Other possible applications for the generated order flow include, though are not limited to, evaluating trading strategies, or as synthetic data for training models. In addition, such generative modelling of the order flow using deep learning is a new problem area to be explored by academic researchers.

Acknowledgements

First and foremost I would like to express my sincere gratitude to my supervisor, Dr. Denise Gorse, for her invaluable supervision, continuous support, and patience during my PhD study. Additionally, I would like to express gratitude to UCL Computer Science for the generous funding and an opportunity to undertake my PhD study with the Financial Computing and Analytics research group.

On the personal side, I am deeply grateful to my parents for their endless support in my studies, and for setting the early foundations for my intellectual pursuits. And finally to Ana, for whom have been a source of tremendous love, understanding and encouragement in the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of this thesis by discussing the motivations behind the research, and stating the objectives and contributions of the research, and the structure of this thesis. The chapter opens with an introduction to order-driven markets and the order flow, and discusses the importance of the order flow in modelling price formation. Then, it briefly discusses previous work in modelling price formation using deep learning, and why further research is needed. The chapter is concluded on the research objectives and literature contributions of the work in this thesis, and the structure of the thesis.

1.1 Motivations from the Literature and Industry

The modern financial securities market is a complex dynamical system [1] comprising of networks of dependencies, competitions, relationships, and other types of interactions, between heterogeneous traders, and explicit rules, regulations and trading mechanisms. The complexity of such systems makes the development of trading models challenging since emerging features of *market microstructure* such as the *price formation process* from the collective behaviour of these complex interactions, when sufficiently large, are intrinsically difficult to model. The main motivation for the research in this thesis is to investigate the modelling potential of the deep learning, using market data at the microstructural level, for potentially untangling this complexity and improve the learning of

the price formation process and other features of the market microstructure.

Financial securities markets are organised as one of two kinds of market systems for facilitating trade, *order-driven* markets and *quote-driven* market. In order-driven markets, all traders trade directly against one another. In quote-driven markets, regular traders trade against designated market makers, dealers, or specialists who quote their buy and sell prices. More than half of the equity and derivative markets in today's high-speed digital world have converged to a common system, which is the order-driven market. Some of the world's largest equity exchanges such as the Hong Kong, Shanghai, Shenzhen, London and Toronto Stock Exchange, the Euronext, and also the exchanges of the Japan Exchange Group, operate as pure order-driven markets. NASDAQ and the NYSE, which are the largest equity exchanges globally, operate a hybrid market system that incorporates the attributes of both order-driven and quote-driven markets.

Traders in order-driven markets trade against one another by submitting *market orders* and *limit orders* to the exchange. A limit order is a type of order to buy or sell a specific quantity at a specified price or better. Although the price is guaranteed, the filling of a limit order is not. If a submitted limit order cannot be immediately executed against an existing order in the *limit order book*, then the limit order joins the price queue in the order book until it is cancelled, amended, or executed against subsequent orders. Market orders, on the other hand, are immediately executed against limit orders queued at the best price in the order book, as fully as possible, and hence cannot be cancelled. Any unfilled portion may then be converted to limit orders at the same price, or executed at the next best available prices until the market order is fully executed. The dynamic placement and cancellation of limit orders and market orders by traders assembles into a stream that forms the sequence of orders called a *order flow*, which is the featured market data used for the work in this thesis.

Order flow data provides the most granular view of market activity, potentially encoding information much needed to unravel the aforementioned com-

plex network of interactions that gives rise to different features of market microstructure. For these reasons, the study of order flow data has for many years inspired a great deal of quantitative modelling research. These studies [2, 3, 4, 5, 6, 7, 8, 9, 10, 11] assume that the arrival of events in the order flow is governed by stochastic processes. From the point of view of the industry, the study of order flow data would also lead to improved price forecasts and trading costs estimation, and thus improve the profitability of trading models [12, 13, 14, 15, 16, 17]. Though the use of stochastic processes to model the arrival of the order flow data, to study price formation process, has shown impressive results in practice, these approaches are not extensively data-driven, and rely on fitting closed-form theory-driven models. This leads to major drawbacks such as sensitivity to regime shifts, intractability, and lack of generalisation power. The application of machine learning to order flow data, as carried out in this thesis, can lead to improvements in modelling the different features of market microstructure as machine learning models are generally more resistant to these drawbacks.

Deep learning has in recent years been the prevailing class of machine learning algorithm, and has displayed ground-breaking performance in application domains such as computer vision [18, 19, 20, 21, 22] and natural language processing [23, 24, 25, 26, 27]. However, there has been substantially less work on econometric subjects, especially in the area of quantitative finance. Deep learning has been applied to the learning of the price formation process [28, 29, 30, 31, 30, 32, 33], though these previous works have not considered utilising the order flow itself in the modelling of the price formation process. However, much work in quantitative finance indicates that, by utilising the order flow, deep learning could provide an important window into the price formation process. Therefore, the main objective of the research in this thesis is to extend the literature by introducing novel deep learning models that use order flow data to improve price prediction models, and thus improve the current state of data-driven modelling of the price formation process. This main objective is broken

down into three parts, as outlined in the following section.

1.2 Research Objectives

The first objective of the research in this thesis is to investigate the use of order flow data as inputs to deep learning models for improving the ability to learn the price formation process. This objective is motivated by the current absence of any machine learning study utilising the informational potential of the order flow for price predictions, even though order flow data is becoming very readily available in many exchanges. As evident from many research studies, the sequence of events in the order flow is one of the main driving factors of the price formation process. To achieve this objective, first an experiment is performed to evaluate the performance of a deep recurrent neural network, trained on order flow data, for predicting directional mid-price movements, compared to approaches in previous works that are not. Then, a study of the factors affecting the price formation process is conducted by using a black-box interpretation algorithm to discover effect of each feature of the order flow, such as the order size, price, and type, on model performance

The second research objective is to improve the prediction capabilities of deep recurrent neural networks by introducing a novel architecture that would allow the network to produce probabilistic forecasts, and also jointly model the direction and size of the price movements. The work here would fill a gap in the applied deep learning literature where currently there is neither any previous work on the probabilistic modelling of price movements, nor any on the joint modelling of direction and size of price movements, even for deterministic models. Due to the inextricable uncertainties in financial markets, the price formation process is never deterministic. Probabilistic price formation models are more realistic representations of real financial markets that are full of risks and uncertainties. In addition, non-deterministic models are also more useful in trading strategies, as these usually depend on uncertainty estimations for risk management. Also, joint modelling of the direction and size of price move-

ments provides a more complete model of the price formation process. Three experiments are carried out to evaluate the novel architecture proposed for this objective. First, the performance of the prediction model implementing the novel architecture is compared against the best available previous work to establish the predictive capability of the new model. Then, the novel architecture is evaluated against an approach that uses two separate networks for forecasting the probabilistic size and direction of price movements, the aim of this experiment being to justify the joint modelling of these outputs in the proposed new architecture. The concluding experiment for the second objective demonstrates the full potential of the novel architecture's probabilistic output and the direction-size joint modelling of the price by implementing a simulated trading scenario.

The third and final research objective is to investigate the deep generative modelling of order flow sequences, with the aim of learning a model that is able to produce realistic sequences. Price changes can then be obtained in a bottom-up way. This builds upon the typical approach in the statistical modelling literature, which model order flow sequences, and study how prices are formed by the arrival of order events. Such generative modelling of order flow sequences is a currently completely novel application of machine learning, and hence fills an important gap in the literature. Besides the use of the simulated sequences to predict price movements, the generative model of the order flow has other useful applications. For example, certain trading strategies can use generated sequences to compute order intensities for estimating some "proxy prices", such as the reservation price in [13], to determine the optimal buy-sell quotes. Additionally, the interrogation of the generative model using a black-box interpreter could provide an interesting data-driven window into the price formation process, and possibly other features of market microstructure. Other useful applications include using the order flow to substitute unavailable historical data for evaluating trading strategies. In order to achieve this final research objective, a framework using deep generative adversarial networks (GANs) for modelling order flow sequences is introduced. The fidelity of the simulated or-

der flow sequences generated by the deep GAN is first evaluated by measuring the sequence similarity between the generated sequences and real sequences. Then, the macro-behaviour of the mid-prices that are dynamically formed by the generated order flow sequences are investigated to determine how well generated sequences reproduce the statistical behaviour and stylised facts in the mid-price time-series that have been reported in existing empirical research.

1.3 Major Contributions

The focus of this thesis is on the development of improved deep learning algorithms and models that utilise the order flow to model price formation, by learning to predict price movements from the order flow sequences. The key contributions of this thesis are as follows.

- Demonstrate the advantages of utilising order flow data to train deep learning models for predicting directional mid-price movements.
- Introduce a novel deep mixture model for joint probabilistic modelling of the direction and size of the mid-price movements.
- Introduce a novel approach using the SeqGAN model [34] for generative modelling of the order flow sequences.

1.4 Thesis Outline

The chapters in this thesis are organised as follows.

Chapter 2 - This chapter presents necessary background for the thesis and related previous work. First, relevant aspects of order-driven markets and the price formation process are introduced. Then, technical background on the relevant deep learning models and algorithms is given. The chapter ends with a survey of previous work on topics related to the research problem of the thesis.

Chapter 3 - This chapter presents an investigation of the potential in utilising order flow data for training deep price formation models. The methodology is first outlined, and then model performance is benchmarked and evaluated.

Finally, an investigation into the order flow factors that contribute to the price formation process is performed using a black-box model interpretation algorithm.

Chapter 4 - This chapter introduces a novel extension to the architecture of Chapter 3 in order to improve the prediction capability of the deep price formation models. The novel architecture is first presented, and then evaluated against suitable benchmark models. The chapter ends with a simulated trading scenario to further illustrate the prediction ability of the introduced model architecture relative to the benchmarks.

Chapter 5 - This chapter introduces the deep generative modelling of order flow sequences. The problem of generative modelling of order flow sequences is defined, and the method for applying the SeqGAN framework to this problem is described. Benchmarking against an order flow model from previous work, the chapter then evaluates the sequence modelling ability of the SeqGAN model. Finally, the chapter compares the statistical behaviour of the mid-price time-series formed from the generated sequences against real data.

Chapter 6 - This is the concluding chapter of the thesis. First, a summary and discussion of the key contributions of the thesis is presented. Afterwards, possible extensions to the work are proposed.

Chapter 2

Background and Related Work

This chapter presents necessary background and a review of related work. For the domain background, concepts related to order-driven markets will be covered. Then, a technical background will be presented which provides descriptions of the deep learning models and algorithms that form the core solutions to the research problems in this thesis. The chapter concludes with a survey and review of previous work in both the machine learning and non-machine-learning literature for modelling price formation using the order flow.

2.1 Order-Driven Markets

In the past, quote-driven markets were the common system adopted by financial marketplaces, where a small number of market makers post the prices at which they are each willing to buy or sell the traded asset. By quoting buy prices lower than sell prices, the market makers profit from the spread between the buy and sell prices, in exchange for taking on the risk of holding inventories of the traded asset and providing liquidity to the market. In such quote-driven markets, buy and sell orders are centralised and are only posted by traders that are designated as market makers. Other traders that wish to trade the asset would have to buy or sell at the prices quoted by these market makers. Nowadays, the largest exchanges in the modern financial market have adopted the *order-driven* market system, which decentralises and democratises the determination

of prices in the market by allowing each trader the option of posting buy or sell *limit orders* which state their desired prices. The NYSE and NASDAQ implement a hybrid system where the prices of the limit orders placed by the traders are shown alongside market maker quotes, but the work in this thesis will only focus on markets that are purely order-driven.

Price formation in an order-driven market is driven by the flow of orders posted to the exchange, the limit order book (LOB) and the matching engine implemented by the exchange. In the following, various terms and concepts related to orders and the LOB relevant to the work of this thesis will be defined, followed by a description of the most common matching engine used which sets the trading rules between orders in an order-driven market. For a more comprehensive treatment of LOB, readers are directed to [35].

2.1.1 The Limit Order Book

A buy (or, sell) limit order x submitted to the exchange at time t_x can be mathematically defined as the tuple $x = (p_x, \omega_x, t_x)$, where $p_x > 0$, and $\omega_x > 0$. The order x represents the commitment of the trader to buy (or, sell) up to ω_x units of the traded asset at a price no greater than p_x (or, no less than p_x).

When a buy (or, sell) limit order x is received by the exchange, the *matching-engine* of a LOB checks if there is a previously submitted sell (or, buy) limit order resting in the LOB to execute x against. If so, the orders are matched and the transaction registered. Otherwise, x will be added to the *limit order book* and rests there until it is either executed against an incoming order, or cancelled. A limit order book $\mathscr{L}(t)$ can then be defined as the set of all resting limit orders at time t. The resting orders in $\mathscr{L}(t)$ can be further partitioned into the *bid*-side $\mathscr{B}(t)$, which is the set of all buy orders, and the *ask*-side $\mathscr{A}(t)$, which is the set of all sell orders.

Orders are grouped by their prices in a LOB. One can think of the LOB as a set of queues for different prices, each containing resting orders sorted by their arrival time t_x. At a given time t, a price on the bid-side can be defined as a pair $(p, n^b(p,t))$, where $n^b(p,t) = \sum_x \omega_x$ for all $x \in \mathscr{B}(t)$ with $p_x = p$. A price

on the ask-side is similarly defined by $(p, n^a(p,t))$ using $\mathscr{A}(t)$. The bid-side (or, ask-side) depth profile for time t is then defined as the set of all ordered pairs $(p, n^b(p,t))$ (or, $(p, n^a(p,t))$). An example of a LOB illustrating the depth profiles for the bid-side and ask-side of the LOB is shown in Figure 2.1. This form of depth profile is often how the LOB is evaluated by traders in practice.

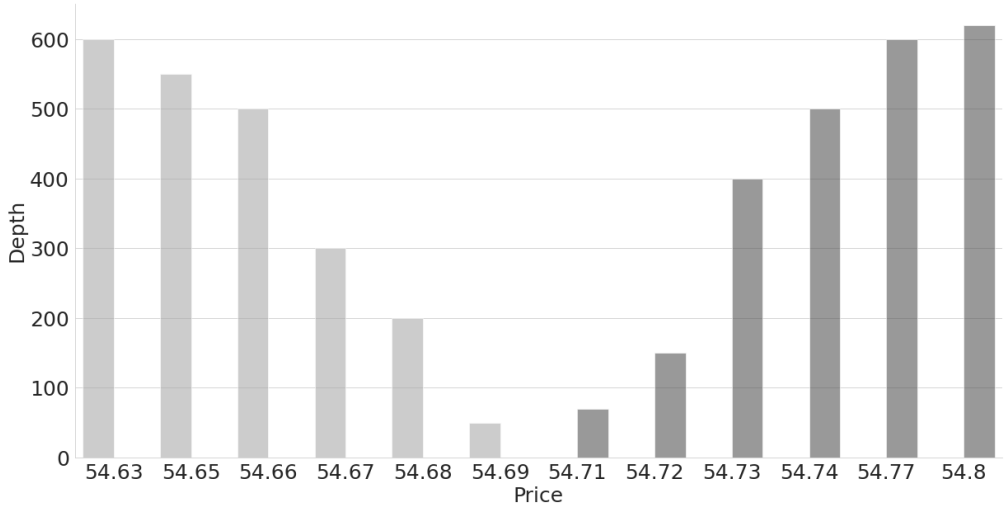


Figure 2.1: A visualisation of the bid-side and ask-side depth profiles in a LOB

There exist both in practice and in the literature a number of common measures of the limit order book, which are also relevant to the work later in this thesis. These can now be defined as follows:

- The *best bid* $b(t)$ at time t is the highest price among the orders resting in the limit order book $b(t) = \max_{x \in \mathscr{B}(t)} p_x$.
- The *best ask* $b(t)$ at time t is the lowest price among the orders resting in the limit order book $a(t) = \min_{x \in \mathscr{A}(t)} p_x$.
- The *bid-ask spread* at time t is $s(t) = a(t) - b(t)$.
- The *mid-price* at time t is $m(t) = \frac{a(t)+b(t)}{2}$

Now that the best bid and best ask have been defined, another type of order a trader can place besides limit orders, the *market order*, can be described. A buy (or, sell) market order $x = (\omega_x, t_x, d_x)$ submitted at time t represents the

commitment of a trader to immediately buy (or, sell) the stated unit of traded asset at the best available prices in the LOB. Any buy (or, sell) market order is always executed immediately against orders resting at $b(t)$ (or, $a(t)$) as fully as possible. If in the LOB $n^b(b(t),t)$ (or, $n^b(a(t),t)$) is smaller than the size of the market order, then, the best bid (or, best ask) is depleted and the remaining portion is filled at the new best bid (or, best ask) until the order is fully executed. Unlike limit orders, the execution price is not guaranteed when submitting market orders, though some exchanges may implement rules to convert market orders into resting limit orders before they are fully executed. Traders can also post marketable buy (or, sell) limit orders, which are orders with $p_x \geq a(t)$ (or, $p_x \leq b(t)$). In modelling, these are generally treated as market orders since they consume liquidity from the LOB as market orders do.

The final concept to be covered in this section is the *relative price*. Since the prices and depths in a LOB are constantly changing over time, it is very rare to consider any specific prices in the modelling or analysis of the LOB and the order flow. Therefore, it would be useful to determine some sort of relative measure of the price. From a modelling perspective, this also helps to normalise the price variable. Also, studies reported in the literature have concluded that the order flow depends on the relative price rather than the actual price [36, 37, 38]. The ask-relative price for a given price p is defined as $\delta^a(p) = a(t) - p$, and the bid-relative price is $\delta^b(p) = p - b(t)$. The work in this thesis applies the ask-relative price to buy orders, and the bid-relative price to sell orders. Also, the term *relative price* will be used from now on, omitting the prefix. To determine if it is bid-relative or ask-relative, it is necessary to consider the order side (buy or sell).

2.1.2 Matching Engine

The matching engine is the mechanism behind the LOB which maintains and updates the state of the LOB as orders are submitted and cancelled, as well as registering all transactions as orders are matched and filled. The most common matching algorithm that is implemented by LOB in practice is the *price-time*

priority algorithm.

When some marketable buy (or, sell) order y that is submitted immediately after time t, the orders that are resting at the best ask $a(t)$ (or, best bid $b(t)$) will be given the highest priority to be matched against y. If there are multiple orders queued at the best bid or best ask, the First In, First Out (FIFO) rule is applied where the oldest order x by timestamp t_x has the highest priority to be filled. If $\omega_x <= \omega_y$, the orders are matched and the transaction is completed and registered. Otherwise if $\omega_x > \omega_y$, the next oldest order will be given the priority to fill the remaining portion $\omega_y - \omega_x$, repeating until the order y is filled. However, if $\omega_y > n^a(a(t),t)$ (or, $\omega_y > n^b(b(t),t)$), the queue at $a(t)$ (or, $b(t)$) will be depleted. As a consequence, the orders resting at the next lowest price $\min_{x\in\mathscr{A}(t_y)} p_x$ (or, highest price $\max_{x\in\mathscr{B}(t_y)} p_x$) will be given the highest priority beginning from the oldest resting order.

An alternative to price-time priority is the *pro-rata* matching algorithm, which gives priority to resting orders at the best prices $b(t)$ (or, $a(t)$), but fills the orders in the queue proportionally to the relative size of each resting order. The work in this thesis however only implements the price-time priority matching engine as it is the most common system in practice.

2.1.3 Limit Order Book Implementation

The LOB is the central component of the work in this thesis as it is needed to transform raw order flow messages into datasets, to compute the relative prices for the events in the order flow, and to compute other metrics such as prices and depth. Therefore, a limit order book needed to be implemented before any work in this thesis could begin.

The implemented limit order book needs to be able to perform three main operations: ”add”, ”cancel”, and ”execute”:

- An ”add” operation places a limit order x in the LOB, at the end of the queue corresponding to its price p_x, as per the FIFO rule.
- A ”cancel” operation removes the corresponding order from the LOB.

- An ”execute” operation removes the an order from the queue $b(t)$ or $a(t)$, as per the price-time priority rule.

The limit order book needs to be organised into an efficient data structure since the number of order messages can get very large. To illustrate this, the average inter-arrival time between the events of the order flow data used for the work in this thesis is 13 milliseconds. The implemented LOB needs to be able to perform these three main operations efficiently, while also enabling the user to query measures such as $b(t)$, $a(t)$, the depths and the relative prices.

Such efficiency can be achieved using a binary search tree of sorted Price objects, where each Price object corresponds to a price queue in the LOB. Each price object needs to be a doubly linked list of Order objects, where each Order object corresponds to a unique order identified by an ID number. Each side of the LOB needs a separate tree so that $b(t)$ and $a(t)$ would correspond to the start and end of the bid-side tree and the ask-side tree respectively. Each Order also needs to be an entry in a hash map keyed off its ID, so that it can be tracked for the three operations above. In addition, each Price object also needs to be in a hash map, keyed off its price. With this structure, theoretically $O(1)$ can be guaranteed for ”cancel”, ”execute”, computing $b(t)$ and $a(t)$, and computing the depth at a given price. For the ”add” operation, $O(\log n)$ is guaranteed for the first order at a given price, and $O(1)$ for others. It is recommended that a red-black tree be implemented for self-balancing as orders will be constantly removed from one side of the tree and added to the other.

2.1.4 Order Flow and Mid-Price Movements

The *order flow* is the sequence of limit orders, market orders and cancellation orders submitted by traders to the exchange. The evolution of the shape of the LOB, and ultimately the dynamical movements of prices, are determined by each order submitted by traders. Since the modelling of the order flow to predict movements of the mid-price is the main focus of this thesis, the series of events in which the mid-price changes as a result of an incoming order is examined here in more detail. This is for the benefit of readers that are less familiar with

the workings of the limit order book.

Consider the effect, on the mid-price of the traded asset, of a buy (or, sell) order x that is submitted to the LOB immediately after time t:

- If $p_x \leq b(t)$ (or, $p_x \geq a(t)$), then x is a limit order that will be placed in the corresponding price queue. It does not immediately or directly cause any mid-price movements.

- If $b(t) \geq p_x \geq a(t)$, then x is a limit order that creates a new price queue within the spread. As a result, $b(t_x) = p_x$ (or, $a(t_x) = p_x$) and hence the mid-price has moved.

- If $p_x \geq a(t)$ (or, $p_x \leq b(t)$), then x is a marketable order that can be immediately executed against the oldest order resting in the queue at $a(t)$ (or, $b(t)$). Depending on the size of the order ω_x, the mid-price may be affected. As per the price-time priority algorithm described in Section 2.1.2, an incoming order will affect the value of the mid-price $m(t_x)$ as follows. If x is an incoming buy order, then the best ask will be shifted as:

$$a(t_x) = max(p_x, q), \text{ where } q = \underset{p'}{\operatorname{argmin}} \sum_{p=a(t)}^{p'} n^a(p,t) > \omega_x,$$

  which may move the mid-price upwards. Meanwhile, an incoming sell order x may move the mid-price downwards due to the possible movements in the best bid:

$$b(t_x) = max(p_x, q), \text{ where } q = \underset{p'}{\operatorname{argmin}} \sum_{p=p'}^{b(t)} n^b(p,t) > \omega_x.$$

The order x could also be a cancellation order, to immediately remove an existing limit order resting in the LOB. Consider an order timestamped $t_{x,c}$ submitted immediately after time t to cancel a limit order x resting in the bid (or, ask) side of the book:

- If $p_x < b(t)$ (or, $p_x > a(t)$), then x is simply removed without immediately

or directly affecting the mid-price.

- If $p_x = b(t)$ (or, $p_x = a(t)$), then x is removed, and the new bid price due to the cancelled order is $b(t_{x,c}) = \max_{x \in \mathscr{B}(t_{x,c})} p_x$ (or, new ask price $a(t_{x,c}) = \min_{x \in \mathscr{A}(t_{x,c})} p_x$), and the mid-price will move correspondingly.

Although the placement and cancellation of limit orders at $p_x \neq a(t)$ or $p_x \neq b(t)$ do not immediately affect the mid-price, these events will change the levels of liquidity in the LOB at different prices, which can affect the arrival of future orders as traders react to the changes.

Therefore, prices are driven by the arrival of events in the order flow as limit orders add liquidity to the LOB, and market orders and cancellation orders consume the liquidity in the LOB. Although it can be computed exactly how the price of a traded asset will affected by a submitted order, it is unknown what orders will arrive in the future. Models of price formation in the literature are aimed towards predicting and understanding this arrival of orders, and the subsequent change in the LOB and the price of the traded asset.

2.2 Deep Learning

Machine learning approaches can be broadly categorised into three groups: supervised learning, unsupervised learning and reinforcement learning. This thesis is only concerned with supervised learning approaches, and therefore the remainder of this section considers only supervised learning. In supervised learning, a model [39] estimates some function $f : X \to Y$ mapping an input space X to an output space Y. More specifically, given an input x and an output y, a model defines a mapping $y = f(x;\theta)$ and aims to find the parameters θ that result in the best functional approximation of the output. Deep learning approaches for supervised learning problems are no exception to this almost-universal model definition.

The core theme of deep learning is multi-layer artificial neural networks composed by chaining a number of such models. For example, the functions f_1, f_2, and f_3 can be connected into a chain structure of the form

$f(x) = f_1(f_2(f_3(x)))$. The functions f_1, f_2, and f_3 are called the first, second and third layers respectively, and form a 3-layer multilayer neural network when chained. The name *deep learning* comes from using this approach to modelling to stack multiple layers to form a *deep neural network*. The parameters of these layers are trained simultaneously using the same objective. In theory, as more layers are stacked, each layer learns a more and more abstract representation of the input data [40, 41], such that the generalisation ability of the whole network for learning the given data is improved.

There exist many different types of layers, each implementing different functional forms, and used for different purposes. The more commonly used layers are the feedforward layer [42], the convolutional layer [43], the max-pooling layer [43] and the recurrent layer [44], although other more exotic layers exist to fulfil niche purposes. Different kinds of layers can be combined depending on the modelling problem at hand. For example, it is common to stack a feedforward as an output layer after multiple recurrent layers, as a way to transform the output of the recurrent layers into classification probabilities. The majority of the work in this thesis employs feedforward layer and recurrent layers, which will now be presented as follows.

Common notations used in this section, and also in later chapters, are as follows. For variables, bold is used to indicate a non-scalar, while bold-capital indicates matrix; for example $\mathbf{x}$ is a vector, $\mathbf{X}$ is a matrix and x is a scalar. A collection of variables will often be shorthand using curly-brackets and a subscript, for instance $\{x_i\}_{1:N}$ indicates $\{x_1, x_2, \ldots, x_N\}$. This could be a set or a sequence, depending on the given context.

2.2.1 Feedforward Layer

The building block of a neural network layer is a *neuron* which also follows the functional input-output mapping paradigm described above. Given an input vector $\mathbf{x} \in \mathbb{R}^k$, the model of a neuron can be described as follows:

$$f(\mathbf{x}) = a(\mathbf{w} \cdot \mathbf{x} + b), \tag{2.1}$$

where $\mathbf{w} \in \mathbb{R}^k$ and b are parameters to be estimated, $a(.)$ is some non-linear activation function [45, 46] and the output $f(\mathbf{x}) \in \mathbb{R}$ is scalar. The neuron model in Equation 2.1 may slightly vary in different models, but with the same form, where there is a non-linear activation function acting on a linear combination. The neuron model in Equation 2.1 forms a *logistic regression model* [47] when a sigmoid function is chosen as the activation function. A neuron is often referred to as a *unit* or *hidden unit* in the literature and these terms will be used interchangeably in this thesis, unless referring to an output layer.

Multiple neurons can be assembled to form a *feedforward layer*. Given the input vector $\mathbf{x} \in \mathbb{R}^j$, a feedforward layer is the collection of neurons as follows:

$$f(\mathbf{x}) = a(\mathbf{w} \cdot \mathbf{x} + \mathbf{b}), \tag{2.2}$$

where $a(.)$ is some non-linear activation function, and the parameters to be fitted, $\mathbf{w} \in \mathbb{R}^{k \times j}$ and $b \in \mathbb{R}^j$, now have larger dimensions. This kind of layer is termed *feedforward* as all information flows in one direction and there are no feedback loops. Feedforward layers can be used on their own by stacking multiple feedforward layers to implement a deep feedforward network. Feedforward layers can also be used with other layers as an output layer or for embedding non-ordinal inputs.

