LEARNING THE LIMIT ORDER BOOK: A COMPREHENSIVE MIX BETWEEN STOCHASTIC AND MACHINE LEARNING MODELS FOR GENERATION AND PREDICTION

LEARNING THE LIMIT ORDER BOOK: A COMPREHENSIVE MIX BETWEEN STOCHASTIC AND MACHINE LEARNING MODELS FOR GENERATION AND PREDICTION

by

Muhammed Imran ÖZYAR

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday July 27, 2021 at 16:00.

Student number:	4458508	
Thesis comittee:	Dr. ir. J. H .M. Anderluh,	TU Delft, supervisor
	Prof. M. J. T. Reinders,	TU Delft
	Dr. ir. J. H. Krijthe,	TU Delft

Contents

SUMMARY

The field of finance is an interesting field in which much research takes place. In particular, its sub-field of modeling the dynamics of order books is an interesting field, since it translates into modeling the behaviour of traders on the market. Most of the models proposed in this field can be divided into stochastic models or machine learning models. The problem with the former ones is that they often make assumptions that do not propagate to practical applications, while the problem with the latter is that the models are often incomprehensible, even though they yield good results. This work tries to combine both of those fields to maintain the transparency of stochastic models and the accuracy of machine learning models. We target the limit order book, which is a book used in financial markets to place orders for financial securities in. It is a challenge to both generate realistic order books and predict them as well, since the model should be able to understand the underlying dynamics of order books to do this. We present a deep-learning-based model that is able to generate order books, through the use of a generative adversarial network (GAN) like architecture, and is able to predict order books using a similar architecture for the generator. Our models are able to outperform traditional generative and predictive models in both instances, while it remains transparent in its calculation: it is still able to extract useful and interpretable features from the underlying data set. These features, like the probabilities for up or down movements, or stagnations, or the rates at which the variables within the order book move, can be used by traders or algorithms to make sense out of the predictions. Traders can, for example, ask themselves whether the extracted features are representative of the analyzed asset and can, through this question, build additional confidence in the model. Another use case could be to utilize those features as inputs to other models. To our knowledge, this is the first work that proposes a combination of a stochastic and machine learning model to learn the dynamics of the order book.

1

INTRODUCTION

In this chapter, we shall introduce the topic of limit order book dynamics from both the mathematical and machine learning perspectives. We explain how both sides have their pros and cons and what the current state of the field is. We shall briefly discuss ongoing and past research in the field and the gaps that our research aims to fill in. We conclude the chapter with a list of our contributions and provide an outline of the rest of the work.

1

1.1. BACKGROUND

The financial market is a very intriguing field for mathematicians. Many researchers have come up with ways to predict stock prices, their simulations, and models describing the dynamics of the orders that work the prices up or down [11, 10]. Not only mathematicians have shown interest in this field, however. The progress the computer science field has made showed the applicability of machine learning models to the financial market as well [58, 53].

To buy or sell a financial asset, such as a company stock, a trader might want to set a given price against which they wish to place their trade. The price determined by the trader might not be one that another trader is willing to sell or buy for, respectively. In that case, the order cannot be executed directly: this type of order is called a limit order and it gets placed in a book, called the limit order book (LOB). The limit order book is traditionally sorted by the best bid and best ask prices, which are, respectively, the highest bid and the lowest ask prices. A trader might also want to place a trade against the market price. In this case, the best bid or ask price is selected. These types of orders are called market orders. Note that even though we will be using the term 'trader' throughout our work, this will usually refer to high-frequency traders (HFT). HFTs are algorithms that operate on the microstructure of the market and analyze data on a micro time scale, usually defined in terms of microseconds [36]. These algorithms buy and sell financial assets, as the name suggests, in a highly frequent manner to maximize profits.

As an example for order books, let us take a look at Figure 1.1. Here we depict a simple order book of a company's stock with three levels on the bid and ask sides. A level in the order book refers to a price level, so the top-level bid price is \$3, the second level bid price \$2, and so on. The top-level prices can be referred to as the level one prices, but we shall continue to refer to them as the best bid and ask prices. Assume for simplicity that each block represents a unique trader which has submitted a limit order. In the first situation, there are three best bids of \$3 in the order book and one best ask of \$6. In this hypothetical example, one trader that had submitted a limit order to buy a share of the company for \$3 suddenly realizes that the difference between the best bid and best ask prices is too large. He decides to cancel his limit order and replaces it with a market order to buy one share. The best ask price at that moment is \$6, so the trader buys that share. As a result, the gap between the best bid and ask prices got bigger. This is just one example of a dynamic within order books. Another dynamic might be considered with the question: what happens after the trader submitted his market order? Do the other bids get replaced by market orders as well, since the other traders are too afraid that the price might rise even further, or do the ask prices get lower because they might suspect no one would buy their shares for such a high price? Naturally, all these decisions depend on multiple factors, such as the sentiment of the traders and market circumstances. It is indeed a very interesting topic for both stochastic, as well as machine learning models.

The field of modeling order book dynamics can be roughly separated into two kinds of models: stochastic models and machine learning models. Both of these models, however, share one common part: they zoom into very small time frames to learn the micro-dynamics of the order book, as opposed to other sub-fields, such as derivative pricing.