The most common role of a feedforward layer is as an output layer. Let the penultimate layer of some deep neural network be denoted as $f_{L-1} \in \mathbb{R}^L$. Note that f_{L-1} does not have to be a feedforward layer. Then, a feedforward layer can be used to implement a final output layer with q hidden units with output $\mathbf{o} \in \mathbb{R}^q$ defined as follows:

$$f_L(f_{L-1}) = a(\mathbf{w} \cdot f_{L-1} + \mathbf{b}), \tag{2.3}$$

where $a(.)$ is some non-linear activation function, and $\mathbf{w} \in \mathbb{R}^{q \times L}$ and $\mathbf{b} \in \mathbb{R}^q$ are parameters to be fitted. From Equation 2.3, a number of things can happen depending on the application. In an M-label classification problem for example,

q is set to M the number of labels, and a normalised exponential function applied to the output $\mathbf{o}$ to compute the probability of each class. For regression problems, q can be set to the number of exogenous variables to be modelled. In some applications, stacking at least one more layer $f_{L+1}(f_L)$ could improve the performance of the model by learning a more abstract representation of the penultimate layer.

Feedforward layers are also commonly used as *embedding layers*. An embedding layer transforms a given non-ordinal categorical variable, which is always one-hot encoded, into some continuous space. The reason why this is needed is because in many datasets, categorical variable can have hundreds, if not thousands of possible values, resulting in an extremely large one-hot encoded vector. By using an embedding layer, the dimension of a one-hot encoded vector can be reduced to the number of neurons in the layer. In certain problems that have categorical variables representing small sets, a larger dimension can be set to improve learning by mimicking the effect of kernel tricks in support vector machines for learning non-linear data [48]. Since embedding layers are part of a deep neural network and are trained simultaneously with other layers, an embedding layer also acts as a mechanism for the representation learning of abstract features of the categorical variable [49]. The feature space of these abstract features can also be analysed using clustering algorithms to provide some unsupervised learning insight into the categorical variable. An embedding layer can be implemented as follows. Given a categorical variable encoded into a one-hot vector x, the embedding layer simply follows Equation 2.2, with $f(\mathbf{x})$ mapping some abstract feature representation of the one-hot vector x. Though embedding layers are powerful and contributes much to the performance a deep neural network, they comes with a computational cost as the number of hidden units in each embedding layer is a hyperparameter that needs to be tuned.

2.2.2 Recurrent Layer

Unlike feedforward layers, a recurrent layer has feedback connections, which makes it specialised for processing sequence data of the form $\{\mathbf{x}_t\}_{1:T}$. Given an

input vector $\mathbf{x}_t$ at timestep t, a recurrent layer produces an output, which is then fed through a feedback connection back into the layer for processing $\mathbf{x}_{t+1}$. The output of a recurrent layer is therefore a $\mathbf{h}_t \in \mathbb{R}^k$ (called a *hidden state*), which is dependent on $\mathbf{x}_t \in \mathbb{R}^j$ and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^k$. This can be expressed as the following:

$$\mathbf{h}_t = a(\mathbf{h}_{t-1}, \mathbf{x}_t), \tag{2.4}$$

where $a(.)$ is some non-linear activation function. Just as with feedforward layers, the recurrent layer in Equation 2.4 can be stacked to form a deep recurrent neural network that learns more abstract representation of the data in each layer. For example, denoting the output first layer as $\mathbf{h}^{(1)}$, the second layer can be stacked and so on, as shown in the following:

$$\mathbf{h}_t^{(2)} = a(\mathbf{h}_{t-1}^{(2)}, \mathbf{h}_t^{(1)}), \tag{2.5}$$

A vanilla recurrent layer can be implemented simply using the linear combination and non-linear activation function framework, similar to the architecture in Equation 2.3:

$$\mathbf{h}_t = a(\mathbf{W} \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}), \tag{2.6}$$

where $\mathbf{W} \in \mathbb{R}^{k \times (k+j)}$ and $\mathbf{b} \in \mathbb{R}^k$ are parameters to be estimated. However, in practice this form of recurrent cell suffers from the vanishing gradient problem [50] during optimisation. The vanishing gradient problem occurs when the layer is not able to learn any long-term dependency in the data between two timesteps, as the gap between the timesteps gets larger. The work in [51] presents a more in-depth study of this problem, and gives reasons why the vanishing gradient problem persists in practice.

In the light of vital problem, the long short-term memory (LSTM) architecture was proposed, to better handle long sequences and deal with the vanishing gradient problem. The core idea of the LSTM architecture is a memory cell $\mathbf{C}_t$,

which depends on the cell state at the previous time step $\mathbf{C}_{t-1}$. Information can be removed or added to this cell state, but this is regulated by structures called *gates*. Gates protect and control the cell-state, partially letting information through by having an output between zero and one, determining which information in each component of the cell state is to be let through. The architecture of LSTMs can be expressed as the following, given an input sequence $\{\mathbf{x}_t\}_{1:T}$:

$$\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b})_i), \\
\tilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C), \\
\mathbf{C}_t &= \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t, \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(\mathbf{C}_t),
\end{aligned} \tag{2.7}$$

where $\sigma(.)$ is the sigmoid function, $\mathbf{W}_{(.)}$, $\mathbf{b}_{(.)}$ are model parameters to be fitted, and the symbol $*$ denotes the Hadamard product.

In Equation 2.7, $\mathbf{f}_t$ is the *forget gate* which decides the information in the previous cell state to let through, $\mathbf{i}_t$ is the *input gate* that decides which values in the cell state is to be updated, and $\tilde{\mathbf{C}}_t$ contains the candidate update values. The resulting new cell state is $\mathbf{C}_t$, which will be used in the next timestep. An interesting note is that, compared to the standard RNN in Equation 2.6, a simple activation function is here used to incorporate the relationship between the input $\mathbf{x}$, the previous hidden state $\mathbf{h}_{t-1}$ and the current hidden state $\mathbf{h}_t$, while the LSTM architecture structures this relationship using the cell state $\mathbf{C}_t$.

Other variants of the LSTM architecture in Equation 2.7 exist, such as the LSTM with peephole connections [52], or the Gated Recurrent Unit [53]. However, an empirical study of different architecture variants shows that the differences in modelling performance between the variants, and the basic LSTM above, are negligible [54]. Also, in the preliminary work for this thesis, both

GRUs and LTSMs were implemented with no performance differences between them. Therefore, the work in this thesis exclusively implements the LSTM architecture.

2.2.3 Optimisation

To optimise the parameters of a deep learning model, typically a loss function needs to be defined to measure how well the parameters of the model predict the examples in the dataset. This section will only cover the cross-entropy loss function for binary classification, which is used in multiple chapters in this thesis. Other loss functions are also used in this thesis, but these are only relevant to specific chapters, and will be covered in those chapters instead.

For a single data sample with true label $y \in \{0,1\}$, and an estimated probability $\hat{y} = Pr(y = 1)$, the cross-entropy loss is computed as follows:

$$F(y,\hat{y}) = -y\log\hat{y} - (1-y)log(1-\hat{y}), \tag{2.8}$$

where $\hat{y}$ is estimated by the output of a model, and is thus a function of the network parameters θ. In practice, the total loss is taken by summing across the dataset $\sum_{i=1}^{N} F(y_i,\hat{y_i})$.

The learning task in a deep learning model is an optimisation problem. First, the model output and the loss function are computed using some initial parameters. Then, the parameters of the model are updated in each layer, within a single step of the optimisation procedure, to decrease the value of the loss. This process is repeated until the proper model is obtained. The most common way to optimise the parameters of a deep learning model is by stochastic gradient descent and its variants. Gradient descent algorithms minimise the loss function by iteratively updating θ in the opposite direction of the loss function gradient w.r.t to θ, following the direction of the loss function surface slope until some minimum is reached. Stochastic gradient descent algorithms update the parameters by a gap corresponding to the Jacobian matrix, which scales very well when the size of the training set becomes large. Readers are directed to

the excellent tutorial on gradient descent algorithms by [55] for further details. For the work in this thesis, the stochastic gradient descent variants Adam [56] and ADAGRAD [57] are deployed for training the implemented models.

2.2.4 Performance Metrics

To evaluate the performance of binary classifiers, the Matthews correlation coefficient (MCC) [58] is used across multiple chapters in this thesis. Other metrics are also employed but are only relevant to specific chapters, and will be covered in those chapters instead. The MCC is a measure of the quality of binary classifications, calculated from the true and false positives and negatives in a confusion matrix. By summarising the contents of a confusion matrix in a single balanced and intuitive measure, it allows for concise and extensive performance comparisons. The following equation computes the MCC:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, \tag{2.9}$$

where TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives. The MCC is essentially a correlation coefficient between the predicted and true binary classifications, with values in the range of $(-1, 1)$. An MCC of $+1$ indicates a perfect classifier, -1 indicates a completely wrong classifier, while 0 indicates either that the classification model is doing no better than making random predictions, or is categorising all examples as being of the same label.

2.3 Related Work

Existing studies relating to the modelling of the price formation process can be broadly categorised into *theory-driven* and *data-driven*. Theory-driven models assume some theoretical view of the formation of prices, whether this is based on economic models of rational trading behaviour, or assumes that some stochastic process is the driving force behind price formation. These theories are usually supported by evidence from empirical research on financial markets.

The naming used here by no means implied that ”theory-driven” models do not consume any data at all; these models are fitted on, and verified by, data. Data-driven models, on the other hand, implement some kind of learning approach to obtain a functional mapping or probabilistic model of the price movements, based on the given data. These models may be parametric statistical models of non-parametric machine learning approaches. This section reviews existing studies of both theory-driven and data-driven modelling of the price formation, beginning with a discussion of theory-driven models.

2.3.1 Theory-Driven Models

Theory driven-models of price formation are characterised by two extremes. On one end are *perfect-rationality* approaches, while on the other are *zero-intelligence* approaches [35]. Economists traditionally approach the problem of LOB modelling using the perfect-rationality framework [59]. Perfect-rationality approaches are trader-centric, addressing the strategic behaviour of rational investors when submitting limit orders and market orders. However, the perfect-rationality framework for modelling the dynamics of the LOB, and consequently price movements, has significant drawbacks due to a consolidation of unobservable parameters using a series of auxiliary assumptions including the rationality of investors, inclusion of a designated market maker, and disallowing cancellations of active orders. Such assumptions lead to inconsistency between the models and observations of trader behaviour [60, 61, 62].

On the other hand, zero-intelligence (ZI) approaches, favoured by the new generation of physicist-quants, do not assume assume any rational trading behaviour is driving the order flow. As implied by their name, zero-intelligence models instead assume that the order flow is governed by stochastic processes. The parameters of the processes are conditional on variables such as the state of $\mathscr{L}(t)$, or past sequences of the order flow, therefore mapping the order flow to the evolution of the LOB and the dynamics of price movements. The industry and the literature have mostly converged to a preference of ZI approaches over perfect-rationality approaches in modelling price formation. Although many ZI

models also assume simplified models of the market, they are even so able to produce quantifiable and falsifiable predictions. This is because the parameters of the stochastic processes can be estimated from historical data, and the outputs are directly comparable to real data. Zero-intelligence approaches additionally treat order flow as dynamic rather than static, as in perfect-rationality. Further discussion in this section will be focused on zero-intelligence models, as this type of modelling approach is closer to work on the deep modelling of price formation in this thesis.

The framework used in most zero-intelligence (ZI) models was inspired by the work in [63], which studied the interplay between perfect-rationality traders and ZI traders. In a market with only ZI traders, price movements were mapped onto a model of diffusing and annihilating particles. Orders submitted by the ZI traders were modelled as a reaction-diffusion system [64], where an order was a particle on a one-dimensional price lattice. Sell orders were represented as type A particles, while buy orders were represented as type B particles. The model begins with all A particles to the right of all B particles, essentially the bid and ask sides of the LOB. Each particle then moves along the lattice by a random walk, which results in variation of prices over time. Although this diffusion model is able to reproduce some empirical regularities reported in real LOB data [63], it has been largely rejected due to the unrealistic premise of diffusion of resting limit orders across different prices [4]. The study however demonstrated that such a zero-intelligence framework, with all the computational and practical advantages that come with it, is able to realistically model the LOB, and consequently the price movements, without the need for modelling rational trading behaviour. By implication, the study also suggests that data-driven models using statistical learning or machine learning will be able to realistically predict price movements without needing to explicitly learn rational trader behaviour.

Using this ZI framework, subsequent work has mostly assumed stochastic dynamics of $\mathscr{L}(t)$, and that consequently prices are driven by the random arrivals of order submissions and cancellations, so that the price movements in a

LOB can be modelled by one or more stochastic process governing the arrival of order events. Early work was focused on the arrival of traders in discrete time, where, on arrival, the trader submits a limit order or market order [65]. Extensions allowed for evaporation orders, which essentially model the cancellation of a limit orders [66]. Though these discrete models in these early works were able to reproduce some empirical regularities such as heavy tails in the price returns, continuous time models, which model irregularly spaced sequential data, are necessary for high-resolution characterisation of the LOB and price dynamics [67].

Continuous-time ZI models assume that the irregularly-spaced market, limit and cancellation orders are each governed by independent counting processes. Much of the work on continuous-time ZI models implements the multiple Poisson process framework, first introduced in [2, 3], and refined in [68]. The Poisson process has seen much application in this area due to its tractability and the ease in obtaining an analytical closed-form solution to the state of the $\mathscr{L}(t)$, $a(t)$, $b(t)$, $m(t)$, and the depth-profiles. The utility of this framework is illustrated in [69], where the authors applied a modified variant of multiple Poisson processes to model the order flow, with the aim of investigating the possible causes of the infamous 2010 Flash Crash [70].

Poisson processes are just one of many kinds of point processes that have the potential for modelling the arrival rates of orders. Another notable process that is often seen in the literature is the Hawkes point process [71]. The Hawkes process is an excitable point process with a time-varying intensity kernel that governs the surge and decay in arrival intensities. The authors in [5] showed that by modelling market order arrivals with a process that can be excited by recent orders, the statistical properties of order flow sequences are improved when compared to actual data. Similarly, the authors in [6] revised the approach in [68] and observed improvements in the empirical regularities of the output compared to real data. Also, the work in [9] proposed a 4-variate Hawkes process to model randomly timed price movements and the arrival of market orders.

However, the parametric estimation of the Hawkes process requires a-priori knowledge of the kernel shape, while the non-parametric estimation problem requires a numerical solution that suffers from stability issues. The simplicity and tractability of independent Poisson processes leads to straightforward implementation and allows for in-depth closed-form analysis. For this reason, there is a wider use of multiple Poisson ZI models both in the industry and in the literature, and thus it will be of much use later in this thesis as a benchmark model. The multiple Poisson models in [2, 3, 69, 68] can be generalised into a single form, and the next subsection will describe this generalised form of the multiple Poisson model.

2.3.2 Multiple Poisson Models

As discussed in previous sections, the stochastic dynamics of the LOB is driven by the submission to, cancellation of, and execution of, limit orders in the LOB over time. Thus, the LOB, and consequently the price movements, can be simulated by multiple stochastic processes modelling the random arrival of order book events. The random arrival of a given type of order can be modelled by an independent counting process such as the Poisson process. The method in which Poisson processes model the order flow can take slightly different forms, referring to the work in [69, 68, 3, 2]. However, a generalised formulation consisting of multiple independent Poisson processes, can be derived as outlined below.

Each Poisson process essentially models a type of order event such as the limit order, market order or cancellation order as follows:

- Bid limit orders to buy at relative price q from the best ask arrive independently in exponential time at the rate of $\lambda_B(q)$.
- Ask limit orders to sell at relative price q from the best bid arrive independently in exponential time at the rate of $\lambda_A(q)$.
- Buy market orders arrive independently at an exponential rate of μ_B.

- Sell market orders arrive independently at an exponential rate of μ_A.
- Cancellation of bid limit orders resting at relative price q from the best ask are cancelled at the exponential rate of $\theta_B(q)$.
- Cancellation of ask limit orders resting at relative price q from the best bid are cancelled at the exponential rate of $\theta_A(q)$.

In the above formulation, there are Q Poisson processes generating bid limit orders, and Q processes generating ask limit orders, at relative prices of up to Q ticks away from the opposite best price. Relative prices further than Q ticks away from the best bid and best ask are not considered since market activity tends to get less the further one gets from the best bid and best ask [72, 73]. Then, a further two processes are used for generating buy and sell market orders respectively. Finally, another $2Q$ processes are used to model the arrival of cancellation orders for the limit orders resting at each relative price q in the limit order book.

The models in [69, 68, 3, 2] make the simplification assumption that the order sizes are equal, where $\omega_x = \omega$ unit for all orders $x \in \mathscr{L}(t)$. With this assumption and with the independent arrival of orders dictated by the multiple Poisson processes, the previously introduced LOB price depths $n^b(p,t)$ and $n^a(p,t)$ can be described as continuous-time Markov processes [68]:

- $n^b(q,t+\Delta t) \to n^b(q,t) + \omega$, with rate $\lambda_B(q)$, for all $1 \leq i \leq Q$
- $n^a(q,t+\Delta t) \to n^a(q,t) + \omega$, with rate $\lambda_A(q)$, for all $1 \leq i \leq Q$
- $n^b(k,t+\Delta t) \to n^b(k,t) - \omega$, with rate μ_A where $k = \frac{m(t)}{\pi}$
- $n^a(k,t+\Delta t) \to n^a(k,t) - \omega$, with rate μ_B where $k = \frac{m(t)}{\pi}$
- $n^b(q,t+\Delta t) \to n^b(q,t) - \omega$, with rate $\theta_B(q)$, for all $q \geq 1$
- $n^b(q,t+\Delta t) \to n^b(q,t) - \omega$, with rate $\theta_A(q)$, for all $q \geq 1$

Enough pieces of the puzzle have now been assembled to show that the ZI models in [69, 68, 3, 2] essentially describe a Poisson birth-death process. The depth $n^b(b(t),t)$ at the best bid is a birth-death process where the quantity available for trade at $b(t)$ enters a "plus one unit of order" state with probability (with the converse true for $n^a(a(t),t)$):

$$\frac{\lambda_B(q)}{\lambda_B(q)+\mu_A+\theta_B(q)}, \tag{2.10}$$

and a "minus one unit of order" state with probability

$$\frac{\mu_A+\theta_B(q)}{\lambda_B(q)+\mu_A+\theta_B(q)}. \tag{2.11}$$

At other bid-side price levels, the quantity available for trade at these price level enters the "plus one" state with probability (with the converse true for the ask-side price levels):

$$\frac{\lambda_B(q)}{\lambda_B(q)+\theta_B(q)}, \tag{2.12}$$

and the "minus one" state with probability:

$$\frac{\theta_B(q)}{\lambda_B(q)+\theta_B(q)}. \tag{2.13}$$

To conclude this section, the procedure for fitting the rate parameter of each of the Poisson processes in the formulation above is described. The average size of limit orders is denoted as S_l, the average size of market orders as S_m, and the average size of cancellation orders as S_c. These averages can be estimated directly from the data. Then, the arrival rate λq of a limit order can be estimated by a power-low function [38, 37] of the form:

$$\lambda'(q) = \frac{k}{q^\alpha}, \tag{2.14}$$

The parameters k and α can be obtained from the data by a least square fit [68]:

$$\min_{k,\alpha} \sum_{q=1}^{Q} \left(\frac{N_l(q)}{H} - \frac{k}{i^{\alpha}} \right)^2, \tag{2.15}$$

where $N_l(q)$ is the number of limit orders arriving in time H with price q. In the above, the subscripts A and B are omitted from the rate λ but the equations apply for both bid and ask limit order processes. Next, the market order arrival rate μ can be estimated by [68]:

$$\hat{\mu} = \frac{N_m}{H} \frac{S_m}{S_l}, \tag{2.16}$$

where N_m is the number of market orders arriving within time H. Finally, the cancellation rate $\theta(q)$ can be estimated by [68]:

$$\hat{\theta}(j) = \frac{N_c(q)}{H \cdot V_q} \frac{S_c}{S_l}, \tag{2.17}$$

where $N_c(q)$ is the number of cancellation orders at price q within time H, and Q_q is the average number of orders at relative price q.

2.3.3 Data-Driven Models

Philosophically, there exists two cultures of statistical modelling when deriving conclusions from data [74]. One assumes a data generating process, while the latter uses algorithmic models that treat the data mechanism as unknown. The former lends to the theory-driven models that have been examined in previous sections, while the data-driven machine learning models falls into the class of algorithmic models. Machine learning models are designed to provide predictions in complex application domains where relations between input and output variables are nonlinear and input space is often high dimensional. Therefore, machine learning models could improve over existing theory-driven models due to their potential to better generalise over a given dataset. In addition, the tractability issues experienced when fitting more advanced ZI models are not a problem due to how machine learning models are estimated. Interpretation issues, which are one of the key advantages of theory-driven models, are also fast being re-

solved by methods such as Shapley Additive Explanations [75].

However, the current state of the literature on the machine learning modelling of the high-frequency price formation using LOB data is disappointingly sparse. The authors in [76] propose a support vector machine approach for predicting price spread and price movement using features engineered from the limit order book. While, a GLM approach is implemented by the authors in [77] to predict directional price movements using engineered features from the limit order book, such as different variations of the order imbalance and the bid-ask spread.

Turning to the application of deep learning, most of the existing models of price formation implement the same standard RNN architecture with LSTM cells to predict directional mid-price movements. The main differences between these models are the features used as inputs rather than any significant advancements in the models themselves. The most common feature used is the multivariate time series of the depth profiles $n^b(p,t)$ and $n^a(p,t)$ of the LOB at different prices [28, 29]. The authors in [30] augment this time series with computed market order intensities, with the argument that such a feature could potentially predict price-flip events, since the arrival of market orders have a high probability of causing a price movement. An exception to the general use of RNNs is in [78], where the authors instead propose to use a bilinear network, and [31] which introduced a deep feature extraction network with a fully connected layer in the output. Both these models are similarly dependent on the time series of $n^b(p,t)$ and $n^a(p,t)$ as input features. It is also difficult to determine the true state of the art, as most of the models in existing work do not have a unified dataset, since the novelty in each work is the input features used to train the models. For that work [78] that did introduce a novel deep learning architecture to tackle the price formation problem a comparison was made between this model and those in [76, 31, 29]. In addition, despite the promising results reported in the work above, there were no attempts to apply black-box model interpretation methods to provide insights into the price formation process. The closest to such analysis

is in [31], in which the authors cluster their learned features to reveal insights into the structure of the financial market.

Though the pervasive use of the order flow in high-impact theory-driven models suggests that it is a very relevant predictor of price formation, at the time of the writing of this thesis, there exist no methods in the literature which explore the use of the order flow in data-driven models. In the next chapter, the first research objective in this chapter will be to explore the use of order flow data to predict directional price movements, and to investigate if the use of such data would prove to be an improvement over existing methods.

Chapter 3

Deep Modelling of Price Formation Using the Order Flow

This chapter presents a novel use of order flow data using a deep recurrent neural network to model the directional movements of the mid-price. The performance of this deep order flow model is benchmarked against previous work on forecasting high-frequency price movements that used deep learning models but did not make use of order flow data. After training the new and benchmark models on order flow data extracted from a Bitcoin exchange, evaluation of the models in the test period indicates that the deep order flow model outperforms the benchmark models. Further analysis of the results also shows that the deep order flow model exhibits strong stationarity: without any retraining, its performance was shown to be stable even as the currency shifts into an extremely volatile regime. The chapter is concluded by studying the aspects of the order flow that affect price formation, using a black-box explainer algorithm, and examining the results in contrast to previous work in the empirical study of market microstructure.

3.1 Introduction

Order flow data is essentially the most granular level *quantitative* data that can be collected containing market activity information that could be used for building prediction models. The fundamental premise of deep learning is the represen-

tation learning of these kinds of raw data, without any feature engineering, to achieve state-of-the-art results in some given machine learning task [79]. Data that has been subjected to manual feature extraction carries the risk of accidentally discarding important knowledge that can potentially be learned in a deep layer. In key application areas of deep learning such as natural language processing [80], visual recognition [81], and speech learning [82], there are no instances of preprocessing in the dataset that produces state-of-the-art results.

However, deep learning of features from non-preprocessed data has not made large incursions into the area of econometric and financial applications. The reasons for this are unclear. It may be in large part due to the difficulty of collecting large enough datasets, due to the cost involved. For instance, the large cost of obtaining order flow data from main stock exchanges is out of the reach of most investors. Other reasons may be that the underlying process of the application is not understood sufficiently well for raw data collection or because data in its preprocessed form is simply inaccessible, such as for instance, the browsing data of consumers in large e-commerce websites for prediction of e-commerce demand. Whichever the real reason for this lack of use of raw data may be, determining it is not the focus of this chapter, nor this thesis. As illustrated in Section 2.3.3, there is exists a clear void in the literature on the deep learning of order flow data in any modelling problems. The main objective in this chapter is to therefore fill this void, by investigating the potential of utilising order flow data for training deep learning models to model price formation process.

To achieve this objective, Section 3.3 introduces the deep learning framework that utilises order flow data to predict high-frequency mid-price movements. Subsequently, Section 3.2 details the benchmark model comprising of previous work selected the studies presented in Chapter 2.3.3 that applied deep learning for predicting high-frequency price movements. The source of the order flow data is described in Section 3.4, and the procedure for extracting the datasets from the raw order messages to train and test the models is outlined in Section 3.5.

A number of experiments were performed to investigate the predictive capabilities of the deep order flow model. The performance of the deep order flow model and the benchmark models are compared in Section 3.6.1. In brief, it is possible to obtain significantly strong performance in predicting high-frequency price movements using order flow trained models compared to the benchmarks. Section 3.6.2 then presents one of the major findings of this chapter. The deep order flow model exhibits remarkable stationarity property compared to the benchmark model, and possible even on its own. A short analysis of the universality property of the deep order flow model is then examined as well in Section 3.6.3, though with a caveat due to the limitations of the datasets. Finally, Section 3.7 concluded the work in this chapter by investigating the factors in the order flow that affect the price formation process. Though deep learning is a completely black-box approach to modelling, a black-box interpretation algorithm introduced to evaluate how each feature of the order flow data affects the performance of the deep order flow model. The results are then compared against existing empirical market microstructure research to provide an alternative data-driven insight into the possible factors for the price formation process.

3.2 Deep Order Flow Model

The order flow model is quite straightforward. An order flow $\mathbf{X}_i$ is defined to be a length T sequence of order events $\mathbf{x}_{i,t}$ as follows:

$$\mathbf{X}_i = \{\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,T}\}, \tag{3.1}$$

where i denotes a point in the dataset. The model then predicts the direction of the next price movement y_i:

$$p(y_i|\mathbf{X}_i), \tag{3.2}$$

Note here that the regular period indicator $1, \ldots, T$ associated with each order does not imply that the input data is regularly spaced. The model takes as input a sequence of irregularly spaced order events, as is natural for an order

flow. Since this is essentially a binary classification problem, the probability distribution of y_i can be described by the softmax function

$$P(y_i = j \mid \mathbf{h}_{i,T}^L, \mathbf{W}_j^D) = \frac{e^{z_j^D(\mathbf{h}_{i,T}^L, \mathbf{W}_j^D)}}{\sum_{k=0}^{K-1} e^{z_k^D(\mathbf{h}_{i,T}^L, \mathbf{W}_k^D)}}, \tag{3.3}$$

where $j \in \{0,1\}$ are $K = 2$ classes indicating the downward and upward price movements respectively, $\mathbf{h}_{i,T}^L$ is some learnt L-layer deep representation of a length T order flow, and z_k^D is the output layer of a D-layer fully-connected neural network.

The representation $\mathbf{h}_{i,T}^L$ is defined by

$$\mathbf{h}_{i,T}^l = \begin{cases} h(\mathbf{h}_{i,T}^{l-1}, \mathbf{h}_{i,T-1}^l, \Theta^l) & \text{if } 1 < l \leq L, \\ h(\mathbf{x}_{i,T}, \mathbf{h}_{i,T-1}^l, \Theta^l) & \text{if } l = 1, \end{cases} \tag{3.4}$$

where $h(.)$ is a function implementing an RNN with LSTM cells, Θ^l are LSTM parameters to be fitted, and $\mathbf{x}_{i,T}$ will be soon addressed. Section 2.3.3 can be referred to for more details on RNNs and LSTMs.

The parameters of the model can fitted using any stochastic gradient descent optimisation algorithm [55] by minimising the negative log-likelihood

$$\mathscr{L}(\mathbf{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=0}^{K-1} \mathbb{I}_{y_i = j} \; log \; p(j), \tag{3.5}$$

where $\mathbb{I}$ is the identity function that is equal to one if $y_i = j$ and is zero otherwise, K is the number of classes, and N is the size of the dataset. To fit the parameters of the model, the Adam optimisation, as described in Section 2.2.3 is used to minimise the above log-likelihood. Regularisation is achieved via a combination of dropout and early-stopping. To obtain the optimal hyperparameters, Bayesian hyperparameter tuning [83] is implemented.

Each order event $\mathbf{x}_{i,t}$ is essentially the vectorised form of a row of order messages and can be expressed as the tuple

$$\mathbf{x}_{i,t} = (x_{i,t}^{(1)}, x_{i,t}^{(2)}, x_{i,t}^{(3)}, x_{i,t}^{(4)}, x_{i,t}^{(5)}, x_{i,t}^{(6)}), \tag{3.6}$$

where:

- $x_{i,t}^{(1)} \in \mathbb{N}$ is the number of milliseconds between the arrival of $x_{i,t-1}$ and $x_{i,t}$, such that $\delta_{i,t} \geq 0$;
- $x_{i,t}^{(2)} \in \mathbb{N}$ is the hour of the arrival of $x_{i,t}$ according its timestamp, such that $h_{i,t} \geq 0$;
- $x_{i,t}^{(3)} \in \mathbb{R}$ is the size of the order $x_{i,t}$, such that $\omega_{i,t} > 0$;
- $x_{i,t}^{(4)} \in \{1,2,3\}$ is the categorical variable for $x_{i,t}$ being a limit order, market order or cancellation order;
- $x_{i,t}^{(5)} \in \{1,2\}$ is the categorical variable for whether $x_{i,t}$ is a buy order or sell order respectively;
- $x_{i,t}^{(6)} \in \mathbb{N}$, such that $\pi_{i,t} \geq 0$, is the relative price of the order $x_{i,t}$ as defined in Chapter 2.1.1;

For each order event, all the non-ordinal categorical features are embedded into a multidimensional continuous space using a fully-connected layer between the RNN and the feature input. Computation of such embeddings is described in Section 2.2.1.

To preserve the order flow in its original form as much as possible, each of the vectorised features above each represent a field in the order message, with the exception of the timestamp. The timestamp is broken down into features $x_{i,t}^{(1)}$ and $x_{i,t}^{(2)}$ due to the difficulty in representing a timestamp as a feature. Admittedly this weakens the ”raw data” claim of this chapter but the choice of timestamp representations is well supported in literature. The timing of order event arrivals has been shown to be most relevant in the modelling of price formation by the choice of stochastic process preferred in the literature[9, 6]. And also, there is strong evidence for intraday volatility patterns [8], which implies change in order flow activity at different times of the day.

3.3 Benchmark Models

The order flow formulation above is benchmarked against models chosen from amongst those previously described in Section 2.3.3. Some of the models presented in there were not chosen because they were shallow models, and are therefore not comparable to the deep learning theme in this chapter. The objective of the experiments in this chapter is to benchmark the result of training deep models using the order flow against previous work in the literature. To this end, it is desirable that a simple common architecture is duplicated across the benchmark models and the order flow model. Therefore, a novel architecture such as [31] and [78] will not be implemented. It can be argued that: if the experiments to follow show improvements when training on order flow instead of the LOB snapshot, the improvements should be applicable to these architectures as well.

The benchmark models hence follow the same modelling approach as described in 3.2 but with changes to the sequence $(\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots \mathbf{x}_{i,T})$. The sequence is now regularly spaced (i.e. a time series), and each element $\mathbf{x}_{i,t}$ no longer represents an order book event but now represents a temporal feature vector appropriate to the benchmark model considered. Descriptions of the models implemented will now follow.

Benchmark 1: Limit Order Book Snapshot

Most commonly in the literature, models are trained on a multivariate time-series of snapshots of the limit order book [76, 28, 29, 78, 31]. Each snapshot can be described as

$$\mathbf{x}_{i,t}^{(snap)} = (b_{i,t}^{(1)}, b_{i,t}^{(2)}, \ldots b_{i,t}^{(J)}, s_{i,t}^{(1)}, s_{i,t}^{(2)}, \ldots s_{i,t}^{(J)}), \tag{3.7}$$

where $b_{i,t}^{(j)} = (\pi_{i,t}^{(j)}, \omega_{i,t}^{(j)})$ is the tuple of price π and volume ω at the j'th highest buy price, $s_{i,t}^{(j)} = (\phi_{i,t}^{(j)}, \kappa_{i,t}^{(j)})$ is the tuple of price ϕ and volume κ at the j'th lowest sell price, J is the number of highest buy or lowest sell prices levels considered in the snapshot. In existing work using this approach, typically the number of

price levels is chosen as $J = 10$.

Benchmark 2: Limit Order Book Snapshot with Market Order Rates

The authors in [30] augmented the LOB snapshot with the intensity of market orders. Market orders are essentially aggressive orders depleting the volumes at the best ask and best bid. It was observed empirically in [30] that an increase in ratio between number of market orders and the volume resting at the top-of-the-book will more likely deplete the best bid/ask, which would contribute to a change in directional price movements later on. Each point in the multivariate time series is in this benchmark described as follows:

$$\mathbf{x}_{i,t}^{(agg)} = (b_{i,t}^{(1)}, b_{i,t}^{(2)}, \dots b_{i,t}^{(J)}, s_{i,t}^{(1)}, s_{i,t}^{(2)}, \dots s_{i,t}^{(J)}, \alpha_{i,t}^{(b)}, \alpha_{i,t}^{(s)}), \tag{3.8}$$

$$\alpha_{i,t}^{(b)} = \frac{m^{(b)}}{\omega_i^{(1)}, t}, \tag{3.9}$$

$$\alpha_{i,t}^{(s)} = \frac{m^{(s)}}{\kappa_i^{(1)}, t}, \tag{3.10}$$

In this, $b_{i,t}^{j}$ and $s_{i,t}^{j}$ are the same as described in 3.7 but without market order rates, and m^b (m^s) are the total number of buy (sell) market orders arriving within the span of the time series.

Volume Order Imbalance

The authors in [77] implemented a multivariate time-series of features engineered from the limit order book as follows:

$$\mathbf{x}_{i,t}^{(imba)} = (v_{i,t}, \rho_{i,t}, r_{i,t}), \tag{3.11}$$

$$v_{i,t} = \frac{\hat{v}_{i,t}^{(B)} - \hat{v}_{i,t}^{(S)}}{\sigma_{i,t}}, \tag{3.12}$$

$$\hat{v}_{i,t}^{(b)} = \begin{cases} 0, & \text{if } \pi_{i,t}^{(1)} > \pi_{i,t-1}^{(1)}, \\ \omega_{i,t}^{(1)} - \omega_{i,t-1}^{(1)}, & \text{if } \pi_{i,t}^{(1)} = \pi_{i,t-1}^{(1)}, \\ \omega_{i,t}^{(1)}, & \text{if } \pi_{i,t}^{(1)} < \pi_{i,t-1}^{(1)}, \end{cases} \tag{3.13}$$

$$\hat{v}_{i,t}^{(s)} = \begin{cases} \kappa_{i,t}^{(1)}, & \text{if } \phi_{i,t}^{(1)} > \phi_{i,t-1}^{(1)}, \\ \kappa_{i,t}^{(1)} - \kappa_{i,t-1}^{(1)}, & \text{if } \phi_{i,t}^{(1)} = \phi_{i,t-1}^{(1)}, \\ 0, & \text{if } \phi_{i,t}^{(1)} < \phi_{i,t-1}^{(1)}, \end{cases} \tag{3.14}$$

$$\rho_{i,t} = \frac{1}{\sigma_{i,t}} \frac{\omega_{i,t}^{(1)} - \kappa_{i,t}^{(1)}}{\omega_{i,t}^{(1)} + \kappa^{(1)}}, \tag{3.15}$$

$$r_{i,t} = \frac{1}{\sigma_{i,t}} \left(\hat{p}_{i,t} - \frac{\hat{y}_{i,t} - \hat{y}_{i,t-1}}{2} \right), \tag{3.16}$$

$$\hat{p}_{i,t} = \begin{cases} \hat{y}_{i,1}, & \text{if } t = 1, \\ \frac{1}{0.01} \frac{\chi_{i,t} - \chi_{t-1}}{V_{i,t} - V_{t-1}}, & \text{if } V_{i,t} \neq V_{i,t-1}, \\ \hat{p}_{i,t-1}, & \text{if } V_{i,t} = V_{i,t-1}, \end{cases} \tag{3.17}$$

In the above equations: $\hat{y}_{i,t}$ is the mid-price at time t; $\chi_{i,t}$ is the total value transacted aggregated at time t; $V_{i,t}$ is the total volume transacted aggregated at time t; $\pi_{i,t}$, $\omega_{i,t}$, $phi_{i,t}$, and $\kappa_{i,t}$ are the price and volume as defined for the LOB snapshot above; and $\sigma_{i,t}$ is the bid-ask spread which can be computed as the difference between the best bid and best ask $\phi_{i,t}^{(1)} - \pi_{i,t}^{(1)}$.

Benchmark 3: Mid-Price Time Series

Lastly, it is always useful to include a ”naive” model as a benchmark. Since the target variable is the directional price movements, the naive model is chosen here to be simply a univariate time-series of the mid-price (with notation as defined above):

$$x_{i,t}^{(mid)} = \hat{y}_{i,t} \tag{3.18}$$

3.4 Data Source

Order flow data from most stock exchanges are usually very expensive or difficult to obtain for the typical trader. However, with the rise of cryptocurrencies, cryptocurrency exchanges provide API libraries that would allow access to the same kind of order flow data from regular stock exchanges at virtually no cost. The data for our experiment is thus obtained from Coinbase, a digital currency exchange. Using the Coinbase API, a websocket feed can be accessed, which provides a real-time stream of market data updates for orders and trades. With this, a bot was implemented to log all the messages in the real-time stream, for currency pairs BTC-USD and BTC-EUR. The bot was operated between 4 Nov 2017 to 22 Dec 2017. The messages are in JSON format, where a snapshot with select attributes are shown as follows:


```
{
  "time":"2017-11-05T17:10:44.732000Z",
  "type":"open",
  "remaining_size":"2.51000000",
  "order_id":"acf8e954-0488-4db0-a0b0-f405812ba0ec",
  "price":"7533.81000000",
  "reason":null,
  "side":"sell"
},
{
  "time":"2017-11-05T17:10:44.739000Z",
  "type":"received",
  "remaining_size":null,
  "order_id":"3dc23e38-36da-4c0d-99db-21e361b1cb44",
  "price":"7543.64000000",
  "reason":null,
  "side":"sell"
},
{
  "time":"2017-11-05T17:10:44.739000Z",
  "type":"open",
  "remaining_size":"0.12000000",
  "order_id":"3dc23e38-36da-4c0d-99db-21e361b1cb44",
  "price":"7543.64000000",
  "reason":null,
  "side":"sell"
},
{
  "time":"2017-11-05T17:10:44.747000Z",
  "type":"done",
  "remaining_size":"0.88000000",
  "order_id":"7bf7355e-3e24-4ab0-aab4-4b1771f4a73b",
  "price":"7554.90000000",
  "reason":"canceled",
  "side":"sell"
},
{
  "time":"2017-11-05T17:10:44.750000Z",
  "type":"received",
  "remaining_size":null,
  "order_id":"c74781bb-79dd-417f-8517-64915f41bdf8",
  "price":"7533.80000000",
  "reason":null,
  "side":"sell"
},
{
  "time":"2017-11-05T17:10:44.750000Z",
  "type":"open",
  "remaining_size":"0.53100000",
  "order_id":"c74781bb-79dd-417f-8517-64915f41bdf8",
  "price":"7533.80000000",
  "reason":null,
  "side":"sell"
}
```


From above, it can be seen that the raw JSON messages that were logged from the real-time stream does not provide a coherent order flow since it is not clear which messages pertains to the limit order, market order or cancellation orders respectively. Instead, they specify the submission, cancellation and execu-

tion of orders. However, is typical in literature to model the arrival of limit orders, market orders and order cancellation instead of order submission, cancellation and execution. Another clear issue here is that the JSON messages above is not a machine learning dataset that is suitable for use with typical deep learning models. Therefore, a limit order book and matching engine as described in Section 2.1.2 and Section 2.1.3 is implemented to process these messages and output a sequence of vectorised order flow with clear labelling of limit order, market order or cancellation orders.

An important note to make here is that the level of volume traded on Coinbase is of billions of US dollar, which is quite small compared to the trillions of US dollar. However, Coinbase is the second largest cryptocurrency exchange globally and is therefore a very liquid market. Results extracted from this data source should therefore generalise to other liquid markets such as other foreign exchange and stock markets. Also, the results in this thesis are relative and anchored on baseline models. Therefore, the contributions and conclusions drawn based on this dataset is valid regardless of the data source.

Similar to the final note in Chapter 3.8, the smaller scale of the trade volume on Coinbase does not restrict the contributions of this chapter. The functional capacity of the novel probabilistic architecture introduced in this chapter is independent of the dataset source.

3.5 Dataset

To compare the performance of different training features, raw order flow data versus the above benchmarks, a reliable procedure is needed to ensure both the alignment of the feature set to the same target variable, and fairness in terms of amount of information contained in the feature set. As mentioned in Section 3.4, the raw JSON message obtained from Coinbase needs to be fed through an implemented limit order book and matching engine to obtain a coherent dataset suitable for machine learning. During this procedure, the datasets for the order flow model and benchmark models are constructed as follows:

- Each arriving order event is vectorised as described in Equation 3.6, and then appended into a fixed-size T FIFO container. At the same time, the benchmark features are vectorised as described in Equations 3.18, 3.11, 3.8, and 3.7. Each benchmark feature vector is appended into an individual fixed-size T FIFO container.

- The mid-price is tracked at the arrival of each order event. When a change in the mid-price is detected:

    1. The direction of the price change is recorded as the *target variable*.

    2. The container of vectorised order flow is recorded as the corresponding *order flow feature* for the price change.

    3. For each benchmark model, the respective container of feature vectors is transformed into a regularly spaced time series by binning each element of the vector at the given point in time.

In this way, the training rows for the order flow and benchmark models are aligned to the same target variable. Also, the amount of information contained in the features is equal; though the sequential length may be different, the physical period of time from which the sequences are obtained is the same.

The datasets are then split into training, validation and holdout sets between dates set out in Table 3.1. A warm-up period is needed to set an initial state for the limit order book before data collection begins. Though the holdout set may seem unnecessarily large, a long holdout set is a good alternative to running multiple iterations of the experiment using a sliding window, which would have added a lot of additional complexity to the overall implementation pipeline. Though, implementing a sliding window or expanding window would be an interesting future work to draw an even stronger conclusion, especially in combination with a long holdout set.

Set	From	To
Warm-Up	4 Nov 2017	5 Nov 2017
Training	6 Nov 2017	16 Nov 2017
Validation	17 Nov 2017	20 Nov 2017
Holdout	21 Nov 2017	29 Jan 2018

Table 3.1: The start and end dates of the warm-up, training, validation and holdout sets

3.6 Results

In this section, for readability, abbreviations are used for the names of the features: i) order *flow*, ii) order book *snap*shot, iii) order book snapshot with *agg*ressors (market order rates), iv) bid-ask volume *imba*lance, v) past sequential *mid*-price changes.

3.6.1 Comparison of Model Performance

Table 3.2 provides an overview of the results, comparing the overall performance of the order flow model against the benchmark models. The metric used here is the Matthews Correlation Coefficient (MCC), which was described in Section 2.2.4. It is easy to see here that, for each dataset, the performance of model trained on the order flow outperforms those of the benchmark models. It is also observed that the performances of all models (with the exception of *mid*) are very good, being able to predict much better than random (MCC=0). This could potentially be attributed to the predictive and generalisation power of deep learning models, though further experiments with non-deep models would be needed for verification.

Dataset	Model				
	flow	*snap*	*agg*	*imba*	*mid*
BTC-USD	**0.485**	0.240	0.287	0.150	0.024
BTC-GBP	**0.309**	0.120	0.169	0.065	0.019
BTC-EUR	**0.398**	0.156	0.226	0.135	0.023

Table 3.2: Overall average MCC measured for each model on BTC-USD, BTC-GBP and BTC-EUR datasets, rounded to three decimal places

Although Table 3.2 clearly shows the better performance of the order flow model, for a more rigorous analysis, one-tailed Welch t-tests are performed be-

tween the order flow model and each benchmark model. For each test, the null hypothesis is stated as $H0 : B <= A$ where B is the mean of the MCC measured on the order flow model across the holdout set, while A is the mean for a given benchmark model. If the null hypothesis is rejected, then the alternative hypothesis $H1 : B > A$ is supported. The results of the statistical tests are presented in Table 3.3. It can be observed that for each test, the null hypothesis is rejected at a very high confidence interval. This results imply that the order flow model performs better than the benchmark models with very high statistical significance.

Dataset	Model							
	snap	*agg*	*imba*	*mid*	*snap*	*agg*	*imba*	*mid*
	t-statistics				p-value			
BTC-USD	32.13	25.37	44.33	127.73	0.00	0.00	0.00	0.00
BTC-GBP	20.73	13.94	33.22	45.76	0.00	0.00	0.00	0.00
BTC-EUR	26.31	23.24	34.84	92.03	0.00	0.00	0.00	0.00

Table 3.3: Results of one-tailed Welch t-test on the null hypothesis that the mean of the performance of the order flow model is less than or equal to the mean of the benchmark models, rounded to two decimal places

3.6.2 Analysis of Stationarity

The stationarity of a model is its ability to maintain consistent prediction performance not just out-of-sample, but across a range of periods where the underlying process that generates the data undergoes drastic changes. The financial market is subject to chaotic shift in regimes, and, as a consequence, a model that is trained and tested in a particular period is not guaranteed to perform as well if some unobservable underlying process of the financial market causes a drastic shift in the statistical properties of the data. As many of these regime shifting processes are subtle and not easily detected, particularly when trading at high frequencies, stationary models are highly desirable, being more robust and reliable as they do not need to be re-trained with recent data to ensure their performance. Though one study [29] suggests that good stationarity may be largely due to the modelling power of deep recurrent models, the order flow is

benchmark against the aforementioned model to show that by using the order flow, model stationarity can be greatly improved.

Other than its practical value, stationary modelling of the price formation process is of interest to the literature as it is related to the *universality* property of the financial markets as a complex dynamical system [1]. The universality of stylised facts and other statistical properties is evident from empirical studies of the financial market [84]. The existence of a stationary price formation model implies the existence of a universal mapping from the model covariates to the price movements across different regimes - a kind of *temporal universality*, to put it in another way.

The range of dates in the holdout period overlaps with the Bitcoin bubble where the price of Bitcoin climbed rapidly to an all-time-high and subsequently burst [85]. From Figure 3.1 it can be seen that, compared to the training and validation period, the holdout period for BTC-USD corresponds to a shift in regime characterised by more volatile price changes and much higher trading activity. The intraday volatility measured for the minute-interval log returns is 2.482 for the training period and 2.776 for the test period. Similar behaviour is observed for BTC-GBP and BTC-EUR. Therefore, it would be of interest here to investigate the stationarity of the models by analysing the change in performance of the models throughout the holdout period.

To perform this analysis, the prediction results on the holdout set for each model are grouped by date and then the average MCC for each date is computed. The resulting sequential change in the average MCC throughout the holdout period is plotted in Figure 3.2.

Though visually from Figure 3.2, it can already be noted that the order flow model performs better over time than the benchmark models, a simple test is implemented for a more rigorous and scientific treatment of the model stationarity comparison. Note here that the purpose is not to determine if a given sequence of MCC throughout the holdout period comes from a stationarity process, making the Dickey-Fuller test irrelevant in this case. The purpose here

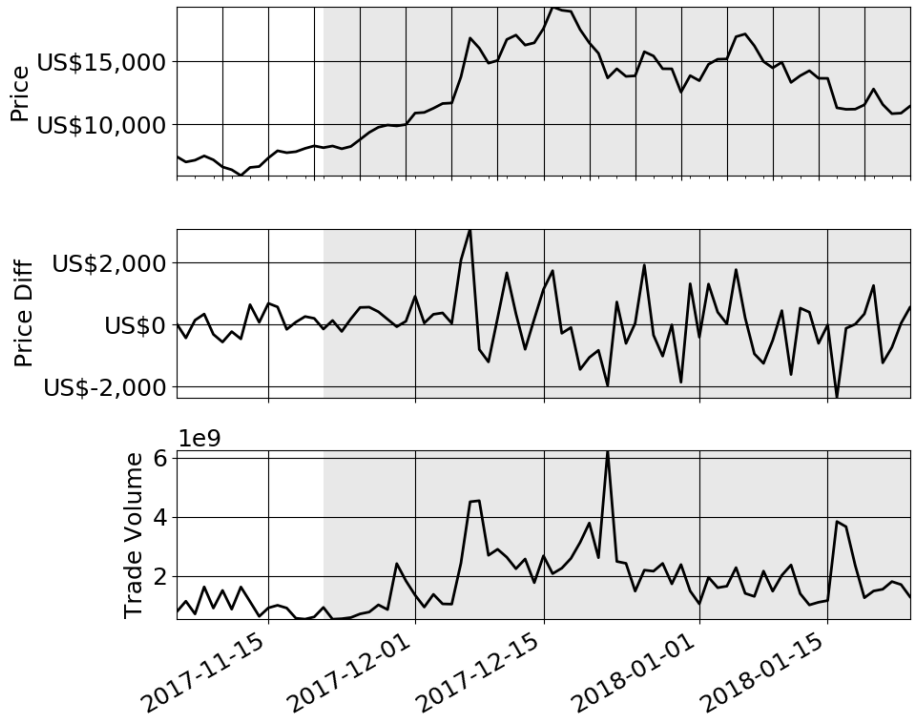


Figure 3.1: Plot of: i) Bitcoin against US Dollar daily prices (Price), ii) 1-day lagged difference prices (Price Diff), iii) volume of trading activity (Trade Volume), where the shaded area is the test period 21 Nov 2017 - 29 Jan 2018

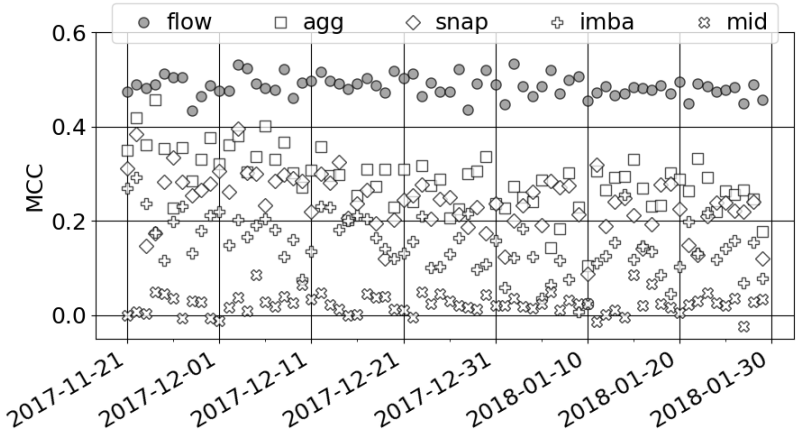
is to determine if and by how much each of the performance of the benchmark models degrade over time compared to the order flow model.

For this, a simple linear model is fitted for the sequential MCC result of each model as follow:

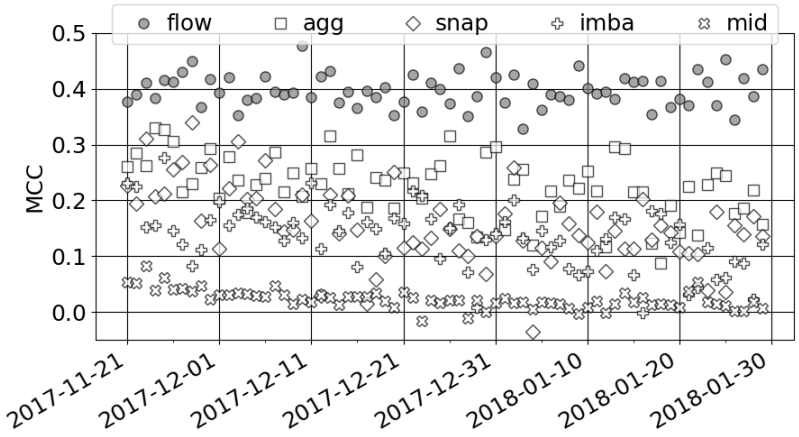
$$MCC^{(m)}(t) = \alpha^{(m)} + \beta^{(m)} t, \tag{3.19}$$

where $m \in \{flow, snap, agg, imba, mid\}$ indicates the model from which the sequence of MCC values is produced from. The slope coefficient $\beta^{(m)}$ can then be used to compare the performance change of the model over the holdout period.

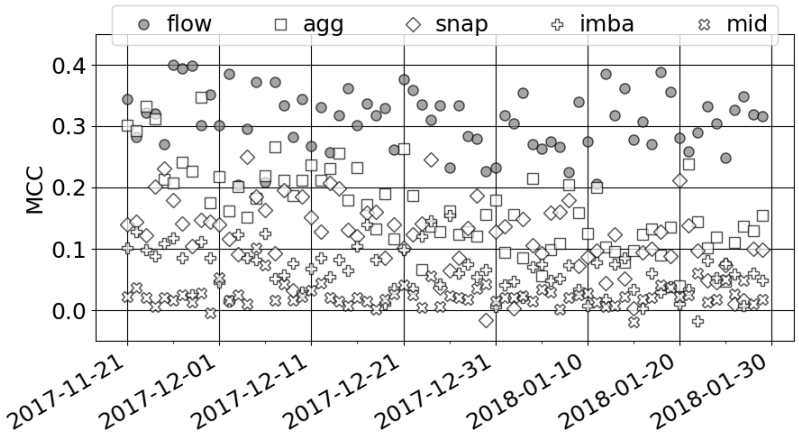
For the order flow model, $\beta^{(flow)}$ is shown in Table 3.4. It can be seen from the coefficients and p-values in Table 3.4 as if there is some decrease in performance over time. It is also noted that the slope of the order flow model trained on BTC-EUR is an order of magnitude smaller compared to the other



(a) Models trained and tested on BTC-USD



(b) Models trained and tested on BTC-EUR



(c) Models trained and tested on BTC-GBP

Figure 3.2: The plot of average MCC for each day in the holdout period for models that are: i) trained and tested on BTC-USD, ii) trained and tested on BTC-EUR, iii) trained and tested on BTC-GBP

two currency pairs. This can mainly be attributed to the sudden increase in comparatively large outliers towards the middle and end of the holdout period. However, from the observed p-values it can be concluded that the coefficients are *not statistically significant*. Since the p-values are quite large, the null hypothesis $H0 : \beta^{(flow)} = 0$ cannot be rejected. This implies very high confidence of there being little change in performance over time throughout the test period.

Dataset	$\beta^{(flow)}$	p-value
BTC-USD	-2.41e-4	0.06
BTC-EUR	-5.67e-5	0.75
BTC-GBP	-4.01e-4	0.17

Table 3.4: Slope coefficients and p-values of simple linear model fitted on the order flow model MCC performance over time for each holdout set, with p-values rounded to two decimal places

Next, the question is how do the resulting linear fits of the benchmark models compare to those of the order flow model in Table 3.4? Table 3.5 shows the slope coefficients and the corresponding p-values for the benchmark models. For the benchmark models *agg*, *snap*, *imba*, and *mid*, the null hypotheses $H0 : \beta^{(m)} = 0$ is confidently rejected in favor of the alternative $H0 : \beta^{(m)} != 0$. This means the negative coefficients in Table 3.5 can be used as strong evidence for performance degradation over time. Comparing this to the results in Table 3.4, it can be concluded that deep learning trained on the order flow exhibits much stronger stationarity in performance compared to the benchmark models.

3.6.3 Brief Investigation of Universality in the Order Flow Model

Although the main topic in this chapter is model stationary, it is in addition possible to show that the representations learnt from the order flow exhibit a hint of the very valuable property of universality, the ability to learn market structures which to some degree generalise across asset classes.

Table 3.6 shows the drop in performance on the out-of-sample test set, when training on one currency pair and testing on the others, is considerably

Dataset	m	$\beta^{(m)}$	p-value
BTC-USD	snap	-1.34e-3	0.00
	agg	-1.70e-3	0.00
	imba	-1.46e-3	0.00
	mid	1.76e-5	0.00
BTC-EUR	snap	-1.75e-3	0.00
	agg	-1.25e-3	0.00
	imba	-1.44e-3	0.00
	mid	-4.90e-4	0.00
BTC-GBP	snap	-1.35e-3	0.00
	agg	-2.36e-3	0.00
	imba	-8.57e-4	0.00
	mid	-3.59e-4	0.00

Table 3.5: Slope coefficients and p-values of MCC regressed on dates in the test period for the benchmark models, with p-values rounded to two decimal places

less when using the order flow model than the benchmark models, demonstrating the above-mentioned hint of universality. The innate ability to generalise across currency pairs is most evident when training on BTC-USD and least when training on BTC-GBP. This is likely due to the volumes traded: the trading volumes of BTC-USD, BTC-EUR, and BTC-GBP between the start of the training period (6 Nov 2017) and the end of the test period (29 Jan 2018) are, respectively, 151.6e9, 20.7e9, and 1.5e9 – when an asset is heavily traded there are more diverse activities at the order book level, resulting in a richer order flow and hence a richer dataset that helps to avoid over-fitting.

The reason why this analysis will not be more in-depth is due to the caveat that the only data available here are for three currency pairs that historically have very highly correlated price movements. The reason for this correlated price movement is that all three currency pairs are tied to Bitcoin by base currency, as well as the quoted-currency of all three pairs being the most highly traded currency in the world. Any opportunities for arbitrage therefore disappears quickly and hence one can observe highly correlated price movements among these three currencies. That said, however, at high-frequency there can still be some differences between the movements of the three currency pairs, and justifying the reason for at least a short treatment of this topic.

Model	BTC-EUR (%)	BTC-GBP (%)
flow	9.299	20.306
agg	59.857	71.749
snap	91.017	81.869
imba	87.504	94.253

(a) Models trained on BTC-USD

Model	BTC-USD(%)	BTC-GBP (%)
flow	19.606	17.387
agg	74.368	80.000
snap	99.619	84.049
imba	83.776	100.162

(b) Models trained on BTC-EUR

Model	BTC-USD(%)	BTC-EUR (%)
flow	28.816	32.232
agg	67.157	67.642
snap	80.432	70.483
imba	93.359	72.855

(c) Models trained on BTC-GBP

Table 3.6: For each model, tables show the mean percentage drop in test MCC between: i) training and testing on the dataset of a single currency pair, and ii) training on a given currency pair training set and testing on the holdout sets of the remaining currency pairs

3.7 Impact of Order Flow Features on Price Formation

Due to the encouraging performance of the order flow price formation model in Section 3.6 above, it would be of great interest for the study of market-microstructure to apply *model interpretation* algorithms to provide an insight into probable factors impacting the formation of prices. Such algorithms work to distinguish the order flow factors that contribute to the model making an upward or downward high-frequency price movement, providing a data-driven view into the price formation process.

Model interpretation is mostly simple in comparison, for shallow models. In [77] a simple linear regression model is used, which makes explaining the model straightforward since one can simply refer to the model coefficients. However, deep learning models are inherently black box models. The inputs to a deep

learning model typically pass through many layers of high-dimensional non-linearity, making it difficult to discern the effects of perturbation to the inputs on the model output. There are of course ways to approach this, especially since the area of *explainable deep learning* is fast become an important area of research [75]. Unfortunately, the state-of-the-art is still some way from explaining multiple layers of sequential models, making this approach unfeasible within this thesis.

Instead of studying how the model *output* varies with the input factors, it is much easier to relax the requirements here and study how the model *performance* changes with the input factors. To this end, the random feature permutation approach from the model interpretation of Random Forest [86] is implemented. This approach is highly suitable here for two reasons. First, unlike many specialised black-box explainers, this approach is model agnostic. Second, no model re-training is required, a quite considerable advantage as the models take a long time to cross-validate and train. The reader might suggest as an alternative to delete a feature, and re-run training and evaluation to determine feature importance. This is unsuitable however as it creates a different model each time a feature is deleted and the model is retrained.

The random feature permutation approach to model interpretation is straightforward. The following steps summarise the procedure for computing the effect of a given feature on the performance of a trained model:

1. Randomly shuffle the rows of the holdout set for only the given feature

2. Re-compute the output using the holdout set with the permutated feature

3. Compare the error of new output to the original output

Of course, for just a single feature, the change in error means nothing without a baseline for comparison. However, if the change in error for a number of features is computed, a comparative analysis can be performed to determine the features that are most important in modelling the price formation process.

The *feature importance* of the deep order flow model that is soon to be presented can be related to relevant empirical studies of high-frequency market behaviour, to open up a discussion of how this data-driven view of the price formation process adheres to, or differs from, previous findings in the literature.

In the feature permutation for the interpretation of the deep order flow model, each order event as defined in Equation 3.6 is modified to $\tilde{\mathbf{x}}_{i,t}^{(j)}$, where j indicates that the feature $x_{i,t}^{(j)}$ is permutated across the whole holdout set, for each given currency pair. In this way, any information from feature j is removed from the input, and also decoupled from the other features. Then, the unchanged trained model can be used to compute a new set of outputs that do not make use of any information from feature j. In this way, any change in the performance of the model can be attributed to the feature j.

For each feature, ten permutation-reforecast iterations are performed; in each iteration, the performance is compared against the none-permutated original forecast as presented in Table 3.2. Multiple iterations are performed to ensure that chance results from performing just one random permutation are avoided. Table 3.7 shows for each dataset the average change in performance for the permutation of each feature compared to the initial performance in Table 3.2.

j	Feature Permutated	Percentage Drop (%) in Average MCC		
		BTC-USD	BTC-EUR	BTC-GBP
1	Inter-arrival Time	1.78	7.83	9.13
2	Hour of Arrival	1.33	3.31	3.98
3	Size	7.57	3.92	4.16
4	Type	10.14	9.80	10.37
5	Direction	53.04	63.62	70.54
6	Price	36.14	49.01	51.90

Table 3.7: Average percentage drop in the mean MCC across the holdout set of each currency pair, after multiple rounds of permutating a given feature, compared to the performance in Table 3.2. The notation j is the index of the feature in the vectorised order event in Equation 3.6 for reference.

From the table, it can be seen that two features of an order book event have a large impact on the performance of the model. The feature that contributes

most to the order flow model performance is the direction of the order, which identifies an order event as either a buy or sell order. The second most impactful feature is the relative price associated with the order event. Of middling importance is the order type, which has considerable impact on model performance but not at the scale of order direction or order price. At the lower end of the feature importance are the inter-arrival time between each order, the size of the order, and the hour of arrival. For these features, how they rank among each other seems to be dependent on the holdout set. For BTC-USD, order size seems to be a lot more important than the inter-arrival time or the arrival hour, both of which have almost negligible effect on the model performance. For BTC-EUR and BTC-GBP, the order size is less important compared to the inter-arrival time. The drop in performance due to permutating the hour of arrival is not that negligible though still very small for BTC-EUR and BTC-GBP. Among the three datasets, the drop in performance from the permutation of any feature seems to be the worst in BTC-GBP but the best in BTC-USD. This once again points to the proposed interpretation that the BTC-USD training set is richer and hence trains the models better, while the BTC-GBP dataset is the worst dataset for this purpose among the three currency-pairs.

Since the role of the order flow in the price formation process is a well-published field of study, it is of interest to compare the findings in Table 3.7 to existing empirical studies in the literature.

3.7.1 Inter-Arrival Time Between Order

Let us first address the effect of feature $j = 1$ to the model performance, which is the inter-arrival time between each order event in the order flow. The negligible effect of the feature is quite surprising, given there is a wide use of temporal point processes in the state-of-the-art for theory-driven price formation models.

The authors in [2, 3] were the first to propose a ”master” equation of the limit order book consisting of multiple stationary point processes to model the arrival rate of each order book event type for both sides of the book. The ”master” equation can then be used to estimate the price formation process in either closed-

form or by Monte Carlo simulation. These equations are extended by [68], relaxing many assumptions previously made, thereby improving the ability of the model to better replicate the statistical behaviour of real-life markets. These temporal point process models can then be further extended by replacing the stationary Poisson processes with multivariate Hawkes processes [6, 10]. Multivariate Hawkes processes are both self-exciting and mutually exciting, which is backed up by market theories on how the arrival of an order book event tends to influence the future rate of arrival of the same event type or other events.

Admittedly here it may be like comparing apples with oranges since these temporal point processes and the deep order flow model approach the price formation process using very different frameworks. However, the importance in the literature of modelling the irregular arrival rates of order book events for computing the price diffusion would imply that the inter-arrival time between the order events, as an input feature, should contain significant information for predicting the directional price movement. However as seen in Table 3.7, the deep order flow model does not seem to have learn much from it.

3.7.2 Order Buy/Sell Direction

Moving on to feature $j = 5$, the buy/sell direction of the orders in the order flow seems to be the biggest contributor in predicting the next directional price movement. There are related empirical findings in the literature regarding the price impact of the direction of an order event. By analysing the kernels of a multivariate Hawkes model, the authors in [9] discovered that in their data buy orders mostly trigger and upward movement in the mid-price. Conversely, sell orders mostly trigger a downward movement in the mid-price. This relationship is later confirmed using higher resolution data and in more detail in [11]. The authors in both [9, 11] also found strong self-exciting behaviour of events of the same type and direction. Since predicting the next directional price movement is technically trying to predict the arrival of a market order that is large enough to move the ask or the bid, it may be that the deep order flow model is picking up signals from the market order of the same direction as the directional price movements.

Note here that a limit order placed inside the bid-ask spread may also move the mid-price but this is very rarely the cause of a mid-price movement since the currency pairs used as datasets are heavily traded and therefore almost always have a tick-size spread.

Though the order direction plays such a role in the deep order flow model, other than the papers above there exists few related studies in the literature. Using the feature permutation approach, unfortunately it is not possible to uncover the exact causal relationship between the direction of each event in the order flow, and the upward or downward mid-price movements themselves.

3.7.3 Order Price

Feature $j = 6$, the price associated with the order in the order flow event, is another feature that has so far had little study done on its price impact. A reminder here that this price is the relative price, computed as the number of ticks from the bid or ask price, depending on the direction of the order. The authors in [87] found that for their dataset, the relative price of a limit order arriving at the order book has a price impact with strength relative to the magnitude of the relative price of the limit order. Limit orders with relative prices one or two ticks away from the bid or ask prices affect the price about 20% less than limit orders placed at bid or ask.

3.7.4 Order Type

The order type feature $j = 4$ is shown in Table 3.7 to have a significant, though not the largest effect on the prediction of the next directional price movement. Referring once again to the work of the authors in [11], the kernel of the fitted multivariate Hawkes process reveals some relationship between the type of the order and the mid-price movements in their datasets. In the dataset for one stock exchange, buy limit orders are found to impact price movements in the upward direction, and conversely for sell limit orders. For another exchange, it was found that limit orders have instead an inhibiting effect on the price movements. Cancellation orders were found to impact the price in similar ways to the limit

orders but in the opposite direction. Market orders were found to be self-exciting in the same direction, which may lead to arrival of market orders that move the mid-price. Buy market orders were found to directly result in higher frequency of upward price movements and vice-versa.

3.7.5 Other Features

Moving finally to what were here determined to be the least significant features, there are some studies empirically showing the existence of intraday seasonality [88] that may contribute to the predictions, but as seen in Table 3.7, feature $j = 2$, the hour of arrival, seems to have very little effect on the model. However, the size associated with the order, feature $j = 3$, has been shown in previous work to impact the prices. In an older study, the authors in [89] found some relationship between the size of market orders and the eventual price movements in their datasets. Other later authors found power-law [90, 91] and logarithmic [92] relationships between the size of market orders and the price impact in their respective datasets. Supporting these authors finding in the relationships between order size and price impact, the model in this chapter was found to have some decrease in performance as the order size feature is permutated.

3.8 Summary

In this chapter, an approach the deep learning of the order flow for modelling high-frequency price movements was introduced. To evaluate the significance of the the deep order flow model performance, benchmark models from the literature were implemented and a methodology laid out for a fair and rigorous comparison process. In Section 3.6.1, the results showed that the deep order flow model outperforms the benchmark model with indisputable statistical significance. A deeper dive into the results in Section 3.6.2 showed that the deep order flow model is able to maintain performance throughout one of the most volatile periods in Bitcoin trading history, suggesting that the learnt representation encodes some sort of temporally universal information about Bitcoin price movements. In contrast, the benchmark models struggle to maintain perfor-

mance in the non-volatile period. Significance testing was performed to support these conclusions. Then, in Section 3.6.3, the deep order flow model was also shown to potentially exhibit the property of universality, though with some caveats.

As an additional investigation, Section 3.7 looked at the impact of each event feature in the order flow on the price formation model. A black-box model interpretation technique was used to reveal the importance of each of the feature in the performance of the model when predicting directional price movements. The results are interesting as they invite both support and contradiction from the literature on the price formation process. Some features that were in this past-published work empirically found to have significant impact on price formation, such as such as inter-arrival times and order size, are shown here to be not too important. However, the importance discovered here of other features, such as the direction or price of the order flow, is shown to correlate with findings in the literature. Due to the encouraging performance of the deep order flow model, the results here may be considered to be significant and to provide an interesting data-driven view of market microstructure that, with better interpreter techniques, could potentially open up new avenues for market microstructure research.

Now that it has been established that the utilising the order flow to train a recurrent neural network results in significant improvements in model performance, the next step would naturally be to improve the use of such models within an automated trading strategy. However it should be noted that in applications of machine learning to quantitative finance, standard deterministic classification or regression outputs are rarely useful in practice. Due to the uncertain nature of the real-world and the high stakes involved, a predictive model should rarely or never be used on its own but should be integrated within a strict workflow to ensure losses are kept in check. In the next chapter, an extension to the architecture of the deep recurrent models described in this chapter that is able to provide a more useful kind of predictions, one enabling a stricter control of

potential losses, will be introduced and tested.

Chapter 4

Deep Probabilistic Modelling of Price Movements

This chapter introduces a deep recurrent architecture for the probabilistic modelling of high-frequency market prices, which would provide a more realistic representation of real markets than the model of the previous chapter, and be more useful for automated trading strategies. The novel architecture incorporates probabilistic mixture models into deep recurrent neural networks, resulting in deep mixture models. The deep mixture models introduced in this chapter simultaneously address several practical challenges in the learning of the price formation process that were previously neglected in the literature: 1) probabilistic forecasting of the price movements; 2) joint modelling of direction and size of price movements. Models derived from the novel architecture are benchmarked against related models in previous work. The joint modelling of direction and size is justified by benchmarking against an approach modelling these variables separately. The chapter is concluded by examining the performance of the deep mixture models when used within an automated trading strategy within a simulated trading scenario.

4.1 Introduction

When designing and developing an automated trading strategy (ATS) in practice, *risk management* is arguably a more crucial part of the pipeline than the predictive model. In a complex dynamical system such as a financial market, it is sensible to expect that even the best trading algorithms, that incorporate state-of-the-art models and are driven by the most expensive and exclusive datasets, will suffer losses at some point. If risk is improperly managed, for example by holding large positions in assets with high variance, the trader may rapidly lose all their capital. When the forecasting component of an ATS has discovered a trading opportunity, the information is passed on to the risk management component to quantitatively determine (based on current trade positions and capital) if the trading opportunity should be taken and, if so, how much capital should be assigned to this opportunity. In order to make these decisions, uncertainties surrounding the trading opportunities need to be known. These uncertainties can either be extrinsically obtained through the forecast errors of historical backtesting, or intrinsically obtained from *probabilistic forecasts*.

Previous work on the application of deep learning to predicting high-frequecy price movements, described in Section 2.3.3, are focused on producing deterministic forecasts, meaning that for risk management the trader has to rely on the less useful measure of uncertainty that is extrinsically obtained from historical backtesting. In the absence of probabilistic forecasts, the authors in [30] introduced a method of using the backtesting false positives and false negatives rates as predictive uncertainty estimates to compute profit risk. Though uncertainties obtained from backtesting can be used for risks management in certain trading strategies, such as the long-short strategy [93], probabilistic forecasts have more flexible use in that they can directly be fed into a much greater variety of industry standards such as the computation of Value at Risk [94] and the Kelly Criterion[95]. In the wider area of machine learning, only the authors in [96] have studied at the probabilistic modelling of price movements using Generalised Linear Models, though not at high-frequency.

Probabilistic forecasts can be obtained from deep learning models either by exploiting dropout to compute uncertainty [97], or by adopting a probabilistic network architecture. However, a probabilistic architecture approach is arguably better as it allows for the specification of application-suitable probabilistic models and likelihood functions, and also reduces the time it takes to produce the probabilistic forecast, which is a crucial factor in time-sensitive trading strategies. Therefore in Section 4.2, a novel architecture that combines deep learning and statistical modelling is introduce to address the main challenge of the probabilistic forecasting of high-frequency price movements. In the novel architecture, a deep recurrent neural network is used as functional approximator for the parameters of mixture models. The selection of mixture models implemented for the experiments in this chapter is presented in Section 4.3. Similar probabilistic architectures [98] have been shown to be successful in the domains of e-commerce [99, 100] and language-processing [101]. The novelty of the architecture introduced in this chapter compared to these related work is in its specific design of the architecture for producing probabilistic forecasts suitable for use in automated trading decision-making workflows. Also, the use of such an architecture provides a bridge between non-parametric approaches and parametric approaches to the statistical modelling of price movements, and hence provides the advantage of both types of modelling approaches.

When it comes to benchmarking the performance of the novel architecture, it will be difficult to compare against the previous work implementing deep learning models described in Section 2.3.3 since all of the output of these models are limited to the deterministic directional predictions. So in this chapter, different benchmark to Section 3.3 of Chapter 3 are defined to enable the comparison of deep mixture models to comparable methods in the literature. These benchmark models are described in Section 4.4. The deep mixture models implementing the novel architecture, and the benchmark models are trained and tested on an order flow dataset described in Section 4.5. The main results comparing the deep mixture models and the benchmark models are then presented in Section

4.6. To briefly outline the results, the models implementing the novel architecture is shown to outperform the models from previous work with statistical significance.

In the machine learning of price movements, it is more common to treat price predictions as a classification problem than a regression problem. Knowing if the price will go up or down is vital as it provides the all-important information as to which position (to buy or to sell) to take on a trading opportunity. Treating price prediction as a regression problem does not tell the trader which direction to bet against. However, along with this directional information, knowing also by how much the price will go up or down (i.e. the size of the movements) will give the trader a much more complete picture of risk and therefore enable better decision making. The attribute of the mixture models in the novel architecture naturally enables the joint modelling of the direction and size of the price movements, and addresses this issue as a secondary challenge. Almost all previous deep learning papers on this topic in Section 2.3.3 are focused only on the *directional* price movements and do not provide any output on the size of the price movements. The only exception is [32], where the output of the model proposed by them can be used to compute the size of the price movements. The joint modelling of the direction and size could provide a more complete prediction model, and improve model performance. Section 4.7 demonstrate that the joint modelling of the direction and size using the deep mixture models learns a better price formation model compared to modelling them separately. To demonstrate this,

At the end of the chapter in Section 4.8, a simulated trading scenario is implemented to examine the full potential of probabilistic joint directional-size predictions of the deep mixture models. The same trading strategy but instead using the output of the benchmark models in Section 4.4 are also examined alongside. The trading strategy implements the Kelly Criterion, which is suitable in this experiment since it relies on both the probabilistic component and the joint directional-size component of the model to compute an optimal investment. The

profitability of the trading strategy using each of the deep mixture model and the benchmark models are investigated over multiple sample runs of the simulated trading scenario. In brief, the trading strategy is shown to be significantly more profitable when utilising the output of the deep mixture model, with statistical significance over the multiple runs.

4.2 Method

4.2.1 Problem Formulation

The *order flow* is the microsecond stream of events arriving into an exchange. Each event is essentially an action taken by a trader, such as placing and cancelling orders. The order flow is the underpinning mechanism behind all the changes we see on the price charts one would find on Google Finance, for instance. Readers are referred to Section 2.1 for a more in-depth explanation of the order flow.

An order flow of length m is denoted here as an irregularly spaced sequence denoted as $\mathbf{x}_{i,1:m} =< \mathbf{x}_{i,t} | t \in \{1,\dots,m\} >$. The aim is to predict the price change after the last event of the order flow $\mathbf{x}_{i,m}$. The word ”after” is loosely defined here such that the price change could be caused by the next event, or be the price change τ seconds after the last event in the stream, and so on. Denoting the price change as $y_{i,m}$, the goal is to then model the conditional distribution of $y_{i,m}$

$$P(y_{i,m}|y_{i,1:m-1}, \mathbf{x}_{i,1:m}, \mathbf{x}_i^{(s)}), \tag{4.1}$$

where $\mathbf{x}_i^{(s)}$ are static (non-temporal) covariates, $\mathbf{x}_{i,:m}$ are the non autoregressive temporal covariates and $y_{i,1:m-1} =< \mathbf{y}_{i,t} | t \in \{1,\dots,m-1\} >$ are the autoregressive temporal covariates. Note here that since the temporal covariates $\mathbf{x}_{i,1:m}$ have irregularly spaced intervals, each of the auto-regressive covariates $y_{i,1:m-1}$ may be computed relative to the timestamp of the temporal covariates. For example, if $y_{i,m}$ is the price change τ seconds after $x_{i,m}$, then $y_{i,m-1}$ is the price change τ seconds after $x_{i,m-1}$.

High-frequency price movements are inherently discrete since the rules laid

down by almost all asset exchanges typically define a *tick* size, which is the minimum amount by which prices can change [35]. Therefore, $y_{i,m} \in \mathbb{Z}$ and the problem can be formulated as one of modelling count data (i.e. counting by how many ticks the price has moved up or down).

4.2.2 Network Architecture for Probabilistic Modelling

The novel architecture in this chapter for modelling the conditional distribution in Equation 4.1 is summarised in Figure 4.1.

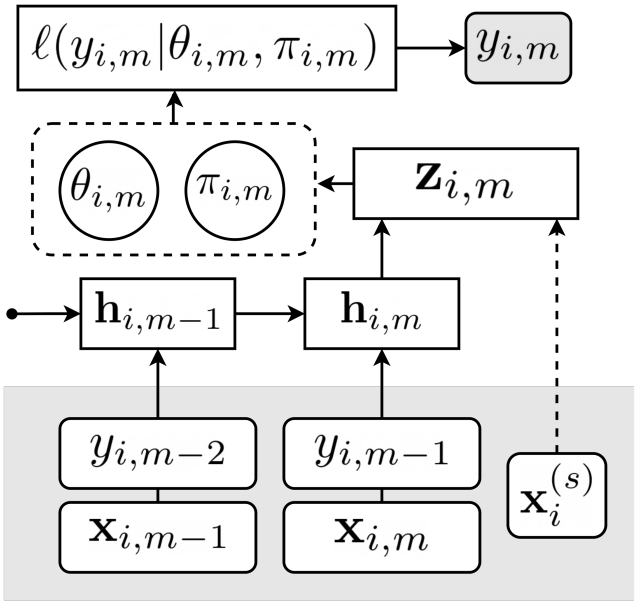


Figure 4.1: Probabilistic architecture.

Let $h(.)$ be a function implementing a deep recurrent neural network (RNN) as described in Section 2.2.2. LSTM cells are implemented in the recurrent layers here. First, the abstract representations of the temporal covariates $\mathbf{x}_{i,1:m}$ and $y_{i,1:m-1}$ are learnt using an L-layer stacked recurrent neural network (RNN). The output at each layer l can be described as follows

$$\mathbf{h}_{i,m}^{l} = \begin{cases} h(\mathbf{h}_{i,m}^{l-1}, \mathbf{h}_{i,m-1}^{l}, \Theta^{l}) & \text{if } 1 < l \leq L, \\ h(y_{i,:m-1}, \mathbf{x}_{i,m}, \mathbf{h}_{i,m-1}^{l}, \Theta^{l}) & \text{if } l = 1, \end{cases} \tag{4.2}$$

where Θ_l are the neural network parameters associated with layer l. Any individual covariates $x_{i,m}^{j} \in \mathbf{x}_{i,m}$ that are non-ordinal and categorical are embedded

beforehand into a multidimensional continuous space before feeding them into the inputs of the RNNs. This embedding, denoted $\tilde{x}_{i,m}^{j}$, is implemented as follows:

$$\tilde{x}_{i,m}^{j} = g\left(\mathbf{W}^{j\mathsf{T}} o(x_{i,m}^{j}) + \mathbf{b}^{j}\right), \tag{4.3}$$

where $o(.)$ is a function mapping the categorical features to one-hot vectors, $g(.)$ is some non-linear activation function, and $\mathbf{W}_q$ and $\mathbf{b}_q$ are parameters to be fitted.

How the static covariates $\mathbf{x}_i^{(s)}$ are treated depends on their cardinality. If the cardinality is small, the static covariates can then be appended at each point in the sequence and then repeated over the whole sequence. On the other hand, if the cardinality is large then the aforementioned method would be inefficient and instead the static covariates would be fed as x_i into a dense layer implemented similarly to Equation 4.3, but without the one-hot mapping function.

Whether or not it is concatenated with $h_i^{(s)}$, $\mathbf{h}_{i,m}^{L}$ is then fed into a D-layer fully connected neural network for a final abstraction step. The outputs at each layer d of this step are computed as follows

$$\mathbf{z}_{i,m}^{d} = \begin{cases} g\left(\mathbf{W}^{d\mathsf{T}} \mathbf{h}_{i,m}^{L} + \mathbf{b}^{d}\right) & \text{if } d = 1, \\ g\left(\mathbf{W}^{d\mathsf{T}} \mathbf{z}_{i,m}^{d-1} + \mathbf{b}^{d}\right) & \text{if } 1 < d \leq D, \end{cases} \tag{4.4}$$

where $\mathbf{h}_{i,m}^{L}$, $g(.)$ is some non-linear activation function, and W_l and b_l are parameters to be fitted.

Now setting aside the architecture preliminaries, to obtain a probabilistic forecast suitable for use in an automated high-frequency trading strategy, a novel integration of mixture models [102] to the output of the architecture is introduced for describing the distribution of the price movements $y_{i,m}$. The mixture probabilities $\pi_{i,m}$ and the parameters $\theta_{i,m}$ of the probability distributions are defined as functions of the dense layer output $\mathbf{z}_{i,m}^{D}$. The model is then fitted by minimising the negative log-likelihood of $y_{i,m}$ given $\pi_{i,m}$ and $\theta_{i,m}$ using the Adam

optimisation as described in Section 2.2.3. Regularisation is achieved via a combination of dropout and early-stopping. To obtain the optimal hyperparameters, Bayesian hyperparameter tuning [83] is implemented.

4.2.3 Covariates and Target Variable

The temporal covariates $\mathbf{x}_{i,:m}$ used in the probabilistic architecture for the experiments in this chapter are the order flow as described in Chapter 3, but with some naturally static features removed. The temporal covariates $\mathbf{x}_{i,\gamma}^{(j)}$ for all $\gamma \in \{1,2,\dots m\}$, with $j \in \{1,2,\dots 5\}$, are therefore defined as follows:

- $x_{i,t}^{(1)}$ is the number of milliseconds between the arrival of $\mathbf{x}_{i,\gamma-1}$ and $\mathbf{x}_{i,\gamma}$, such that $x_{i,t}^{(1)} \geq 0$;
- $x_{i,t}^{(2)}$ is the size of the order, such that $x_{i,t}^{(2)} > 0$;
- $x_{i,t}^{(3)} \in \{1,2,3\}$ is the categorical variable for the type of the order (i.e. limit price orders, open price orders, or cancellation orders);
- $x_{i,t}^{(4)} \in \{1,2\}$ is the categorical variable for the side of the order (whether it is a buy order or sell order);
- $x_{i,t}^{(5)}$, such that $x_{i,t}^{(5)} \geq 0$, is the price associated with the action described by the order;
- $x_{i}^{(s),(1)} \in \mathbb{N}$ is the hour of the day in which our target variable the price movement $y_{i,m}$ falls, which can help capture intra-day trading activity seasonality.

The static covariates $\mathbf{x}_{i}^{(s)}$ are non-temporal features that would otherwise be repeated inefficiently across the input temporal periods. For the experiments in this chapter, the only static covariate used is a categorical variable denoting the currency pair (note that unlike in Chapter 3, the currency pairs are combined and used to train the model together, for reasons explained in Section 4.5.

Given a sequence of orders $\mathbf{x}_{i,:m}$, the target variable for prediction is the price change $y_{i,m}$ at τ seconds after the arrival of the last order in the sequence

$\mathbf{x}_{i,m}$. For the experiments in this chapter, $\tau = 15$ seconds is chosen based on domain knowledge. If τ is too small, there would be many null events and the target variable will be bloated with zeroes which would add another layer of complexity to the modelling problem. If τ is too large, predicting too far into the future could be difficult when working with microsecond timestamped order flow data, and might also render the model unworkable in a high-frequency trading scenario.

4.3 Mixture Likelihoods

The choice of the type of mixture models, and consequently the likelihood function, can be adapted to the statistical properties of the data. Recall that the problem of predicting prices is that of modelling count data, and therefore the Poisson and other similar models will be suitable here. For the experiments in this chapter, three different models will be considered at the output for modelling of mixtures of count data: 1) the Poisson mixture model, 2) the Negative Binomial mixture model, 3) the Zero-Truncated Poisson mixture model.

The most common and straightforward approach to modelling count data is to use the standard Poisson distribution [103], where equal means and variances are assumed. Denoting the mixture probability for component k as $\pi^k_{i,m}$, the Poisson rate parameter for component k as $\lambda^k_{i,m}$, and the number of mixture components as K. Also, let $k = 1$ correspond to downward price movement and $k = 2$ to upward price movement. Given the dense layer output of the recurrent neural network $\mathbf{z}^D_{i,m}$, the log-likelihood ℓ_P of the 2-component Poisson mixture model can then be defined as follows, with i, m labelling omitted for readability:

$$\pi^k = \frac{e^{\mathbf{W}^{(\pi^k)^\intercal}\mathbf{z}^D + \mathbf{b}^{(\pi^k)}}}{\sum^K e^{\mathbf{W}^{(\pi_k)^\intercal}\mathbf{z}^D + \mathbf{b}^{(\pi_k)}}}, \tag{4.5}$$

$$\lambda^k = log(1 + e^{\mathbf{W}^{(\lambda^k)^\intercal}\mathbf{z}^D + \mathbf{b}^{(\lambda^k)}}), \tag{4.6}$$

$$\ell_P(y|\pi,\lambda) = log\left(\sum_k^K \pi^k \frac{\lambda^{k|y|} e^{-\lambda^k}}{|y|!} \mathbb{I}_{p(y)=k}\right), \tag{4.7}$$

In the above equations, $W^{(.)}$ and $b^{(.)}$ are neural network parameters to be fitted, $p(.)$ is a function mapping the sign of $y_{i,m}$ to k, and $\mathbb{I}_{(.)}$ is an indicator function for the given statement.

However, the assumption of equal means and variances does not hold up well on most-real world datasets. Much of the real-world data exhibits *overdispersion*, where the variance is higher than the mean; where the Poisson distributions hence become unsuitable since they do not specify a variance parameter. In this case an alternative approach should be considered, such as to instead use the standard Negative Binomial distribution [103]. Using a similar notationa; style as in the Poisson mixture model definitions above, and letting $\mu_{i,m}^k$ and $\alpha_{i,m}^k$ be the mean and shape parameters respectively, the log-likelihood ℓ_{NB} of the two-component Negative Binomial mixture model can be defined as follows (with i,m once again omitted for readability):

$$\mu^k = log(1 + e^{\mathbf{W}^{(\mu^k)^\intercal}\mathbf{z}^D + \mathbf{b}^{(\mu^k)}}), \tag{4.8}$$

$$\alpha^k = log(1 + e^{\mathbf{W}^{(\alpha^k)^\intercal}\mathbf{z}^D + \mathbf{b}^{(\alpha^k)}}), \tag{4.9}$$

$$\ell_{NB}(y|\pi,\mu,\alpha) = log\left(\sum_k^K \pi_k \eta_1 \eta_2 \mathbb{I}_{p(y)=k}\right), \tag{4.10}$$

$$\eta_1 = \frac{\Gamma(|y| + \frac{1}{\alpha^k})}{\Gamma(|y|+1)\Gamma(\frac{1}{\alpha^k})}, \tag{4.11}$$

$$\eta_2 = \left(\frac{1}{1+\alpha^k\mu^k}\right)^{\frac{1}{\alpha^k}} \left(\frac{\alpha^k\mu^k}{1+\alpha^k\mu^k}\right)^{|y|}, \tag{4.12}$$

In the above equations, $W^{(.)}$, $b^{(.)}$ and $p(.)$ are defined as in the Poisson mixture model. The method of computation for the mixture probabilities π_k is not defined

here since it is exactly the same as in Equation 4.5.

While both the Poisson and Negative Binomial mixture models implicitly allow for the target variable to be zero (no price change), they do not explicitly model $y_{i,m} = 0$. Therefore, this chapter also proposes to experiment with a three-component Zero-Truncated Poisson [104] mixture model to explicitly model zero price changes with an additional component. With mixture component $k = 3$ representing $y_{i,m} = 0$, the likelihood ℓ_{ZP} of this mixture model can be defined as follows (omitting i, m once again):

$$\ell_{ZP}(y|\pi,\lambda) = log\left((\pi^3 \mathbb{I}_{p(y)=3} \sum_{k}^{2} \pi^k \frac{\lambda^{k|y|}}{(e^{\lambda^k}-1)|y|!} \mathbb{I}_{p(y)=k}\right), \tag{4.13}$$

where the mixture probabilities π^k and rate parameters λ^k are as defined in Equations 4.5 and 4.6.

4.4 Benchmark Models

As previously noted, since the benchmark models used in Chapter 3 are deterministic they cannot be used for direct comparison here and hence the benchmark models used in this chapter have to be redefined. Currently there is no appropriate deep learning model in the literature that can be directly used in this chapter to produce probabilistic forecasts of asset price movements. Thus the benchmarks will be chosen from the next best relevant approaches in the literature for probabilistic prediction of high-frequency price movements. Specifically, the following two probabilistic approaches are implemented.

4.4.1 Benchmark 1: Poisson Mixture GLM

[96] is the only other work in the machine learning literature that produces probabilistic outputs with the same form as the novel architecture. The authors described a Poisson mixture regression approach for predicting the size and direction of price movements using a generalised linear model (GLM). In this paper, the only covariate used as a predictor is the total order size occurring before a

price movement. For the implementation in this chapter, a number of static covariates found through domain knowledge to be useful for predicting the prices is included alongside the order size. Such features include the VWAP, number of market orders, time features, moving averages, the average spread and the bid-ask elasticity. The GLM is then trained with Elastic Net for regularisation and feature selection.

4.4.2 Benchmark 2: Multiple Poisson Process

The work in [68] is an extremely well established approach for the stochastic modelling of high-frequency price movements. This multiple Poisson process approach has already been described in Section 2.3.2. An algorithm is implemented in this chapter in order to use the multiple Poisson process to produce a probabilistic forecast in a form suitable for direct comparison with the novel architecture. In the algorithm, the multiple Poison process will be fitted for each point in the dataset. Then, order flow samples will be drawn from the fitted process and fed through the limit order book emulator to obtain a distribution over paths of the price movements. Though the algorithm does not explicitly output the order size of each order event, the empirical distribution of historical order is modelled and sampled to obtain the distribution for each order event independently.

4.5 Dataset

The order flow data used for the work in this Chapter is obtained from Coinbase, as described in Section 3.4. Just as in Chapter 3, due to the different data requirements of the model implementing the novel architecture, and each benchmark model, a reliable procedure is needed to ensure both the alignment of the feature set to the same target variable, and fairness in terms of amount of information contained in the feature set. The datasets here are however generated differently compared to in Chapter 3. As mentioned in Section 3.4, for each currency pair the dataset for the dataset has to be constructed by running the raw JSON messages in the implemented limit order book and matching engine.

During this procedure, the dataset is constructed as follows. After a warming-up period, the mid-price is tracked at every 15 seconds starting from the end of the warm-up period, corresponding to the chosen $\tau = 15$, and pushed into a sequence container. At every point the mid-price is tracked, and a length T of the order flow before each current point in time is directly vectorised and pushed into another sequence container. Then the mid-price and order flow sequence containers are paired, and, finally to obtain the datasets for the model in this chapter and the benchmark models, the following is performed for each element in the paired containers:

- The tuple of mid-price directional and size movements are recorded as the target variables. The mid-price size is converted into a count variable representing the number of ticks. Note that half-ticks are possible, in which case the count will be rounded to the nearest integer.

- For the models using the novel probabilistic architecture, the order events are vectorised and then recorded as the temporal covariates. Then the currency pair symbol is noted and converted into a categorical variable as the static covariate.

- For Benchmark 1, GLM covariates are computed from the order events.

- For Benchmark 2, the order events are transformed into a form for easier processing and stored.

In this way, the training rows for each of the order flow and Benchmark 1, and the data for fitting Benchmark 2, are aligned to the same target variable. Also, since the physical period of time from which the sequences are obtained is the same, the amount of information contained in the features is equal, removing any biases due to any model having more exogenous information than the others.

Unlike in Chapter 3, here for each model the datasets for all the currency pairs are concatenated into one large dataset. In Chapter 3, it was shown that

for each currency pair, the results turn out to be quite similar. Therefore, here they are combined to create a richer dataset with the model cross-learning between the information in all three currency pairs. The datasets is then portioned into training, validation and holdout sets between dates as given in Table 4.1. A warm-up period is needed to set an initial state for the limit order book before dataset collection begins. Though the holdout set may seem unnecessarily large, as mentioned in Section 3.5, it is a better alternative to running multiple iterations of the experiment using a sliding window.

Set	From	To
Training	6 Nov 2017	16 Nov 2017
Validation	17 Nov 2017	20 Nov 2017
Holdout	21 Nov 2017	29 Jan 2018

Table 4.1: The start and end dates of the training, validation and holdout sets

4.6 Main Experimental Results

The natural way to evaluate the accuracy of a probabilistic forecast would be to compute the quantile loss [99, 100]. However, computation of the quantile loss requires the quantile function of the random variable to be evaluated. Closed-form derivation of the quantile function for mixture models is quite involved, and out of the scope of this thesis. An alternative would be to use a Monte Carlo approach where samples are obtained from the mixture models and the quantile loss is evaluated on the empirical cumulative distribution function. But in this case it would still be difficult to evaluate the performance of the models using the quantile loss since the importance of predicting the correct direction of the price movement is not accounted for. If the model assigns very low mixture probabilities to the right direction, that means there would be a high risk of betting in the wrong direction (e.g. buying instead of selling when asset price is going down). Computation of the quantile loss for evaluation is therefore problematically complex, however approached.

Instead, for the experiments in this chapter a two step evaluation procedure is implemented to test the performance of the uncertainty estimates in the

probabilistic forecasts:

1. First, the *directional risk* is defined as the performance measure describing how well the models predict the direction of the price change. The directional risk is evaluated by taking the mixture component with the highest probability to obtain directional point forecasts. The problem of evaluating the directional risk has in this way been reduced to the standard machine learning problem of comparing classification performance. The Matthews Correlation Coefficient (MCC), described in Section 2.2.4, is used to evaluate the directional performance.

2. Next, the *size risk* is defined as as how well the models predicts by how much the price changes. To compute the size risk, for every directional point forecast that is correct, the quantile loss for the associated distribution is evaluated. In other words: if the model gets the direction of the price movement right, how good is the subsequent prediction of the size of the movement? The ”goodness” of the prediction at different quantiles of the prediction distributions can be evaluated from the quantile loss. Since the quantiles for Poisson and Negative Binomial distributions are well-known, let $\hat{y}^{\rho}_{i,m}$ be the computed ρ quantile for the predicted size of the price movements. Then, given the true value $y_{i,m}$ the ρ quantile loss is defined as the following:

$$L_{\rho} = (y_{i,m} - \hat{y}^{\rho}_{i,m})(\rho \mathbb{I}_{\hat{y}^{\rho}_{i,m} > y_{i,m}} - (1-\rho)\mathbb{I}_{\hat{y}^{\rho}_{i,m} \leq y_{i,m}}), \tag{4.14}$$

   where $\mathbb{I}_{(.)}$ is an indicator function for the given statement. For the Zero-Truncated Poisson mixture model, only the quantile loss using Equation 4.14 is evaluated if non-zero directional price movements are correctly predicted.

It is noted here that although turning the probabilistic forecasts into point forecasts in Step 1 appears to defeat the purpose of the probabilistic architec-

ture, it is a temporary compromise that is needed in order to be able to benchmark the performance of the model using standard, well-established and understood metrics. However, in Section 4.8, the models are put through a simulated trading scenario to fully evaluate the probabilistic forecasts.

4.6.1 Results 1: Directional Risk

The directional risk for each model, in terms of the average MCC, is summarised in Table 4.2. Here the results in the holdout period are split into groups, one for the Pre-Bubble period and the other for the Bubble period. The Pre-Bubble period in the first week of December 2017 has trading behaviour similar to the training and validation period, while the Bubble period is from the second week of December 2017 onward and exhibits an increase in trading volatility as described in Chapter 3. The labels Poisson, Zero-Truncated Poisson (ZTP) and Negative Binomial (NB) refer to the form of mixture model used in the architecture of this chapter, while the benchmark models are as described in the previous section. For predicting the direction of the price change 15 seconds into the future, it can be seen in the table that the probabilistic deep learning approach outperforms the benchmark models in both the Pre-Bubble and Bubble period. Also it can be observed that the deep learning model also greatly outperforms the benchmarks by largely maintaining performance through the Bubble period, as would be expected based on the results on the temporal stability of an order flow trained model from Chapter 3.

Model	Pre-Bubble	Bubble
Poisson	0.12	0.10
Negative Binomial	0.14	**0.14**
Zero-Truncated Poisson	**0.16**	0.13
Benchmark 1	0.06	0.01
Benchmark 2	0.08	0.02

Table 4.2: Average MCC of the deep learning model and benchmarks for $\tau = 15$ in the bubble and pre-bubble holdout periods.

Comparing the different mixture models used in the output of the novel probabilistic architecture, the standard Poisson output is seen to be the least

effective in modelling the directional price movements, while the ZTP output is best. Although only the performance on the directional forecast is compared here, one might assume that Poisson, NB and ZTP would be comparable since the directional forecast relies only on the mixture probabilities. However, the mixture probabilities and the parameters of the distribution of each component are jointly trained through the likelihood. There is then a complex interaction between the estimated component distribution parameters and the estimated mixture probabilities, which for the same datapoint can lead to very different mixture probabilities being learned in individual models. Reasons why ZTP and NB outputs outperform Poisson here may be because of the explicit modelling of zero price changes in the mixture, for ZTP, and the ability to account for overdispersion in the component probability distributions for NB.

Although Table 4.2 clearly shows the better performance of the novel architecture, for a more rigorous analysis, one-tailed Welch t-tests are performed between each of the models using the novel architecture, and each of the benchmark models. For each test, the null hypothesis is stated as $H0 : B <= A$ where B is the mean of the MCC measured on one of the deep learning models across the holdout set, while A is the mean for a given benchmark model. If the null hypothesis is rejected, then the alternative hypothesis $H1 : B > A$ is supported. The results of the statistical tests are presented in Table 4.3. It can be observed that for each test, the null hypothesis is rejected at a very high confidence interval, confirming that each of the deep learning models performs better than the benchmark models with very high statistical significance.

Models	p-values	
	Benchmark 1	Benchmark 2
Poisson	0.00	0.00
Negative Binomial	0.00	0.00
Zero-Truncated Poisson	0.00	0.00

Table 4.3: Results for one-tailed Welch t-tests on the null hypothesis that the means of the MCCs for the models in each row are less than or equal to the means of the models in each column, rounded to two decimal places.

4.6.2 Results 2: Size Risk

Table 4.4 shows the average 0.5 and 0.9 quantile loss, across the holdout set split into the Pre-Bubble and Bubble periods, of the size movement distribution when the mixture components are correctly predicted. Since the quantile losses have an arbitrary scale that is meaningless on their own, it is better to make comparison by scaling the results in the table to a baseline model. For this purpose, Benchmark 1 is chosen as the baseline model as, between the two benchmarks, it has outputs most directly comparable with those of the deep learning models. At a glance, it can be seen in Table 4.4 that that all the models using the novel architecture outperform the benchmark models.

	q=0.5	
Model	Pre-Bubble	Bubble
Poisson	0.78	0.77
Negative Binomial	**0.65**	**0.61**
Zero-Truncated Poisson	0.70	0.68
Benchmark 1 (Baseline)	(1.00)	(1.00)
Benchmark 2	1.23	1.29
	q=0.9	
Model	Pre-Bubble	Bubble
Poisson (Baseline)	0.72	0.70
Negative Binomial	**0.60**	**0.56**
Zero-Truncated Poisson	0.72	0.71
Benchmark 1 (Baseline)	(1.00)	(1.00)
Benchmark 2	0.98	0.97

Table 4.4: 0.5 and 0.9 quantile loss of the deep learning models and benchmarks in the bubble and pre-bubble test periods, scaled to baseline model

Comparing the different mixture models used in the output of the novel architecture, it is noted that NB outperforms both ZTP and Poisson. The reason for this may be the ability of the deep NB mixture model to model the overdispersion in population of price movement sizes. To understand this, the performance of each model between the Pre-Bubble and Bubble periods are compared. As the market becomes volatile, the better performance of the deep learning models in relation to the baseline model increases. However, this increase in relative performance is quite small for the Poisson mixture. The highly volatile behaviour

of the market in the Bubble period may have caused a higher degree of overdispersion to arise from the clustering of the price changes. Hence this may have caused the deep Poisson mixture model, which does not have a variance parameter, to underperform compared to the deep NB model, which specifically models the variance parameter.

On the other hand, the ZTP mixture underperforms here even though directly modelling zeroes in the mixture probabilities in this model can reduce the overdispersion in the data. Since it performed well previously in predicting the directional price movements, the successful modelling of the zeroes should have helped with the overdispersion, but it would seem from Table 4.4 that it is clearly not sufficient. Also, looking at the 0.9 quantile, it can be observed that overall the deep learning models have a lesser tendency to overpredict compared to both benchmarks. Comparing the different deep mixture models, it is observed that ZTP tends to overpredict more often, which may be due to the fact that zeroes are truncated in its likelihood (i.e. distributions model $y_{i,m} > 0$).

Finally, to remark on the relative performance of the benchmark models, it would seem that Benchmark 1 suffers slightly from over-predicting compared to Benchmark 2, from observing the 0.9 quantile loss. However, both benchmark models underperform at about the same rate in the volatile Bubble period compared to the Pre-Bubble period.

As before, though Table 4.4 clearly shows the better performance of the deep learning models, a more rigorous analysis is performed using one-tailed Welch t-tests. For each test, the null hypothesis is stated as $H0 : B <= A$ where B is the mean quantile loss measured on one of the deep learning models across the holdout set, while A is the means for a given benchmark model. If the null hypothesis is rejected, then the alternative hypothesis $H1 : B > A$ is supported. The results of the statistical tests are presented in Table 4.5. As expected, it is observed that the null hypothesis is rejected for each test at a very high confidence interval.

	q=0.5	
Model	Benchmark 1	Benchmark 2
Poisson	0.00	0.00
Negative Binomial	0.00	0.00
Zero-Truncated Poisson	0.00	0.00
	q=0.9	
Model	Benchmark 1	Benchmark 2
Poisson (Baseline)	0.00	0.00
Negative Binomial	0.00	0.00
Zero-Truncated Poisson	0.00	0.00

Table 4.5: For quantiles 0.5 and 0.9, the results for one-tailed Welch t-tests on the null hypothesis that the means of the quantile loss for the model in each row are less than or equal to the means of the models in each column, rounded to two decimal places.

4.7 Comparing Against Two Separate Models

In this section the joint modelling approach in the deep mixture model will be defended by comparing its performance with the obvious alternative of two deep recurrent models, predicting the direction and the size of the price movements respectively. Both of these component models are supplied with exactly the same dataset as the deep mixture model.

The first model is the deep recurrent model used in Chapter 3, and described in Section 2.2.2, for predicting the direction the price movements. Here there is an additional currency pair categorical variable, since now the model is trained using a dataset containing all the currency pairs combined. This model is essentially just the recurrent component of the architecture of this chapter, without the mixture or static components, but with a softmax output and the cross-entropy loss described in Section 2.2.3.

For predicting the size of the price movements, the same recurrent model in Chapter 3 is used once again, along with the currency pair categorical variable addition as with the directional price prediction model above. However, the loss function here is changed to the Negative Binomial loss, which provides a probabilistic output to model the size of the price movements. This loss is defined as follows:

$$\ell_{NB}(y|\mu,\alpha) = log(\frac{\Gamma(|y|+\frac{1}{\alpha})}{\Gamma(|y|+1)\Gamma(\frac{1}{\alpha})}\eta_2)\left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}}\left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{|y|}, \quad (4.15)$$

where, α and μ are parameters to be estimated by the deep recurrent model.

4.7.1 Results

It can be observed from Table 4.6, showing the mean MCC for the Bubble and Pre-Bubble period of the holdout set, that the architecture of this chapter does provides a small advantage in predicting the directional price movement. For the novel architecture, the table only shows the best MCC between the three different forms of the novel architecture. The statistical significance of the results here is proven using once again the one-tailed Welch t-test. The null hypothesis here is $H0 : B <= A$ where B is the mean MCC of the novel architecture across the holdout set, while A is the means for the benchmark. The resulting p-value was found to be 0.02, meaning that the results are reasonably significant.

Model	Pre-Bubble	Bubble
Novel Architecture	0.16	0.14
Non-Mixture Benchmark	0.15	0.14

Table 4.6: Mean MCC comparison between best of the novel architecture and non-mixture benchmark the bubble and pre-bubble holdout periods.

Next, Table 4.7 shows the quantile loss of the novel architecture scaled to the quantile loss of the deep negative binomial model, as a baseline. The table only shows the best loss between the three different form of the novel architecture. Here, a stark difference to the direction prediction is observed. The architecture of this chapter provides a large improvement to the probabilistic size predictions and this is especially clear during the Bubble period. A reason for the better performance of the novel architecture points to the fact that the size of price movements, especially near and during volatile periods, can be very dependent on the direction of the price movements. By jointly modelling the direction, the novel architecture is able to better learn the magnitude of the price

movements. The results in Table 4.7 are shown to be significant with p-values very close to zero using the one-tailed Welch t-test. The null hypothesis here is $H0 : B <= A$ where B is the mean quantile loss of the novel architecture across the holdout set, while A is the means for the benchmark.

Model	q=0.5		q=0.9	
	Pre-Bubble	Bubble	Pre-Bubble	Bubble
Novel Architecture	0.81	0.64	0.73	0.60
(Baseline)	(1.00)	(1.00)	(1.00)	(1.00)

Table 4.7: 0.5 and 0.9 quantile loss of the novel architecture, scaled to those of the non-mixture benchmark (as a baseline), in the bubble and pre-bubble holdout periods.

4.8 Application to a Simulated Trading Scenario

In this section, the practical use of the novel architecture and the benchmark models is evaluated by using the models in a simulated trading scenario. Previously in Section 4.6, a compromise was made in order to properly establish a baseline comparison by evaluating the models using standard measures. Here the profitability potential of the full direction-size probabilistic forecasts of each model is demonstrated by using the predictions within a simple automated trading strategy. Simpler trading strategies here are desirable to prevent this chapter from being impeded by any complexity in implementation, and also to improve the transparency of the result analysis. For similar reasons the simulated trading environment will be kept simple as well.

4.8.1 Trading Strategy

Here, an optimal betting approach using the Kelly Criterion [95] with uneven payoffs is used as the simple trading strategy for testing the models. Based on the uncertainties surrounding the direction and size of the price movements, the Kelly Criterion determines the optimal proportion of the capital should be allocated to an investment such that the potential profit is maximised and the potential risk to the total capital are simultaneously minimised. If there is too much uncertainty, a smaller proportion will be allocated and vice-versa. Let's

denote the proportion of capital X_0 allocated to a bet on the movements of the asset as f. Also, let's take the perspective of a long strategy. If the bet bet is won then the capital becomes $X_0(1+b)$, and f the bet is lost then capital becomes $X_0(1-a)$. Therefore the amount of capital after n bets can be computed as follows:

$$X_n = X_0(1-a)^{L_n}(1+bf)^{W_n}, \tag{4.16}$$

where, L_n and W_n are the number of times the bet is lost or won in the n bets respectively. Then, assuming an exponential rate of increase during each bet, the overall profit is:

$$g(f) = q\log(1-af) + p\log(1+bf), \tag{4.17}$$

where, $q = \frac{Ln}{n}$ is the probability of losing the bet and $p = \frac{W_n}{n}$ is the probability of winning the bet. Taking the derivative of Equation 4.17 to maximise $g(f)$ on f gives the Kelly Criterion as follows:

$$f^* = \frac{p}{a} - \frac{q}{b}, \tag{4.18}$$

Since a long strategy perspective is taken here, p and q are the probability of an upward and downward price movement respectively. Hence, π^2 and π^1 from the model in Section 4.3 can be substituted in. Similarly, a can be computed by taking $\frac{\hat{y^1}}{s}$ where, $\hat{y_1}$ is the expected value of the price change if the price were to go down as predicted by the model, and s is the current price of the asset. Conversely, b can be computed by $\frac{\hat{y^2}}{s}$. Then the overall trading strategy at time t can be expressed by:

$$f_t = s_t(\frac{\pi^2_{t+\tau}}{\hat{y}^1_{t+\tau}} - \frac{\pi^1_{t+\tau}}{\hat{y}^2_{t+\tau}})\varepsilon, \tag{4.19}$$

where, ε serves as the risk aversion constant.

Since the above is formulated from the point of view of taking the long strategy, the sign of f_t determines the position to take in the investment. A

positive value of f_t indicates a long position should be taken while a negative value indicates a short position. For simplicity, a few assumptions are made. The first assumption made is that the strategies are able to leverage risk-free loans to invest in the given trading opportunity if $|f_t| > 1$. To simplify analysis it is also assumed that the strategies are able to exit the position cost-free after τ seconds before the trading signal at time $t + 1$ comes in. Finally, an assumption is made that the market does not react to the trading activities of the strategies due to the simple nature of the simulation environment.

4.8.2 Experimental Method

The simulation environment is achieved through a sampling approach, from the holdout set described in Section 4.5. Each strategy starts off with an assigned capital of $\$10,000$. Also it is assumed that the assigned model has been already fitted on the training and validation sets.

To run the experiments, first the datapoints in the holdout sets are sorted by timestamp. Due to the way the datasets are created, no two points are associated with a single point in time. The random sampling of the datapoints is done monotonically such that, if a datapoint at time t has been sampled, then datapoints timestamped before t are exempted from the subsequent sampling process. In each iteration, the sorted holdout sets are sampled and the sampled datapoint for each dataset is then fed to the appropriate strategies. Taking the sampled as a trading signal indicating investment opportunity, each strategy computes the Kelly Criterion to determine what proportion of the capital to invest in this trading opportunity such that the long-term growth of the capital is sustained. This is repeated until T iterations are achieved. This algorithmic procedure from $t = 0$ to T is referred to here as a scenario. Multiple scenarios are run to ensure statistical significance of the results.

4.8.3 Results

The described procedure is used to simulate trading scenarios by random sampling of the datasets with $T = 500$ iterations in each scenario, and $K = 10,000$

scenarios. Figure 4.2 shows the empirical distribution of the capital held by each model at the end of each trading scenario. The values are scaled to a hypothetical baseline model that makes a perfect prediction at each iteration. It can be observed from the figure that the deep models overall do better than the benchmark models. Although the mode of the benchmark model distributions is comparable to those of the deep learning models, the smaller peaks and slight right skew indicates that the benchmark models are often less profitable compared to the deep learning models.

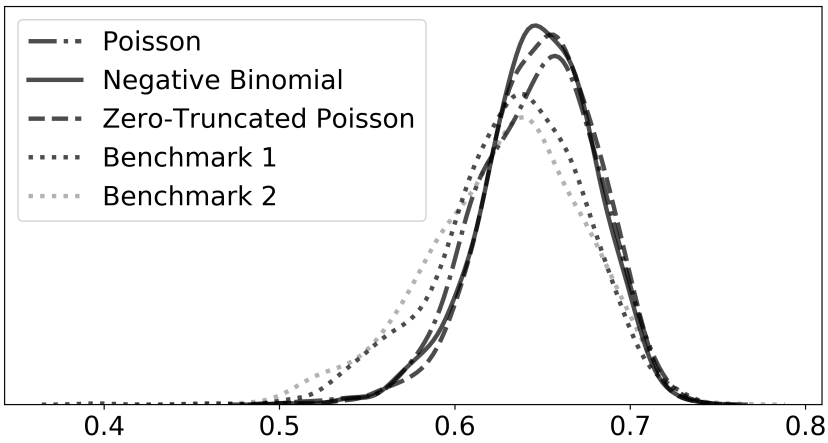


Figure 4.2: Empirical distribution of the final capital held by each model after 500 iterations of trading across 10000 trading scenarios, scaled to a hypothetical perfect prediction baseline model.

To assess the significance of the results in Figure 4.2, paired Student t-tests are performed on the distributions. For each test, the null hypothesis is stated as $H0 : B <= A$ where B is the mean of the final capital of one of the deep learning models, while A is the final capital means for a given benchmark model. If the null hypothesis is rejected, then the alternative hypothesis $H1 : B > A$ is supported. As shown in Table 4.8, the null hypothesis for each test is rejected with very high confidence interval.

To take a closer look at why and how the deep learning models performed better in the trading scenarios, Figure 4.3 illustrates the start-to-end instance of one of the K scenarios. Observing the trajectories before the $t = 300$ mark, the strategy running Benchmark 2 seems like it would be performing better than the

Models	Benchmark 1	Benchmark 2
Poisson	0.00	0.00
Negative Binomial	0.00	0.00
Zero-Truncated Poisson	0.00	0.00

Table 4.8: The p-values of paired Student t-tests on the null hypotheses that the profit distribution for a given benchmark model is no different to the those of a deep mixture model, rounded up to two decimal places.

deep mixture models. However, at around $t = 318$, due to an over-estimation of the Kelly Criterion caused by sub-optimal probabilistic estimations in Benchmark 2, the strategy allocates too much capital into a risky trading opportunity and ends up losing much of its capital. Around the same time, it can be observed that the deep ZTP mixture model also wrongly predicts the direction of the trade but does not suffer as much loss due to a good probabilistic prediction of the price change when computing the Kelly Criterion.

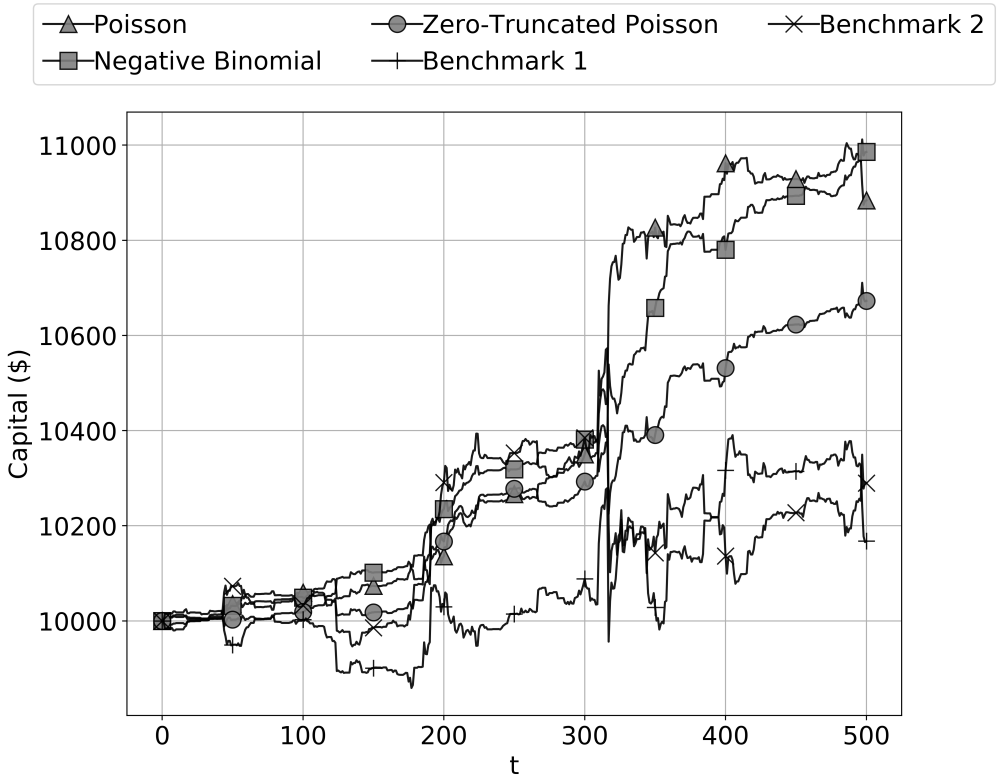


Figure 4.3: A single sample of the simulated trading scenario showing the change in capital due to the trading decisions made by the models at each iteration.

4.9 Summary

In this chapter, a novel deep learning architecture is introduced for the probabilistic direction-size forecast of high-frequency price movements in order to improve the utility of deep recurrent models in automated trading strategies. The novel architecture implements a deep mixture model, which is a middle ground bridging parametric and non-parametric approaches in modelling price formation. To test the novelty and significance of the deep learning models, the results from three different experiments were presented. All results were benchmarked against carefully chosen state-of-the-art models derived from the literature on modelling high-frequency price movements.

The main experiment compared the performance of the novel architecture against the benchmark models using a two-step procedure assessing both the directional risk and size risk using standard performance measures. In Section 4.6 it was shown that the novel architecture outperforms the benchmark models when measured with the Matthews Correlation Coefficient and the quantile loss with high statistical significance as shown in Table 4.3 and Table 4.5. Then, in Section 4.7, the value of the novel mixture model approach is demonstrated by comparing it against two separate deep recurrent models predicting the direction and size respectively. As can be seen in Table 4.7, the performance of the novel architecture is better than the model described in Chapter 3, which was itself established to outperform the state-of-the-art, with confidence. Finally, the practical utility of the novel architecture is demonstrated by examining its profit potential in a simulated trading environment. The probabilistic forecasts produced by the novel architecture result in a higher profit when implemented into an automated trading strategy, the statistically significance of which is shown Table 4.8.

In chapters so far, it has been shown deep learning models that are trained on order flow data have tremendous potential in modelling the price formation process by directly learning to predict high-frequency price movements. To further reach towards the main research objective of improving deep price forma-

tion models, an inspiration is taken from previous work in zero-intelligence models where the order flow sequences themselves are modelled directly. Therefore, the next chapter will introduce the deep generative modelling approach that learns to model the order flow sequences directly, such that the price formation model can constructed in a bottom-up manner.

Chapter 5

Deep Generative Modelling of Order Flow Sequences

This chapter introduces a novel application of a Generative Adversarial Network (GAN) for the generative modelling of order flow sequences. The problem of the generative modelling of order flow sequences is formulated, and the method of applying GANs to the problem is presented. Since the generative modelling of order flow sequences using machine learning has not been previously studied, a suitable order flow model is selected from the theory-driven literature. Sequence similarity algorithms are used to measure how well the each of the models are able to reproduce the real order flow sequences. Then, an analysis of the macro-behaviour of the generated order flow sequences is performed, by studying how well the generated sequences are able to reproduce the statistical behaviours of mid-price time-series that have been reported in the literature.

5.1 Introduction

In the taxonomy of machine learning models, one of the many kinds of model classifications categorises models as either *discriminative* models, or *generative* models. Discriminative models directly estimate the posterior probability $p(y|x)$ of the output y conditioned on the input x. Such models can be employed to predict the price movements of an asset, given some predictive features, as

demonstrated in previous chapters. On the other hand, generative models estimate the joint probability of the data $p(y,x)$, or $p(x)$ if the data has no labels. Though generative models can also be used for prediction tasks by using Bayes' rule to compute the conditional probability of the output, discriminative models are generally preferable. The reason for this is that discriminative models directly solve the classification or regression task of concern, without needing intermediate steps such as modelling the likelihood to compute the posterior for in a generative model, for instance [105]. However, because generative models include the distribution of the data itself, they can be queried for the probability of the data and hence are able to generate new data instances, i.e. simulate an order flow sequence.

In the application of machine learning to asset trading, the focus of the majority of the work in the literature is the use of discriminative models for the prediction of asset prices. This general focus on predicting price movements is completely justifiable as the very basis of trading is to buy low and sell high. However, there exist a number of machine learning tasks for facilitating trading that would require the use of generative models. The situations in which these are needed may not be as obvious as the prediction of price movements, but they can have as much of an impact. The most common example of tasks that would utilise generative model, would be the modelling of financial sequences to generate realistic simulated sequences. Such simulated sequences have a number of practical uses in quantitative trading, with a few examples as follows:

- The simulated order flow sequences can be used to predict price movements at any desired timescales. Using the simulated sequences as input to an existing order book that has been tracking the order flow, prices can be observed throughout the length of the simulated sequence. This will not only provide the mid-price, but potentially the spread series, the bid-ask and other price series as well. In addition, multiple runs of the simulated order flow can be sampled to produce a probability distribution of these prices over time.

- Another potential use for simulated order flow sequences in quantitative trading would be to estimate probable order intensities at future points in time. These order intensities are important factors for estimating the prices at which the bid and ask limit orders should be posted when computing market making strategies [13], which are the most common type of strategy employed by high-frequency traders. Order intensities also play an important role in computing the optimal policies [16] of optimal execution algorithms. These order intensities could definitely be predicted directly using discriminative models, but the formulation of such a problem could be a lot more difficult since the order intensities might be needed for hundreds of price levels on either side of the book.

- If a deep generative model is able to model order flow sequences well enough to produce realistic order flow sequences, then it would be very interesting to interrogate such models using black-box explainer algorithms to provide useful insights into how order flow sequences are formed. Similar to the work in Section 3.7, where a deep recurrent model was shown to have promising results in modelling the price formation based on the order flow, interesting machine learning insight into how the order flow affect price formation could be obtained.

- Simulated order flow sequences can be used for evaluating and back-testing trading strategies [106, 107]. Back-testing strategies on historical data might not be as productive since historical data do not react to the actions taken through the strategy policies - especially in high-frequency trading, actions taken via trading strategies could result in very dynamic changes in the market that need to be accounted for when evaluating these strategies. Also, generative models could be used for producing different market scenarios through the simulated order flow. In this case, sequences related to hypothetical scenarios could be observed from the scenario-conditioned simulations and used for stress-testing trading strategies.

Therefore, it is fair to say that problem of modelling order flow sequences using deep learning model is an interesting and valuable one. Indeed in quantitative finance literature, the research into modelling order flow sequences is quite advanced, as can be observed from the previous work detailed in Section 2.3.3. However, there is currently a gap in the machine learning literature in applying deep learning, or any machine learning models at all, to modelling order flow sequences. The work in this chapter aims to fill this niche.

Towards this end, a novel application of deep generative adversarial network is introduced in this chapter to address the challenge of generative modelling order flow sequences. Section 5.2 provides the general background on the deep generative models and the Sequence Generative Adversarial Network (SeqGAN) framework used for the experiments in this chapter. Then, the methodology for applying the SeqGAN framework for generative modelling of the order flow is outlined in Section 5.3. Since there are currently no previous work related to this topic in machine learning literature, Section 5.4 details the model chosen from the quantitative finance literature for benchmarking the performance of the SeqGAN approach for generating modelling order flow. To train these models, Section 5.5 details the dataset extracted from the Bitcoin order flow obtained in Section 3.4. The output of this model is however not straightforward to evaluate since the order flow is by nature non-deterministic. Two order flow sequences that are not exactly equal may have the same affect on trading algorithm when evaluating the algorithm on each of the two sequences. Or, order intensities computed from these two order flow sequences may both result in the same value. There is a similar issue in NLP regarding the issue of evaluating natural language generation [108].

Therefore, the experiments in this chapter will evaluate the output of the generative models from two angles. First, to examine how well the SeqGAN model and the benchmark model are fitted to the order flow data, the output of the models is subjected to sequence similarity tests. This is to determine how well each model is able to reproduce the real order flow. Section 5.6 describes

this experiment. Next, a macro-behaviour analysis is performed in Section 5.7 to investigate how well the emerging properties of the simulated order flow match the statistical properties of the real order flow. This is achieved by comparing various statistical properties of the mid-price resulting from the simulated order flow against the real order flow. These statistical properties are chosen from the financial literature as important descriptors of financial data.

5.2 Technical Background

5.2.1 Deep Generative Models

There have been great strides in the field of deep learning, using deep generative models, for solving the problem of generating artificial sequential data that imitates real data. This includes application areas from the leviathan that is natural image generation [109, 110, 111, 112], to network security [113], or medicine [114], and even dentistry [115]. In the area of sequence generation, the work on deep generative models is focused mainly on natural language generation [116, 44, 117, 118]. In the econometric domain of machine learning, which is more closely related to the work in this thesis, there exists little work on the application of deep generative models for generating artificial sequential data that imitates real data. The reason for this could be lack of obvious use for the generated data compared to other areas such as image generation or natural language generation. There exist papers in the econometric machine learning literature describing the use of generative models for producing sequences [100, 119], but the objective of the work in these papers is the multi-horizon prediction of the future, rather than generating imitation data.

Similarly, there exist some papers on applying generative models for financial data, but focused on the objective of multi-horizon forecasting rather than generating artificial data [120, 121]. An exception here is [122], where the authors aim to study the ability of GANs to generate realistic financial time series, by observing various statistical properties and stylised facts of the artificial time-series generated by their models. However, the authors in [122] focused on

financial time series, while the work in this chapter aims to model the order flow.

Due to the aforementioned diversity of potential applications, deep generative models are an active area of research which has drawn a significant amount of attention. In fact, generative models were studied even before the current explosion of research in deep learning. The authors in [123] first demonstrated the potential of deep architectures by proposing the generative modelling approach of using the contrastive divergence algorithm to efficiently train deep belief nets. Later research into generative models includes de-noising autoencoders [124] and variational autoencoders [125]. These generative models are trained by maximising the likelihood of the training data, which suffers from tractability problems in probabilistic computation.

An alternative framework for training generative models, called the generative adversarial net (GAN), is proposed by the authors of [126]. The GAN framework involves a minimax game between a discriminator model and a generative model to train the generative models. This circumvents the problems associated with maximum likelihood learning in previous generative models. While the standard GAN and its later variants has been shown to be able to generate very convincing images, there were challenges to applying GANs to generate discrete token-based sequences, such as those of natural language, or in the case of this chapter, the order flow. To deal with such data, the SeqGAN framework is proposed by the authors in [34]. The SeqGAN framework is also an improvement over existing approaches in generating token sequences since it does not incur any exposure bias [127] between the training phase and the inference phases, when generating sequences. Besides SeqGAN, there exists another related GAN framework specially designed for sequences, more specifically for generating time-series [128]. However, it is not suited to the objective in this chapter since the order flow is not a typical time-series.

5.2.2 The SeqGAN Framework

Like all generative adversarial networks (GAN) [126] frameworks, SeqGAN is made up of a generator G_θ and discriminator D_ϕ, parameterised by θ and

ϕ respectively. The generator G_θ is trained to produce a sequence $Y_{1:T} = (y_1, \ldots, y_T)$, where $y_t \in \mathscr{Y}$ and $\mathscr{Y}$ is some set of discrete tokens. The discriminator D_ϕ is then a binary classifier trained to distinguish the generated sequence $Y_{1:T}$ from a real sequence $X_{1:T}$. The probability $D_\phi(Y_{1:T})$ of how likely the generated sequence is to a real sequence is used as a feedback for guidance on further improving G_θ. In the problem of generating a simulated order flow, $\mathscr{Y}$ would be $\mathscr{O}$ and $Y_{1:T}$ would be an order flow sequence. In SeqGAN, the learning of the generator G_θ is treated as a reinforcement learning (RL) problem. At any given timestep t, the state s is the sequence produced thus far $y_1, \ldots, y_{t-1}$. The action a is then which token to select as y_t from $\mathscr{Y}$. The action to be taken in a given state is determined by the generator $G_\theta(y_t|Y_{1:t-1})$, which is a stochastic policy parameterised by θ. The generator is updated via policy gradient utilising rewards in the form of the output of the discriminator D_ϕ.

There are two phases in the SeqGAN framework, the pre-training phase and the adversarial training phase. In the pre-training phase, given a dataset of real sequences, the parameters of the generator are fitted using maximum likelihood estimation (MLE). The pre-trained generator is then used to produce negative examples for pre-training the discriminator. Using these negative examples, along with positive examples from real sequences, the parameters of the discriminator are fitted using by minimising the cross-entropy (CE). The objectives when optimising for the CE are described in Section 2.2.3. After pre-training both models, the adversarial training phase can then commence. In the adversarial training phase, the objective of the generator or policy G_θ is to maximise the expected end reward R_t from the initial state s_0. This is essentially the state-value function $V_{D_\phi}^{G_\theta}(s_0)$ of the RL problem. The objective can then be expressed as follows:

$$\begin{aligned} J(\theta) &= V_{D_\phi}^{G_\theta}(s_0) = \mathbb{E}[R_t|s_0, \theta], \\ &= \sum_{y_1 \in \mathscr{Y}} G_\theta(y1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1), \end{aligned} \tag{5.1}$$

where $Q^{G_\theta}_{D_\phi}(s,a)$ is the action-value function of the RL problem. Though it may seem as if the generator is being optimised on just y_1, this is not the case for the following reason. Because the state transition is deterministic, and the intermediate reward is set to zero, the following relation can be derived:

$$Q^{G_\theta}_{D_\phi}(s = Y_{1:t-1}, a = y_t) = V^{G_\theta}_{D_\phi}(s = Y_{1:T}), \tag{5.2}$$

Given this relation, the action-value function $Q^{G_\theta}_{D_\phi}(s_0, y_1)$ inside Equation 5.1 can be rolled out as follows:

$$\begin{aligned} Q^{G_\theta}_{D_\phi}(s_0, y_1) &= V^{G_\theta}_{D_\phi}(Y_{1:1}), \\ &= \sum_{y_2 \in \mathscr{Y}} G_\theta(y_2|Y_{1:1}) \cdot Q^{G_\theta}_{D_\phi}(Y_{1:1}, y_2), \end{aligned} \tag{5.3}$$

where $Y_{1:1}$ is the state s_1. The Equation in 5.3 can be further rolled out recursively in the same way to unroll the whole sequence. This manner of rolling out the objective function is then used by the authors in [34] to obtain the derivative of the objective for learning θ via policy gradient update. Combined with the likelihood ratio trick, the derivative of $J(\theta)$ is computed as follows:

$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{T} \nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}_{D_\phi}(Y_{1:t-1,y_t}), \tag{5.4}$$

The derivative for policy gradient update relies on the action-value function $Q^{G_\theta}_{D_\theta}(s,a)$. To estimate $Q^{G_\theta}_{D_\theta}(s,a)$, the estimated probability of the generated sequence being labelled as real by the discriminator D_ϕ is used. The action value is defined by the authors as follows:

$$Q^{G_\theta}_{D_\phi}(s = Y_{1:t-1}, a = y_t) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y^n_{1:T}), \text{ where } Y^n_{1:T} \in \text{MC}^{G_\theta}(Y_{1:t}; N) & \text{if } t < T, \\ D_\phi(Y_{1:t}), & \text{if } t = T, \end{cases} \tag{5.5}$$

where $\text{MC}^{G_\beta}(Y_{1:t}; N)$ represents an N-times Monte Carlo search [129] algorithm

for sampling the unknown last $T-t$ tokens using the generator G_θ as the rollout policy, and $Y^n_{1:T}$ is a sampled sequence.

After improving the generator via policy gradient update for a number of epochs, the discriminator is then re-trained using negative examples produced from the improved generator. The re-training of the discriminator is achieved by simply minimising the binary cross-entropy loss as follows:

$$\min_{\phi} -\mathbb{E}_{Y\sim p_{data}}[\log D_\phi(Y)] - \mathbb{E}_{Y\sim G_\theta}[\log(1-D_\phi(Y)], \tag{5.6}$$

where p_{data} is the dataset of positive examples of real sequences. After the discriminator is improved, then the generator is once again updated using the rewards from the improved discriminator. This adversarial training where the generator and discriminator try to out-do one another is repeated until convergence.

5.2.3 Convolutional Neural Network

The work in this chapter makes use of the Convolutional Neural Network (CNN) in the SeqGAN framework. Therefore, an introduction to the architecture for the standard convolutional neural network [130, 131] will briefly presented as follows.

The input to the CNN is a sequence of k-dimensional embedded tokens $x_1,\ldots,x_T$, where $x_t \in \mathbb{R}^k$. This input sequence is first assembled into a matrix representation as follows:

$$\varepsilon_{1:T} = x_1 \oplus x_2 \ldots x_T, \tag{5.7}$$

where $\oplus$ is the vertical concatenation operator for assembling the matrix $\varepsilon_{1:T} \in \mathbb{R}^{T\times k}$. Then, a convolutional operation with a window size of l is applied using a kernel $\mathbf{w} \in \mathbb{R}^{l\times k}$ to produce a new feature map:

$$c_i = g(\mathbf{w} \otimes \varepsilon_{i:i+l-1} + b), \tag{5.8}$$

where $\otimes$ is an operator indicating the summation of element-wise product, b is some parameter to be fitted and g is some non-linear activation function. Here, the architecture is not limited to a single kernel. Multiple kernels of different sizes can be used to learn different feature representations. A given kernel $\mathbf{w}$ of size l will produce the following vector of feature maps:

$$\mathbf{c} = [c_1, \ldots, c_{T-l+1}], \tag{5.9}$$

and a max-pooling layer with output $\tilde{c} = max\{\mathbf{c}\}$ is applied for all kernels, resulting in a vector of pooled features $\tilde{\mathbf{c}}$. To enhance the optimisation of the CNN, the highway architecture [132] is added after the max-pooling layer as follows:

$$\begin{aligned} \tau &= \sigma(\mathbf{W}_T \cdot \tilde{\mathbf{c}} + \mathbf{b}_T), \\ \tilde{\mathbf{C}} &= \tau \cdot H(\tilde{\mathbf{c}}, \mathbf{W}_H) + (1 - \tau) \cdot \tilde{\mathbf{c}}, \end{aligned} \tag{5.10}$$

where $\mathbf{W}_T$, $\mathbf{b}_T$, $\mathbf{W}_H$ are network parameters to be fitted, H denotes an affine transformation followed by some non-linear activation function, and σ is the sigmoid function. Finally, to obtain the output probability $\hat{y}$ that the given input sequence is real, a fully-connected layer with a sigmoid activation function is used at the output:

$$\hat{y} = \sigma(\mathbf{W}_o \cdot \tilde{\mathbf{C}} + b_o), \tag{5.11}$$

where, $\mathbf{W}_o$ and b_o are more CNN parameters to be fitted by gradient-descent.

5.3 Method

5.3.1 Problem Formulation

The work in this chapter uses a different form to represent the order flow and the order events, in comparison to previous chapters. Here, order events are defined in the form of discrete event tokens representing bid and ask market, limit and cancellation orders at up to Q relative prices away from the best bid

and best asks respectively. Let $\mathscr{O}$ be this set of event tokens. The set $\mathscr{O}$ can then be defined as follows:

$$\begin{gathered}
\mathscr{O} \in \{\mathscr{L} \cup \mathscr{M} \cup \mathscr{C} \cup \mathscr{E}\}, \\
\mathscr{L} \in \{l_{B,1}, \ldots, l_{B,Q}, l_{A,1}, \ldots, l_{A,Q}\}, \\
\mathscr{C} \in \{c_{B,1}, \ldots, c_{B,Q}, c_{A,1}, \ldots, c_{A,Q}\}, \\
\mathscr{M} \in \{\mu_B, \mu_A\}, \\
\mathscr{E} \in \{\eta_B, \eta_A\},
\end{gathered}$$

where $\mathscr{L}$ is the set of all limit order event tokens, $\mathscr{M}$ is the set of all the market order event tokens, $\mathscr{C}$ is the set of all cancellation event tokens, and $\mathscr{E}$ is the set of all other event tokens. Each event token in $\mathscr{L}$, $\mathscr{M}$, $\mathscr{C}$, and $\mathscr{E}$, can be described as follows. The token $l_{B,q}$ represents bid limit order at relative price q from the best ask, while $l_{A,q}$ represents ask limit order at relative price q from the best bid. Using similar notation, $c_{B,q}$ and $c_{A,q}$ are tokens for cancellation of active bid and ask limit orders. All limit and cancellation orders not within Q relative prices are represented by a single token η_A for the ask side and η_B for the bid side of the book respectively. Finally, market orders arriving at the best bid and best ask are represented by μ_B and μ_A respectively. For the limit and cancellation orders, prices more than Q ticks away from the best bid and ask are not considered since trading activities that impact the market occur mostly at prices closer to the best bid and best ask [73, 72]. Given this set $\mathscr{O}$ of order event tokens, the order flow in this chapter is defined as a finite-length sequence denoted as $O_{1:T} = \{o_1, \ldots, o_T\}$, where $o_t \in \mathscr{O}$ is a token indicating the type of order event arriving at a given relative price. The sequence generation problem can now be formally described below.

As has been previously noted, generative models are models trained to estimate the joint probability $p(y,x)$ of the data, as opposed to the more commonly used discriminative models, which estimate the posterior probability $p(y|x)$ di-

rectly. The exact approach to estimating the joint probability of the data is quite varied and hence the term ”generative models” can be used to refer to a wide variety of models. To address the problem of generating simulated order flow in this thesis, the term ”generative models” refers to models that are described as follows [133]. Given a training set consisting of samples drawn from some distribution, a generative model learns to represent a distribution which estimates the given training distribution. To produce a novel sequence of order event tokens $O'_{1:T} = \{o'_1, \ldots, o'_T\}$, a generative model is trained on a set of real sequences $O_{1:T}$ such that probabilistic difference between the generated sequence and real sequences is minimised.

The deep learning models implementing recurrent neural networks (RNN) architectures in previous chapters could be used as both discriminative and generative models. RNNs essentially model the joint probability of the data by decomposing it into the conditional probabilities for each point in the sequence, given the previous steps. This is essentially a ”tractable density” [133] generative model. The training of RNNs is typically performed by by maximising the likelihood of the conditional probability in each step of the sequence [134]. However, this maximum likelihood approach suffers from exposure bias [127], which is the discrepancy between the training and inference stages. The model generates a new step of the sequence conditioned on the previous steps, but the previous steps may not have been observed in the training data since these models are usually tasked to generate novel but realistic sequences. Amongst the proposed solutions for alleviating this problem, one of the currently most interesting and promising is by using the Generative Adversarial Net (GAN) framework [126] to train the generative models. However, standard GANs are not suitable for problems involving sequences with discrete tokens, such as the order flow as noted in Section 5.2.1. This is because the generator in standard GANs needs to be able adjust the output continuously, which does not work when a sequence consists of discrete tokens. The state-of-the-art approach for addressing this problem of adversarial learning of structure sequences is the SeqGAN frame-

work introduced in Section 5.2.2.

5.3.2 SeqGAN Modelling of Order Flow Sequences

The algorithm for training the generator G_θ and discriminator D_ϕ to generate simulated order flow is described in Algorithm 1. For the work in this thesis, recurrent neural network (RNN) with LSTM cells is implemented as the generator model G_θ, with θ being the parameters of the network. As for the discriminator, D_θ is implemented by a convolutional neural network (CNN) with parameters ϕ. In both networks, the tokens $(y_1,\ldots,y_T)$ are embedded into a continuous space $(x_1,\ldots,x_T)$ using a fully connected layer. The embedding layer and RNN architectures are the same as those implemented in previous chapters, as described in Section 2.2.

In [34], the start state s_0 in the SeqGAN framework is meant to be a special token defining the start of a sequence, commonly used in natural language processing datasets. For the work in this chapter, it is proposed that the start state is a sequence of order flow. The reason for this is that it would be unnatural for an order flow to abruptly start, unlike text sentences. Therefore, to generate an order flow $O'_{1:T}$, a start sequence is defined as $O_{-T+1:0} = \{o_{-T+1,-T+2,\ldots,o_0}\}$, where the length of the start sequence is set to be the same as the length of the sequence to be generated. The reason for the notation is as follows. A given start sequence is always associated with a positive example of the order flow such that $O_{-T+1:0}$ concatenated with a positive example $O_{1:T}$ forms a continuous sequence of order flow that exists in real data. Then, when generating a simulated sequence $O'_{1:T}$, to obtain the start state, the set of start sequences can either be iterated through, or uniformly sampled from with replacement.

GANs in general are very difficult to train due to the adversarial situation where the improvements in one model will degrade the performance of the other, leading to training instability. For the work in this chapter, the following approaches were found to help the generator and discriminator to converge.

- Before adversarial training, Bayesian hyperparameter tuning [83] is implemented to determine the hyperparameters such as the number of layers or

Algorithm 1: Algorithm for training the SeqGAN generator and discriminator on the order flow.

Input:
Order flow real sequences $X = \{O_{1:T}\}_{1:N}$;
Order flow start sequences $S = \{O_{-T+1:0}\}_{1:N}$
Initialise G_θ and D_ϕ with random parameters θ and ϕ;
Pre-train G_θ using MLE on X with starting sequences S;
Generate samples using G_θ using starting state S;
Pre-train D_θ by minimising CE on generated samples as negative examples and X as positive examples;
repeat
  for *g-steps* **do**
    Uniformly sample starting sequence s from S;
    Generate $o'_1, \ldots, o'_T$ using G_θ with starting state s;
    for t *in* $1:T$ **do**
      Compute $Q(a = o'_t, s = O'_{1:t-1})$ using Eq. 5.5;
    Update θ using Eq. 5.4;
  for *d-steps* **do**
    for *each* s *in* S **do**
      Generate sequence sample using G_θ with starting state s;
      Append sequence sample to array of negative examples;
    Uniformly sample equal number of negative examples and positive examples X;
    Use bootstrapped data to train D_θ for a number of epochs given by Eq. 5.5;
until *SeqGAN converges*;

the number of hidden units for G_θ and D_ϕ, or the kernel sizes and number for the D_ϕ. Hyperparameter tuning is applied *only* during the pre-training phase. Also, when tuning the number of layers, the number of layers in the discriminator is set to be much deeper than the generator.

- During adversarial training, the following was found to work best:
  - The number for the d-step parameter in Algorithms 1 is set to be much larger than g-step.
  - For backpropogating the gradients, the Adam optimiser is used for the generator, while the Adagrad optimiser is used for the discriminator. The algorithms for both these optimisers are described in Section

2.2.3.

– For the generator, dropout is used both when training the generator itself, and when generating sequences for training the discriminator.

5.4 Benchmark Model

Since the current application of machine learning to generate simulated order flow sequences is novel, no direct comparison with existing machine learning approaches in literature can be made. However, there exist in the quantitative finance literature some very well-established stochastic process approaches for the modelling of order flow to simulate a dynamic limit order book. Among them, the Poisson process framework is the most well-established method for order flow simulation, and will be used in this chapter as the benchmark for the deep generative modelling approach for generating the order flow. Though there exists other more complex models such as the queue-reactive model [135], or the Hawkes process framework [10], the simplicity in fitting stable models makes the Poisson process framework more desirable as a reliable benchmark.

The Poisson framework used as the benchmark is the multiple Poisson process described in Chapter 2.3.2. Though Hawkes processes could better simulate the jumps found in real mid-price time-series, as mentioned in Chapter 2.3.1, the kernel of the Hawkes process tend to be difficult to fit, leading to computation issues. In the multiple Poisson model, one process is added for each of the order event token that was described in Section 5.3.1. After the arrival rate parameters for the processes are fitted, the procedure for generating a sequence of order event tokens is quite straightforward. For each order event token representing an event type at a given price, the arrival times of the token can obtained by simply sampling from the process representing the token. Then, all of the generated token sequences are concatenated into a single data-structure and sorted by time to obtain the generated order flow.

5.5 Dataset

The experiments again make use of BTC-USD data from captured from Coinbase just as described in Section 3.4. As previously described, these are raw JSON messages that needs to be transformed into a dataset suitable for machine learning using an implemented limit order book and matching engine. Unlike in Chapter 3 and Chapter 4 however, no capture of the target variables are required since here the sequence of order flow its self is used for training and testing.

The order flow between 4 Nov and 29 Nov is used for training. In this period, the order flow is partitioned into slices of 400 events. Each of the slices is split equally into two to obtain the real order flow sequence $\hat{O}_{1:T}$ and the start sequence $\hat{O}_{-T+1:0}$. All of the real order flow sequences are concatenated into a single dataset for training the discriminator D_ϕ, while the start sequences are concatenated into a dataset to be used for generating a sequence of order flow in the generator G_θ

For sequence similarity testing, the order flow between 30 Nov and 1 Dec is used. Within this period, the real order flow in the test period is split into overlapping slices of 400 events. Unlike in the case of the training data, here the last 200 events of a given slice overlap with the first 200 events of the next slice. In other words, the first sequence of events $o_{1:200}$ is used as the start sequence to generate a simulated order flow $o'_{1:200}$, and the remaining sequence of orders $o_{201:400}$ is used for micro-behaviour analysis of $o'_{1:200}$. Then, the sequence is used as the start sequence for generating $o'_{201:400}$, and $o_{401:600}$ is used for analysing $o'_{201:400}$, and so on.

The period between 30 Nov and 1 Dec is also used for investigating the macro-behaviour of the simulated order flow. The dataset for the investigation is obtained by capturing the mid-price at every 1-minute interval when the raw JSON messages were used to build the order flow sequence dataset using the implemented limit order book and matching engine.

5.6 Sequence Similarity

In this section, the ability of the SeqGAN model and the benchmark model to continue a given start sequence is tested. Each start sequence in real data would have an ensuing sequence, and a perfect model should be able to repeatedly generate this ensuing sequence with perfect similarity to real data. The sequence similarity test set described in Section 5.5 will be used for this experiment. Each point in the test set contains a sequence 400 events long. The first half of the sequence is used as the start sequence to generate a sequence o'. The second half for is used to evaluate o'. Note that the main objective of the work in this chapter is to learn a model that is able to generate simulated order flow sequences, and not to reproduce any given order flow sequence perfectly. Nevertheless, the test described in this section is important in evaluating the ability of each model in generating correct sequences of the order flow.

Sequences similarity metrics are employed here to measure how well the SeqGAN model and the benchmark model are able to realistic order flow. In literature, there exist a large number of algorithms which computes the similarity between multiple sequences, the choice of which depends largely on the application. Two sequence similarity measures are identified as suitable for evaluating the output of the deep generative model, and the benchmark model. These measures are the Levenshtein distance [136] and the Jaccard index [137]:

- The Levenshtein distance between two discrete sequences is computed using an algorithm which counts the minimum number of single element edits that transform one sequence to the other. The edits allowed are insertions, deletions and substitutions. The Levenshtein distance can be normalised such that the similarity score is bounded between 0 and 1, where 1 means both sequences are a perfect match and 0 means the sequences are as dissimilar as can be.

- The Jaccard index for two discrete sequences computes the ratio between the number of common single tokens between both sequences, and the

total number of unique single tokens in both sequences. Thus, the Jaccard index is also bounded between 0 and 1.

For evaluating the simulated order flow produced by the deep generative model and the benchmark model, the features of the Levenhstein distance and the Jaccard index complement one another. The Levenshtein distance provides a more granular view of how each element and its neighbours in the generated sequence are similar to the real sequence, while, the Jaccard index provides an overall view of how well the set of order events in the generated sequence matches up with the real sequence.

For the deep generative model, the generator G_θ is trained on the dataset in the training period as described in Section 5.5. Then it is given the start sequences of 200 event length from the test set to generate the simulated order flow sequences. On the other hand, the benchmark model is always trained from the last 600 order events when simulating for each point in the test set. It was found that this is much better for the benchmark model than using the whole training period parameter fitting. The multiple Poisson model used in the benchmark is small model consisting of only one parameter for each process, and a large data training set (such as that in the training period) would have saturating effect on the models very quickly. Also, by using the latest 600 events in the order flow, the parameters of the benchmark model would be fitted to the latest data. This helps mitigates the weakness of the benchmark model in regime shifting situations, and thus strengthening the standards against which the deep generative model is compared.

The sequence similarity between each model and the order flow in the test period is then evaluated as follows. For each datapoint in the test set, the models will generate 100 simulated sequences each. These multiple samples are necessary for evaluation rather than generating just one sequence due to the randomness in the generative models. Then, for each of the 100 simulated sequences, the Levenshtein distance and Jaccard index are computed between the simulated sequence and the real sequence.

5.6.1 Results

Table 5.1 shows the overall mean of both similarity measures, for both models, taken across all the simulated sequences, and across the whole of the test set. It can be observe that overall the ability of either model in producing realistic order flow sequences is weak. The Jaccard indices for both models are much larger than their Levenshtein distances. This implies that the simulated order flow generated by both models contains at least somewhat-correct assortment, even-though the arrangement of the events in the sequences themselves are not as good.

	SeqGAN Model	Benchmark Model
Levenshtein Distance	0.04	0.01
Jaccard Index	0.19	0.06

Table 5.1: Average Levenshtein distance and Jaccard index across multiple samples of simulated order flow, and across the whole test set, rounded to two decimal places.

But, the results observed in Table 5.1 are not unexpected since there are a very large number of price levels, with Q set to 3000 based on the median of the relative prices. Financial market datasets are in general very hard to predict, let alone at the order flow level with so many different price levels. The more important question to ask here is if the deep learning approach statistically significantly improves on the benchmark.

To this end, the distribution of the average distances between simulated and actuals, for each point in the test set, is computed for each model. The distribution for the SeqGAN model here are denoted just for now as A, and the distribution for the benchmark model as B. Then both distributions are tested against each other using the one-tailed Welch t-test under the null hypothesis $H0 : B \geq A$. The resulting p-value of very close to 0 implies that the SeqGAN model indeed significantly outperforms the benchmark model in the test of sequence similarity to actuals.

Next, it is interesting to loosen the requirements and test if both the SeqGAN model and benchmark models are at least generating the correct order

type and order direction. For this purpose, all tokens in both the simulated and real sequences are binned such that the price elements are stripped from the token. For instance, the tokens for buy limit orders at each price q are binned to a single token for just buy limit order. Such sequences could still be very useful since order type and order direction are important for anticipating any price movements. Readers are reminded that in Chapter 3, the order type and order direction were found to contain strong predictive power in forecasting the directional price movements. Thus, the experiment is repeated here with the binned sequences, and the results are presented in Table 5.2. The Jaccard index is now perfect, but it should be noted that the reason for this is that the cardinality of the binned tokens is now quite small ($= 7$). Since the simulated sequences are quite long ($= 200$), it is not difficult for a given simulated sequence to contain all the binned tokens. The Levenshtein distance, on the other hand, is also shown to have massively improved, and this is more meaningful. It implies that the order type and direction of the events in the simulated sequences have some semblance to real data, albeit at incorrect price levels.

	SeqGAN Model	Benchmark Model
Levenshtein Distance	0.39	0.16
Jaccard Index	1.00	1.00

Table 5.2: Average Levenshtein distance and Jaccard index across multiple samples of simulated order flow, and across the whole test set, after binning rounded to two decimal places.

This section is concluded by noting that the SeqGAN model is shown here in Table 5.2 to yet again have statistically significantly better performance than the benchmark model. The Welch-test with the same hypothesis as before is applied to this results in this experiment, with the null hypothesis of $H0 : B \geq A$ being rejected with very high confidence.

5.7 Macro-Behaviour Analyses

In real markets, the flow of orders into the limit order book results in a random fluctuation of the mid-price. Take for instance an order flow sequence that is

generated by either model. If the simulated sequence is a realistic reproduction of real order flow sequences, the resulting mid-price time-series should have comparable statistical properties to the real mid-price series in the same period. This section presents the investigation of these statistical properties and stylised fact in the mid-price time series produced from the simulated order flow sequences, and the mid-price time-series from the real data.

Recall that both the deep learning and benchmark model produce only the tokenised form of the order flow. To simulate the mid-price time series from the simulated order flow, two additional components need to be estimated for the SeqGAN model, and one for the benchmark model. For both models, the order size of each order event needs to be estimated, and for just the SeqGAN model, the inter-arrival times between each event in the sequence need to be estimated as well. The order size for each simulated event can obtained by sampling from empirical distributions. The order size empirical distributions V_μ, V_l, V_c for market, limit and cancellation orders respectively are fitted from the training data in the period between 23 Nov 2017 and 1 Dec 2017. Similarly, the inter-arrival times are obtained by sampling from an empirical distribution, though with only one distribution U being fitted for all order types. The inter-arrival time is not needed for the benchmark model as each Poisson process directly models the arrival rate of the associated order event.

One thing to note here is the treatment of cancellation order events in the implementation. In reality, each arriving cancellation order is paired to an active limit order in the LOB, since it is that paired limit order that is being cancelled. In both the SeqGAN model and the benchmark, arriving cancellation orders at price q are simply events stating that the volume at price q should be removed, and are not paired to any active limit order due to the unnecessary complexity in modelling such relationships. In this case, if there are active limit orders resting at price q, they are removed. Otherwise, nothing happens. Also, since cancellation orders more than Q relative prices away from the best bid and best ask are not explicitly modelled, when the mid-price moves, any active limit orders

with relative prices larger than Q are automatically cancelled. Correspondingly, any token in the simulated order flow which represents limit orders with price larger than Q are discarded.

Algorithm 2: Algorithm for sampling a single 1-minute mid-price time series from SeqGAN model.

Input:
Real order flow start sequence $O_{1:200}$;
Generator model G_θ;
Empirical distributions V_μ, V_l, V_c, U;
Warmed-up limit order book object L;
Total time in seconds T;
Output: 1-minute mid-price time series S

Initialise empty list P;
Initialise empty list O';
Initialised total time in milliseconds $t = 0$;
Initialise start sequence $r = O_{1:200}$;
repeat
  Initialise empty list o';
  for i *in* $\{1:200\}$ **do**
    Sample event token o'_i from G_θ using r and o'_{i-1};
    Sample order sizes v_i from either V_μ, V_l or V_c depending on order type of o'_i;
    Sample inter-arrival times τ_i from U;
    Append (o'_i, v_i, τ_i) to o'
  Extend list O' with o';
  $t \leftarrow t + \sum_i \tau_i$;
  $r \leftarrow o'$;
until *until* $t = T$;
Initialised total time in milliseconds $t = 0$;
for (o', v, τ) *in* O' **do**
  Input o', v into order book L;
  $t \leftarrow t + \tau$;
  if $t \geq 60000$ **then**
    Capture mid-price s;
    Append s to S;
    $t \leftarrow 0$

The approach for sampling a regular interval mid-price time series from the SeqGAN model is then described in Algorithm 2. For the work in this section, an interval of 1 minute is set for the mid-price time-series, with the length of the

mid-price series totalling 48 hours. An interval of 1-minute sets a good balance between being large enough for statistical tests, and small enough to be relevant to the price formation process, which is the primary objective of this thesis. Using a start sequence of real order flow with 200 events, the generator G_θ is used to generate a simulated sequence of 200 events. Then, this first sequence is in turn used to generate further simulated sequences. This approach is taken instead of having G_θ continually generating order flow events in a single sequence because the length of the order flow sequence needed to produce the mid-price time-series can get very large. The average inter-arrival time between orders is 13 milliseconds, meaning about 277200 order events need to be generated to produce a time-series length of just 1 hour. And recall here that, a 48 hour time-series is needed here to cover the period in the test set. Although not described in Algorithm 2, a memory management procedure is implemented to efficiently handle a data structure of that size.

Another thing to note in Algorithm 2 is that the limit order book object L needed for processing the order flow sequence into the mid-price time-series has to be ”warmed up”. The warming up procedure feeds the limit order book object with a sufficiently long sequence of real order flow to ensure the book is properly populated at the right price levels.

Using Algorithm 2, 100 random simulation paths of the order flow are then generated from the SeqGAN model to produce 1 minute interval mid-price time series for a period of 48 hours. The same number of samples of the order flow is produced from the benchmark model for a period of 48 hours. A number of investigations are then performed on the simulated mid-price and the real mid-price series to allow for comparison between the deep learning and benchmark models. The aim of these analyses is to discover how well the emerging properties of the order flow from each of these models matches the statistical properties of the real mid-price time-series in the test set. The rest of the section will therefore present an analysis of the mid-price log-returns distribution, and the mid-price volatility, between the mid-price simulations produced by each of the

SeqGAN model and benchmark models, and the real mid-price from the test set. Both the log-returns distribution and volatility are important statistical properties of a financial time-series, and close re-production of these macro-behaviours is important to ensure the simulated order flow sequences are realistic.

5.7.1 Macro-Behaviour 1: Mid-Price Returns Distribution

In the first investigation the log-returns from the simulated samples are compared to the log-returns of real mid-price time series. The aim of this investigation is to discover of how realistic are the 1 minute interval mid-price returns produced from the order flow sequences generated by each model. To compare the empirical log-returns distributions of a simulated mid-price series, to the real mid-price series, the two-sample Kolmogorov-Smirnov (K-S) test [138] is employed. Denoting the dataset of log-returns computed from one of the simulated time series as A, and the dataset of log-returns from the real mid-price series as B. The K-S test is then performed under the null hypothesis that datasets A and B are sampled from the same distribution. Since 100 simulated order flow sequences were obtained for each of the SeqGAN model and the benchmark model, the K-S test has to be performed 100 times for each model.

However, when performing these tests, the issue of multiple comparison becomes relevant since the more simulated mid-prices are generated, the more likely one would pass the K-S test. To avoid this, a control procedure needs to be applied to the family-wise error rate of the tests. Specifically, the Hochberg's step-up procedure [139] is implemented as an additonal step to control the outcome of the multiple K-S tests. The procedure sorts the hypotheses of the 100 K-S test performed by p-value, and determines which of these hypotheses with the lowest p-value should be rejected for the SeqGAN model and the benchmark model respectively. For the tests performed in this section, a larger than usual significance level of 0.1 is chosen since simulating noisy financial time-series is quite an immense challenge. Between the SeqGAN model and the benchmark, the one with least number of hypotheses rejected by the Hochberg's step-up procedure is then the model that is the more likely to produce an order flow with

realistic macro-behaviour.

Time-Series Length	SeqGAN Model	Benchmark Model
1 Hour	**73**	86
6 Hours	**88**	91
48 Hours	**98**	100

Table 5.3: Number of Kolmogorov-Smirnov test hypotheses (out of 100 samples each) that are rejected in the Hochberg's step-up procedure. The time-series length column refers to the first 1, 6 and 48 hours for each of the 100 samples.

Table 5.3 below shows the number of hypotheses rejected for the SeqGAN model and benchmark model, with the experiments replicated for the first 1 hour, 6 hours and 48 hours of the mid-price time-series. It can be observed from the table that as the time-series length is increased, the similarity between the log-return distributions of the simulated order flow and the real order deteriorates, as expected. For the longer time-series, quite a large number of the samples are rejected for both models, but this is to be expected as high-frequency financial time-series are extremely challenging to realistically replicate, especially for long time periods.

Recall that the simulated order flow for the mid-price time series are produced iteratively where, initially, a new simulated sequence is generated from a starting sequence of real order flow. Then this generated sequence is fedback as a starting sequence to generate another new sequence, and so on. The performance for time-series of different lengths in Table 5.3 would suggest that as each new sequence is generated, conditioned on a previously generated sequence, the resulting statistical behaviour of the mid-price log-returns starts to deviate from the actuals. Although Generative Adversarial Networks in theory mitigate this exposure bias problem, it seems as if for this experiment the problem even so persists in the long run.

Nonetheless, Table 5.3 shows the simulated order flow produced by the SeqGAN model is better at reproducing the mid-price log-returns of real data compared to the multiple Poisson benchmark model. This observation applies to all three time-series lengths in the experiment.

5.7.2 Macro-Behaviour 2: Mid-Price Returns Tail-Exponent

Next, the tails of the absolute log-returns distributions for the simulated mid-price of each of the models are compared to those of the real data. Empirical studies have reported strong evidence of power law behaviour [140, 141, 142] in the absolute log-return distributions of financial time series. Power law probability distributions are "heavy-tailed", meaning the right tails of the distributions still contain a great deal of probability. This implies that financial time series tend to have a higher probability of extreme returns events [143], which may be for better or worse depending on whether the return is positive or negative. Power law distributions are probability distributions with the form $p(x) \propto x^{-\alpha}$, and it is the tail-exponent α that is the subject of this analysis in this section.

The Jarque-Bera test [144] is employed on the real mid-price series, for the first 1 hour, 6 hours and 48 hours, to determine if there are heavy tails in the absolute log-returns distribution. From Table 5.4, it can be observed that the kurtosis of the test distributions is much larger than 3, indicating heavy tails, and the Jarque-Bera null hypotheses of Normal-distribution kurtosis and skewness are rejected with very high significance. Therefore, it appears the real mid-price series in the test period shows heavy-tail behaviour in its absolute log-returns. The computed tail-exponents are also presented in Table 5.4. These tail-exponents are on the lighter end of what is observed from the literature, where empirical studies of stock returns present strong evidence of exponents between 2 and 3 [145, 146, 147]. There is some evidence in the literature pointing to lighter tails as well, however. The work in [141] argued for a lighter tail, between that of an exponential distribution and a power law. Also, in [148] the authors reported that returns at 1-minute time scales exhibit tail-exponents larger than 3, which correlates with the findings in Table 5.4.

Next, an investigation is performed to determine if the tails observed in the simulated mid-price series replicate those of the real time-series. The procedure is as follows. First, the absolute log-returns of the simulated mid-price series are tested for heavy-tail behaviour. Table 5.5 shows the aggregated results of the

Time-Series Length	Tail-Exponent	Kurtosis	Jarque-Bera p-value
1 Hour	3.67	8.79	0.00
6 Hour	2.98	8.46	0.00
48 Hour	3.30	10.98	0.00

Table 5.4: The kurtosis and p-values from the Jarque-Bera test, and the computed tail-exponents, for the real mid-price time-series absolute log-returns in the test period. The time-series length column refers to the first 1, 6 and 48 hours of the real mid-price time series.

Jarque-Bera test across the 100 samples generated by the SeqGAN model and benchmark model respectively. The measured kurtoses are averaged, and the Hochberg Step-Up procedure described in Section 5.7.1 is applied to determine the proportion of tests to be accepted at a 1% significance, after controlling for repetition bias. It can be observed in Table 5.5 that the average kurtosis is much larger than 3, and the null hypotheses of the Jarque-Bera test across all samples are rejected. The results strongly indicates that the simulated mid-price series for both the SeqGAN model and the benchmark model do replicate the heavy-tail behaviour reported in the literature real financial time-series.

Length	Metric	SeqGAN Model	Benchmark Model
1 Hour	Mean Kurtosis	7.31	6.97
	Rejection Count	0	0
6 Hours	Mean Kurtosis	7.49	7.19
	Rejection Count	0	0
48 Hours	Mean Kurtosis	8.80	7.53
	Rejection Count	0	0

Table 5.5: The mean kurtosis from the Jarque-Bera test, and the number of Jarque-Bera tests rejected by the Hochberg Step-Up proceduure, across the 100 mid-price time-series samples generated by the SeqGAN model and benchmark model respectively. The length column refers to the first 1, 6 and 48 hours for each of the 100 samples.

Next, the absolute log-returns tail-exponents of the simulated mid-price series produced by the two models are compared to the results in Table 5.4. The distribution of tail-exponents are computed across the sampled mid-price time-series of each model. Then, the one-sample two-tailed Student t-test is employed with the null hypothesis $H0 : A = B$, where A is the mean of a tail-

exponent distribution, and B is the tail-exponent in the real time-series data given in Table 5.4. Table 5.6 shows the resulting p-value and t-statistics. The null hypotheses of the tests for both models are rejected with high confidence. This implies that neither of the models has produced mid-price time-series with realistic tail-exponents in the absolute log-returns. It can also be observed from the t-statistics that the distribution tails of both models are lighter than those of the real returns distribution. However, the t-statistics of the SeqGAN model are observed to be much much smaller that those of the benchmark model. This indicates that the SeqGAN model generate order flow sequences that produces mid-prices with closer tail-exponents to those of the real data.

Length	Metric	SeqGAN Model	Benchmark Model
1 Hour	p-value	0.00	0.00
	t-statistic	**2.66**	3.29
6 Hours	p-value	0.00	0.00
	t-statistic	**2.84**	4.05
48 Hours	p-value	0.00	0.00
	t-statistic	**2.23**	3.71

Table 5.6: Results of one-sample two-tailed Student t-test for the tail-exponent distributions of each model against the real tail-exponents computed in the test period, rounded to two decimal places. The time-series length column refers to the first 1, 6 and 48 hours for each of the 100 samples.

5.7.3 Macro-Behaviour 3: Mid-Price Volatility

Next, the volatility in the mid-price produced by the two models is compared to the volatility of the real mid-price in the test set. Volatility is one of the most important measures of an asset's value [149]. It measures the risk that would be undertaken when trading the security and is crucial in the construction of optimal portfolios [150]. There are a number of measures of volatility defined in the literature [151], and the choice depends on the purpose. For high-frequency price movements, the following volatility measures are found to have significant importance [152, 153]:

- Given a time-series of absolute log-returns $r = \{r_{t_1}, \ldots, r_{t_k}\}$, its *realised volatility* is the standard deviation of the values in the series, which can be

computed as follows:

$$v_r = \sqrt{\frac{\sum(r_i - \mu)^2}{k}}, \tag{5.12}$$

where μ is the mean of the values in the series r.

- Let $r = \{r_{t_1}, \ldots, r_{t_k}\}$ be a time-series of absolute log-returns, where t_i is the arrival of the ith market order between t_1 and t_k. Then, its *realised volatility per trade* is the standard deviation of the values in the series, which can be computed as follows:

$$v_p = \sqrt{\frac{\sum(r_i - \mu)^2}{k}}, \tag{5.13}$$

where μ is the mean of the values in the series r.

- Given a price series $s = \{s_{t_1}, \ldots, s_{t_k}\}$, its *intraday volatility* is computed by:

$$v_d = \log\left(\frac{\max(s)}{\min(s)}\right), \tag{5.14}$$

Time-Series Length	v_r	v_p	v_d
1 Hour	0.00177	0.00149	0.0308
6 Hours	0.00186	0.00153	0.099
48 Hours	0.00257	0.00211	0.178

Table 5.7: The different volatility measures computed from the real mid-price time-series in the test period. v_r refers to the realised volatility, v_p refers to the realised volatility per trade, and v_d refers to the intraday volatility. The time-series length column refers to the first 1, 6 and 48 hours for each of the 100 samples.

Table 5.7 shows each of the volatility measures computed from the real mid-price in the test period. Then, the comparison of the mid-price volatility behaviour between the SeqGAN model and the benchmark model is performed as follows. First, the empirical distributions of the volatility measures are computed across the simulated mid-price series produced by each of the models. Then, a one-sample two-tailed Student t-test is employed with the null hypothesis $H0 : A = B$, where A is the volatility of the empirical distribution, and B is

the volatility computed from the mid-price in the test period, as shown in Table 5.7. The alternative hypothesis is $H1 : A \neq B$. The results from the tests are presented in Table 5.8, where it can be observed that the null hypotheses for all the tests are rejected with high confidence. This implies that the order flow generated by both models is not able to reproduce the volatility of the real mid-price time series. The negative t-statistics imply that the simulated mid-price time-series actually have much lower volatility compared to the real time-series. Comparing the SeqGAN model and the benchmark model, it can be observed from the t-statistics in Table 5.8 that the volatility of the mid-price is in general better replicated by the SeqGAN model. An exception to this would be the intra-day volatility for time-series lengths 6 hours and 48 hours, which the t-statistics show it was reproduced much more closely by the benchmark model than the SeqGAN model.

Length	Volatility	SeqGAN Model		Benchmark Model	
		t-statistics	p-value	t-statistics	p-value
1 Hour	v_r	**-0.92**	0.00	-0.99	0.00
	v_p	**-0.99**	0.00	-1.10	0.00
	v_d	**-0.89**	0.00	-0.93	0.00
6 Hours	v_r	**-1.04**	0.00	-1.13	0.00
	v_p	**-0.99**	0.00	-1.19	0.00
	v_d	-1.11	0.00	**-0.95**	0.00
48 Hours	v_r	**-1.32**	0.00	-1.46	0.00
	v_p	**-1.27**	0.00	-1.43	0.00
	v_d	-1.18	0.00	**-1.03**	0.00

Table 5.8: The p-value and t-statistics of the one-sample two tailed Student t-test between different volatility distributions of the SeqGAN model and the benchmark model, against the real volatility measures, rounded to two decimal places. v_r refers to the realised volatility, v_p refers to the realised volatility per trade, and v_d refers to the intraday volatility. The length column refers to the first 1, 6 and 48 hours for each of the 100 samples.

5.8 Summary

In this chapter, a novel application of deep generative adversarial networks (GANs) for generating simulated order flow sequences was proposed. First, the problem of generating order flow sequences using Sequence GANs (SeqGAN)

was formulated. The method for training the SeqGAN generator and discriminator for modelling the order flow sequences was described. And, a suitable model was also selected from the literature to act as a benchmark against the SeqGAN model. Then in Section 5.6, an experiment was performed to evaluate, using sequence similarity distance measures, the ability of the SeqGAN model to reproduce the real order flow sequences in the test period. It was shown from the resulting Levenshtein Distance and Jaccard Index that the SeqGAN model is able to better reproduce the real sequences in the test sets than the benchmark. Then, in Section 5.7 an investigation into the macro-behaviour of the mid-price movements created by the generated order flow was performed by measuring how well the statistical properties of the simulated mid-price series replicate those of the real mid-price series in the test period. It was shown that the mid-price time-series from the SeqGAN model is statistically significantly better able to replicate the overall returns distribution, the returns distribution tail and the volatility of the real mid-price time-series, compared to the benchmark model.

However, the results revealed that there is still much work to be done in terms of improving the approach for the generative model of the order flow. The objective of this chapter is quite ambitious as financial in general sequences are hard to predict, let alone simulate. Although the SeqGAN model is shown to outperform the benchmark, the current level of performance is not yet ideal for adopting the model in practice. From Section 5.6, the average Levenshtein distance of between the sequences generated by the SeqGAN model, and the real sequences, illustrates that the SeqGAN model is not likely to produce an order flow with events arriving in the right order. Although, the Jaccard index between the sequence generated by the SeqGAN model, and the real sequences, indicates that the set of events in the generated order flow corresponds quite well to real order flow sequences. This positive results suggests that it could be applied for computing order intensity measures. Additionally, when the events in the order flow were binned by collapsing all prices into order type and direction,

and the sequence similarity experiments were repeated, the SeqGAN model showed better performance. This suggests that the generative learning of the order flow sequences were at least partly achieved, but not the right price levels. Finally, in the analysis of the macro-behaviours of the generated order flow in Section 5.7, showed that the mid-price series resulting from the SeqGAN model generated order flow is much less volatile on average, and have much lighter tails in the log-returns distribution than the real data. On a positive note, the heavy tails in the absolute log-returns of the real mid-price, which is an important statistical behaviour of the mid-price time-series, is successfully reproduced by the SeqGAN model.

Chapter 6

Conclusion

This chapter concludes the work in this thesis. First, a summary and discussion of the main contributions of this thesis are presented. Then, future work and possible extensions to the work of this thesis are proposed.

6.1 Discussion and Summary of Contributions

The primary research objective of this thesis was to use order flow data in deep learning models to improve price prediction models, and to provide a data-driven approach to learning the price formation process. Here, the key contributions of the thesis towards this objectives are summarised. Though dataset from a single exchange, Coinbase, is used to draw conclusions leading to the key contributions in this thesis, they remain valid and not limited to only cryptocurrency assets. Coinbase is the second largest cryptocurrency exchange globally and is therefore a very liquid market in terms of the order flow behaviour. Results extracted from this data source should therefore generalise to other liquid markets such as other foreign exchange and stock markets. Also, the results in this thesis are relative and anchored on some baseline models, as will be recapitulated in the following summaries.

The work in Chapter 3 demonstrated why deep learning models should utilise order flow data for learning price prediction models, as opposed to previous work in which deep price prediction models are trained from data that are derived from the order flow, such as the order imbalance and the price levels in

the order book. For this purpose, a novel approach to training a deep recurrent neural network (RNN) using the order flow was introduced, with the objective of predicting the directional mid-price movements. Models from previous work were used to benchmark the deep order flow model and illustrate the improvements to prediction performance. The results show the deep order flow model to be statistically significantly outperforming the benchmark model in terms of its ability to predict the directional movement of the mid-price. Further analysis of the results revealed that the deep order flow model has additionally managed to learn a stationary price formation model: the experiments make use of a large test period, which overlaps with a market crash event, and subsequent volatile trading period; even so, the deep order flow model maintains its ability to maintain the mid-price, while the performance of the benchmark models degrade throughout the test period. Due to the encouraging performance of the deep order flow model, a black-box interpretation algorithm was used to investigate what features of the order flow had contributed to the learning of the price formation process. The results of the investigation are interesting as they both support and conflict with existing research on the price formation process. Some factors that were reported to have significant impact on price formation in the literature, such as such as inter-arrival times and order size, were revealed to not be a huge factor here for learning the directional movements of the mid-price. Other factors, however, agree with the findings in previous work. The overall result of the investigation of this chapter illustrates the potential of data-driven models, even black-box models, in studying the market microstructure.

In Chapter 4, the focus turned to improving the price modelling capability of the deep RNN model that was implemented in Chapter 3, by introducing an extension to the model architecture. This extended architecture combines deep neural networks and mixture modelling, with the objective of producing probabilistic predictions of price movement, and jointly modelling the direction and size of the movements. A deep learning model that produces such output for predicting high-frequency price movements is entirely novel, and therefore

suitable models from other areas of research had to be chosen to provide a benchmark. Results show that the deep mixture model outperforms the benchmark models in its ability to reduce the directional risk and size risk of the price predictions. Similarly to in Chapter 3, the large test period in the dataset used here overlaps with a market crash event. Here, once again, the deep mixture model is shown to be able to significantly maintain its predictive power much better than the benchmark models. A further experiment was conducted which showed that the joint modelling of the price movements size and direction in the deep mixture model is an improvement over a more straightforward approach that uses two separate deep neural network, each predicting the probabilistic size and direction of the price movements respectively. Finally, a simulated trading scenario, that uses real order flow data taken from the the test period of the dataset, is implemented, which further illustrates the potential of deep mixture model. The outputs of the models are integrated into separate trading algorithms that implements the same trading strategy. Multiple runs of the trading scenarios were then performed, demonstrating that the deep mixture model was statistically significantly more profitable than the benchmark models.

Chapter 5 then further improves on the deep learning approach to modelling price formation by introducing deep generative adversarial networks (GAN) for modelling the order flow sequences themselves. Similarly to work with zero-intelligence models described in Section 2.3.1, price movements can then be obtained from the generated order flow sequences in a bottom-up way. Currently, there exists no previous work that uses deep learning models to model the order flow directly, and study the resulting price formation. Therefore, to benchmark the performance of the SeqGAN model that was introduced in this chapter, a well-known stochastic process model of the order flow from the quantitative finance literature was selected as a suitable benchmark. Then, the sequence similarity between real order flow sequences and the output of the models was investigated. When a starting order flow sequence was given to the SeqGAN model, it was shown to be able to produce sequences that are statistically sig-

nificantly better than the sequences produced by the benchmark models in two ways. First, the arrangement of order events in the sequences generated by the SeqGAN model was shown to be much closer to the real sequences than the benchmark. Then, the set of order events in the SeqGAN generated sequences was shown to overlap with the set of events in real data much more than the benchmark sequences. Finally, an analysis of the macro-behaviour of the order flow was performed to study the mid-prices that are dynamically formed by the order flow. Statistical testing revealed that the distributions of the mid-price log-returns for the SeqGAN model was closer to real mid-price time-series, than in the case of the benchmark. In addition, the important behaviour of heavy-tails was replicated in the mid-price log-returns distribution of the SeqGAN model, and the computed tail-exponent of the SeqGAN mid-price log-returns distribution was found to be closer to real data than was the true for the benchmark. The volatility of the model generated mid-price was also compared to the volatility of the real mid-price time series. The advantage of the SeqGAN model is further reinforced here as different volatility measures of the real mid-price time-series were better replicated in the order flow generated by the SeqGAN model, compared to the benchmark.

6.2 Future Work

Due to the novelty of the research area, and the promising results from the work described in the thesis, many possible extensions to the work can be suggested. The more straightforward, though less exciting, continuation of the work of this thesis would of course be direct improvements to the deep learning models. More suitable model architectures, better neural network layer formulation, improved loss functions and learning algorithms could be proposed through deep dive analyses of the model predictions. The generative model in Chapter 5 in particular, though it outperformed the benchmark, could benefit from such an analysis and the resulting improvement to the model, generating better order flow sequences. Other than these kinds of direct model improvements, select

extensions that are potentially more interesting for future work are proposed as follows.

6.2.1 Universality Property of the Order Flow Models

In Section 3.6.2, a brief analysis of the possible universality property of the deep order flow model is presented with a caveat: the datasets used in the analysis are extracted from currencies that are known to move together, which detracts the observation that the price formation model learned from an order flow dataset can be transferred across to other order flow datasets. Such universality properties have been reported for deep learning models of stock time-series across different symbols [33]. Therefore, it would be interesting to investigate if the universality observed in Section 3.6.2 would persist when the model is transferred to datasets extracted from other currencies or financial securities. If the transferred model were not immediately usable on the new dataset, then there would also be the valid question of how close it is to being usable, by measuring how much effort it would take for the transferred model to converge, compared to a new model trained entirely on the new dataset. Such work could reveal if deep learning models are able to partly or fully learn universal models of the price formation process in complex financial markets from order flow data.

6.2.2 Representation Learning of Order Flow Features

One of the core concepts of deep learning is that abstract representations of given input data are learnt in each layer of the hierarchical deep neural network architecture. The abstract representation that is learnt in each layer is the transformation of the input of the layer into features that make it easier for the model to learn the predictive task at hand. Due to the promising results of the deep order flow model, it is of interest to investigate how powerful is the representation that has been learnt by the model. In application domains such as image classification, such representations can be visualised [154] and inspected by manually by eye. However, such visualisation is currently not possible for econometric data such as the order flow. An experiment can however be conducted by making

use of the output of the last recurrent layer as features for training other types of models, such as random forest. The performance of other models trained on these features can be compared against training with the order flow data directly, to investigate the power of the learnt representation.

6.2.3 Generative Modelling of Full Order Flow

In Chapter 5, the order flow had to be tokenised into price-direction events so that could can be used with the introduced model. When studying the price formation, the inter-arrival time between the orders, and the size of the order, had to be sampled from empirical distributions that were fitted on data. Given this, it would be of interest to develop a framework that would be able to learn to jointly model all the features of the order flow for generating the full order flow. It would be of interest to test whether such an approach, with its possible added complexities, would be better than the current disjointed approach. The vanilla RNN might be able to achieve this, but the effects of exposure bias could be a major disadvantage when generating longer sequences. Certain kinds of GANs could potentially solve this problem, such as that described in [128]. However, the full order flow sequence contains both categorical tokens and continuous variables and the handling of tokens in the adversarial training is still an open question in these models.

6.2.4 Generating Synthetic Data for Training Models

One of the interesting uses proposed for the SeqGAN model in Chapter 5, though not directly related to learning the price formation, is to generate synthetic data for training models. Order flow data is notoriously expensive, and even for the relatively small dataset used for the experiments in this thesis, it had taken significant resources in terms of monetary costs and time. Synthetic data generated by generative models such as the SeqGAN could democratise research by providing order flow data that could otherwise be unobtainable to many researchers. If time had permitted, it would have been interesting to revisit the experiments in Chapter 3 and Chapter 4 with the synthetic data generated

by the SeqGAN model. Although the performance of the SeqGAN model is yet ideal, it would nonetheless have been interesting to perform such an experiment, comparing the synthetic data against real data, in training a model to predict price movements in the test period.

6.2.5 Concluding Remarks

The work in this thesis have opened up new avenues for research by being the first application of deep learning to order flow data for modelling and studying the price formation process. Although promising results have been demonstrated in this thesis, the modelling of complex financial markets remain however, a highly challenging research area. There is much potential to build on the presented work, as has been discussed in the preceding section on Future Work. It is hoped that the research community in computational finance will be able to make productive use of this new avenue in the deep learning of price formation models for further research, and equally as important, for profit.

Bibliography

[1] Melanie Mitchell. Complex systems: Network thinking. *Artificial Intelligence*, 170(18):1194–1212, 2006.

[2] Marcus G Daniels, J Doyne Farmer, László Gillemot, Giulia Iori, and Eric Smith. Quantitative model of price diffusion and market friction based on trading as a mechanistic random process. *Physical review letters*, 90(10):108102, 2003.

[3] Eric Smith, J Doyne Farmer, L szl Gillemot, Supriya Krishnamurthy, et al. Statistical theory of the continuous double auction. *Quantitative finance*, 3(6):481–514, 2003.

[4] J Doyne Farmer, Paolo Patelli, and Ilija I Zovko. The predictive power of zero intelligence in financial markets. *Proceedings of the National Academy of Sciences*, 102(6):2254–2259, 2005.

[5] Linqiao Zhao. A model of limit-order book dynamics and a consistent estimation procedure, 2010.

[6] Ioane Muni Toke. "market making" in an order book model and its impact on the spread. In *Econophysics of order-driven markets*, pages 49–64. Springer, 2011.

[7] Emmanuel Bacry, Khalil Dayri, and Jean-François Muzy. Non-parametric kernel estimation for symmetric hawkes processes. application to high frequency financial data. *The European Physical Journal B-Condensed Matter and Complex Systems*, 85(5):1–12, 2012.

[8] Rama Cont and Adrien De Larrard. Price dynamics in a markovian limit order market. *SIAM Journal on Financial Mathematics*, 4(1):1–25, 2013.

[9] Emmanuel Bacry and Jean-François Muzy. Hawkes model for price and trades high-frequency dynamics. *Quantitative Finance*, 14(7):1147–1166, 2014.

[10] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

[11] Emmanuel Bacry, Thibault Jaisson, and Jean-François Muzy. Estimation of slowly decreasing hawkes kernels: application to high-frequency order book dynamics. *Quantitative Finance*, 16(8):1179–1201, 2016.

[12] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

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

[14] Olivier Guéant, Charles-Albert Lehalle, and Joaquin Fernandez-Tapia. Dealing with the inventory risk: a solution to the market making problem. *Mathematics and financial economics*, 7(4):477–507, 2013.

[15] Alvaro Cartea, Sebastian Jaimungal, and Jason Ricci. Buy low, sell high: A high frequency trading perspective. *SIAM Journal on Financial Mathematics*, 5(1):415–444, 2014.

[16] Alvaro Cartea and Sebastian Jaimungal. Optimal execution with limit and market orders. *Quantitative Finance*, 15(8):1279–1291, 2015.

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

[18] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[19] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[20] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, 6(2), 2016.

[21] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[23] Cicero Dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *International Conference on Machine Learning*, pages 1818–1826. PMLR, 2014.

[24] Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International conference on intelligent text processing and computational linguistics*, pages 171–189. Springer, 2011.

[25] Sunil Kumar Sahu. *Neural architectures for named entity recognition and relation classi cation in biomedical and clinical texts*. PhD thesis, Indian Institute of Technology Guwahati, 2018.

[26] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[27] Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 473–483, 2017.

[28] 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.

[29] 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 *2017 IEEE 19th Conference on Business Informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017.

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

[31] 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.

[32] Justin A Sirignano. Deep learning for limit order books. *Quantitative Finance*, 19(4):549–570, 2019.

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

[34] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[35] 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.

[36] Bruno Biais, Pierre Hillion, and Chester Spatt. An empirical analysis of the limit order book and the order flow in the paris bourse. *the Journal of Finance*, 50(5):1655–1689, 1995.

[37] Jean-Philippe Bouchaud, Marc Mézard, Marc Potters, et al. Statistical properties of stock order books: empirical results and models. *Quantitative finance*, 2(4):251–256, 2002.

[38] Ilija Zovko, J Doyne Farmer, et al. The power of patience: a behavioural regularity in limit-order placement. *Quantitative finance*, 2(5):387–392, 2002.

[39] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. In *The elements of statistical learning*, pages 9–41. Springer, 2009.

[40] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012.

[41] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[42] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[43] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5, 2001.

[44] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[45] Sagar Sharma and Simone Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

[46] P Sibi, S Allwyn Jones, and P Siddarth. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*, 47(3):1264–1268, 2013.

[47] Raymond E Wright. Logistic regression. *Reading and understanding multivariate statistics*, 1995.

[48] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[50] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[51] 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.

[52] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[53] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[54] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

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

[57] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[58] D. M. W. Powers. Evaluation: From precision, recall and f-measure to ROC., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

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

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

[61] Daniel Kahneman, D Kahneman, A Tversky, et al. Experienced utility and objective happiness: A moment-based approach. *2000*, pages 673–692, 2000.

[62] Thomas Lux and Frank Westerhoff. Economics crisis. *Nature Physics*, 5(1):2–3, 2009.

[63] Per Bak, Maya Paczuski, and Martin Shubik. Price variations in a stock market with many agents. *Physica A: Statistical Mechanics and its Applications*, 246(3-4):430–453, 1997.

[64] Nicholas F Britton et al. *Reaction-diffusion equations and their applications to biology.* Academic Press, 1986.

[65] Sergei Maslov. Simple model of a limit order-driven market. *Physica A: Statistical Mechanics and its Applications*, 278(3-4):571–578, 2000.

[66] Damien Challet and Robin Stinchcombe. Analyzing and modeling 1+ 1d markets. *Physica A: Statistical Mechanics and its Applications*, 300(1-2):285–299, 2001.

[67] Robert F Engle and Jeffrey R Russell. Autoregressive conditional duration: a new model for irregularly spaced transaction data. *Econometrica*, pages 1127–1162, 1998.

[68] Rama Cont, Sasha Stoikov, and Rishi Talreja. A stochastic model for order book dynamics. *Operations research*, 58(3):549–563, 2010.

[69] Mark Paddrik, Roy Hayes, Andrew Todd, Steve Yang, Peter Beling, and William Scherer. An agent based model of the e-mini s&p 500 applied to flash crash analysis. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on*, pages 1–8. IEEE, 2012.

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

[71] Luc Bauwens and Nikolaus Hautsch. Modelling financial high frequency data using point processes. In *Handbook of financial time series*, pages 953–979. Springer, 2009.

[72] Rama Cont, Arseniy Kukanov, and Sasha Stoikov. The price impact of order book events. *Journal of financial econometrics*, 12(1):47–88, 2014.

[73] Zoltan Eisler, Jean-Philippe Bouchaud, and Julien Kockelkoren. The price impact of order book events: market orders, limit orders and cancellations. *Quantitative Finance*, 12(9):1395–1419, 2012.

[74] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[75] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

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

[77] Darryl Shen. *Order Imbalance Based Strategy in High Frequency Trading*. PhD thesis, oxford university, 2015.

[78] 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*, 30(5):1407–1418, 2018.

[79] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[80] Alexandre Bérard, Christophe Servan, Olivier Pietquin, and Laurent Besacier. Multivec: a multilingual and multilevel representation learning toolkit for nlp. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016.

[81] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 36–45, 2015.

[82] Jan Chorowski, Ron J Weiss, Samy Bengio, and Aäron van den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE/ACM transactions on audio, speech, and language processing*, 27(12):2041–2053, 2019.

[83] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[84] H Eugene Stanley, LAN Amaral, P Gopikrishnan, P Ch Ivanov, TH Keitt, and V Plerou. Scale invariance and universality: organizing principles in complex systems. *Physica A: Statistical Mechanics and its Applications*, 281(1-4):60–68, 2000.

[85] Jerome L Kreuser and Didier Sornette. Bitcoin bubble trouble. *Forthcoming in Wilmott Magazine*, pages 18–24, 2018.

[86] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[87] Nikolaus Hautsch and Ruihong Huang. *Limit Order Flow, Market Impact, and Optimal Order Sizes: Evidence from NASDAQ TotalView-ITCH Data*, chapter 6, pages 137–161. John Wiley & Sons, Ltd, 2012.

[88] Takatoshi Ito and Yuko Hashimoto. Intraday seasonality in activities of the foreign exchange markets: Evidence from the electronic broking system. *Journal of the Japanese and International Economies*, 20(4):637–664, 2006.

[89] Joel Hasbrouck. Measuring the information content of stock trades. *The Journal of Finance*, 46(1):179–207, 1991.

[90] Fabrizio Lillo, J Doyne Farmer, and Rosario N Mantegna. Master curve for price-impact function. *Nature*, 421(6919):129–130, 2003.

[91] Wei-Xing Zhou. Universal price impact functions of individual trades in an order-driven market. *Quantitative Finance*, 12(8):1253–1263, 2012.

[92] Jean-Philippe Bouchaud and Marc Potters. *Theory of financial risk and derivative pricing: from statistical physics to risk management*. Cambridge university press, 2003.

[93] Nikitas Goumatianos, Ioannis Christou, and Peter Lindgren. Stock selection system: building long/short portfolios using intraday patterns. *Procedia Economics and Finance*, 5:298–307, 2013.

[94] Philippe Jorion. *Value at risk*. McGraw-Hill Professional Publishing, 2000.

[95] Edward O Thorp. The kelly criterion in blackjack sports betting, and the stock market. In *The Kelly Capital Growth Investment Criterion: Theory and Practice*, pages 789–832. World Scientific, 2011.

[96] Rasitha R Jayasekare, Ryan Gill, and Kiseop Lee. Modeling discrete stock price changes using a mixture of poisson distributions. *Journal of the Korean Statistical Society*, 45(3):409–421, 2016.

[97] 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.

[98] Iti Chaturvedi, Yew-Soon Ong, and Rajesh Vellore Arumugam. Deep transfer learning for classification of time-delayed gaussian networks. *Signal Processing*, 110:250–262, 2015.

[99] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

[100] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2019.

[101] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, pages 6754–6764, 2017.

[102] Geoffrey J McLachlan and David Peel. *Finite mixture models*. John Wiley & Sons, 2004.

[103] Stefany Coxe, Stephen G West, and Leona S Aiken. The analysis of count data: A gentle introduction to poisson regression and its alternatives. *Journal of personality assessment*, 91(2):121–136, 2009.

[104] Martin Ridout, Clarice GB Demétrio, and John Hinde. Models for count data with many zeros. In *Proceedings of the XIXth international biometric conference*, volume 19, pages 179–192. International Biometric Society Invited Papers. Cape Town, South Africa, 1998.

[105] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[106] Rui Hu and Stephen M Watt. An agent-based financial market simulator for evaluation of algorithmic trading strategies. In *6th International Conference on Advances in System Simulation*, pages 221–227. Citeseer, 2014.

[107] Jianling Wang, Vivek George, Tucker Balch, and Maria Hybinette. Stockyard: A discrete event-based stock market exchange simulator. In *2017 Winter Simulation Conference (WSC)*, pages 1193–1203. IEEE, 2017.

[108] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. Bleurt: Learning robust metrics for text generation. *arXiv preprint arXiv:2004.04696*, 2020.

[109] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

[110] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[111] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[112] Rui Huang, Shu Zhang, Tianyu Li, and Ran He. Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis. In *Proceedings of the IEEE international conference on computer vision*, pages 2439–2448, 2017.

[113] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[114] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[115] Jyh-Jing Hwang, Sergei Azernikov, Alexei A Efros, and Stella X Yu. Learning beyond human expertise with generative models for dental restorations. *arXiv preprint arXiv:1804.00064*, 2018.

[116] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[117] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[118] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*, 2018.

[119] Yuyang Wang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. Deep factors for forecasting. In *International Conference on Machine Learning*, pages 6607–6617. PMLR, 2019.

[120] Kang Zhang, Guoqiang Zhong, Junyu Dong, Shengke Wang, and Yong Wang. Stock market prediction based on generative adversarial network. *Procedia computer science*, 147:400–406, 2019.

[121] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018, 2018.

[122] Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261, 2019.

[123] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[124] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *arXiv preprint arXiv:1305.6663*, 2013.

[125] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[126] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[127] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*, 2015.

[128] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[129] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[130] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[131] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[132] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[133] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[134] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[135] Weibing Huang, Charles-Albert Lehalle, and Mathieu Rosenbaum. Simulating and analyzing order book data: The queue-reactive model. *Journal of the American Statistical Association*, 110(509):107–122, 2015.

[136] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[137] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

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

[139] Charles W Dunnett and Ajit C Tamhane. A step-up multiple test procedure. *Journal of the American Statistical Association*, 87(417):162–170, 1992.

[140] Abhay K Singh, David E Allen, and Powell J Robert. Extreme market risk and extreme value theory. *Mathematics and computers in simulation*, 94:310–328, 2013.

[141] Yannick Malevergne*, Vladilen Pisarenko, and Didier Sornette. Empirical distributions of stock returns: between the stretched exponential and the power law? *Quantitative Finance*, 5(4):379–401, 2005.

[142] Francois M Longin. The asymptotic distribution of extreme stock market returns. *Journal of business*, pages 383–408, 1996.

[143] John B Taylor and John C Williams. A black swan in the money market. *American Economic Journal: Macroeconomics*, 1(1):58–83, 2009.

[144] Carlos M Jarque and Anil K Bera. Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics letters*, 6(3):255–259, 1980.

[145] A Assaf. Extreme observations and risk assessment in the equity markets of mena region: Tail measures and value-at-risk. *International Review of Financial Analysis*, 18(3):109–116, 2009.

[146] Manfred Gilli and Evis Këllezi. An application of extreme value theory for measuring financial risk. *Computational Economics*, 27(2):207–228, 2006.

[147] Ramazan Gencay and Faruk Selcuk. Extreme value theory and value-at-risk: Relative performance in emerging markets. *International Journal of Forecasting*, 20(2):287–303, 2004.

[148] S Drożdż, M Forczek, J Kwapień, P Oświe, R Rak, et al. Stock market return distributions: From past to present. *Physica A: Statistical Mechanics and its Applications*, 383(1):59–64, 2007.

[149] Ole E Barndorff-Nielsen and Neil Shephard. Volatility. *Encyclopedia of Quantitative Finance*, 2010.

[150] Roger W Klein and Vijay S Bawa. The effect of estimation risk on optimal portfolio choice. *Journal of Financial Economics*, 3(3):215–231, 1976.

[151] Neil Shephard. *Stochastic volatility*. Oxford University, 2005.

[152] Yacine Aït-Sahalia, Per A Mykland, and Lan Zhang. Ultra high frequency volatility estimation with dependent microstructure noise. *Journal of Econometrics*, 160(1):160–175, 2011.

[153] Torben G Andersen and Viktor Todorov. Realized volatility and multipower variation. *Encyclopedia of Quantitative Finance*, 2010.

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