Derivatives are financial contracts between two parties that serve various purposes. An example of a derivative is a futures contract, which gives the buyer the obligation

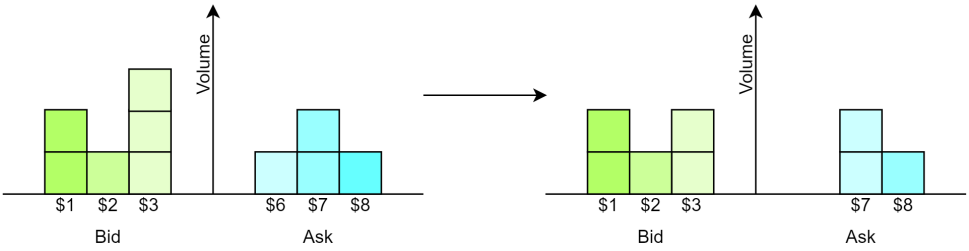

Figure 1.1: An example of two order books, where a buyer cancels his limit order and replaces it with a market order.

to buy a specified asset (within the contract) at a specified time for a specified price in the future. This derivative resembles a call option, which is a contract that gives the buyer the right, but not the obligation, to do the exact same. These derivatives have prices for which they are sold and the field of derivative pricing considers itself with this. However, since the times specified in those contracts usually range in the domain of months, the models deployed to price the derivatives have to take long-term information into account. The prices of these derivatives depend on a couple of factors, of which the price of the underlying asset is one of. This is the same price, of course, as depicted by the top-level bid and ask prices in the order book, so there is indeed some connection between these two fields.

We shall zoom in a bit more into derivative pricing models to see how they relate to limit order book models. In classic derivative pricing theory, the Black-Scholes model is used. This model is a partial differential equation (PDE) which, through some assumptions, governs the price evolution of derivatives. One of these assumptions is that the returns on the underlying asset are log-normally distributed. This follows from the fact that the underlying asset, in the Black-Scholes model, follows a geometric Brownian motion (GBM). GBM models the stock price S_t at time step t through

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right), \tag{1.1}$$

for a drift parameter μ, volatility σ, start price S_0, and Wiener process W_t. As mentioned before, a property of the GBM is that the returns are log-normally distributed, which may not always be the case. To see how, we refer to [41] for the exact derivation of all its properties. This assumption underestimates the probabilities of extreme events happening, i.e. the possibility of achieving much higher or much lower returns. Furthermore, GBM also assumes σ to be constant, while volatility varies in practice. Finally, GBM is continuous at every point, meaning that it is unable to model sudden jumps in stock prices, which occur due to unpredictable events happening. Hence, trying to use GBM as an underlying stock price model to generate or predict realistic LOB data will result in inaccurate samples and bad accuracy scores, respectively.

Stochastic models are the traditional ones and were there even before machine learning made its entry into the field. Stochastic models utilize empirical observations and other convenient assumptions to construct rigorous models to model the underlying

order book dynamics. It had, for example, empirically been observed that once there would be much movement in the price of a share, the volume from lower levels of the order book would shift upwards towards higher levels, meaning that more traders would be tempted to take part in the active trade. In their work, [10] constructed a stochastic model that was able to model this behaviour, while making additional assumptions. One of these assumptions was that high-frequency traders would affect the order flow in a similar manner to that of a Wiener process. Other assumptions made include, but are not limited to, the assumption that the arrivals and cancellations of limit orders happen through independent Poisson-point processes [11]. There is ample empirical evidence to suggest such an approach, but there is also ample evidence that counters it.

On the other hand, we have machine learning models. Machine learning models benefit from the fact that they usually make no underlying assumptions about the data. In return, however, they often become a black-box model or simply too convoluted to make sense out of them. Random forests are a great example of the latter. Random forests are populated with decision trees that are trained to branch off based on learned conditions for some given input features. However, the trees get wider as they get deeper to deliver better results. A random forest usually contains hundreds of those trees: it gets quite convoluted very quickly, even though the concept seems relatively easy to comprehend: and we have not even touched upon neural networks yet!

Neural networks are on the rise, and along comes great misery, but also promise. These models are able to accurately classify and regress on the underlying data. Recent progressions [17] even made them able to generate new, unseen, yet similar data from a given data set. However, these models are not particularly human-friendly: we are not able to understand why the neural network gives the classification or regression it gives. Various neural networks have been deployed in the field of finance and have achieved promising results as well, but they lack explainability, something that the mathematical models build their foundations on. Various researchers claim that their machine learning models, whether it would be support vector machines or neural networks, are able to learn the underlying dynamics in LOBs solely based on the results that they have acquired. However, good results do not imply a sound understanding of the dynamics within order books: they solely imply that the model was able to match patterns within the input data to outputs that provide optimal measurements. Knowing why a model gives the output it does is especially important in the financial field, where lots of assets are at stake, so there is ample need for risk management. Trusting a machine is hard on its own, let alone trusting a machine solely based on its answer, without any argumentation. A trader who is able to extract useful information from a model, apart from its answer, can utilize it to both build confidence in the model and utilize the information for other purposes, such as to gain more insights into the underlying data.

This research focuses on combining a class of machine learning methods with mathematical models, to combine accuracy with explainability in modeling the dynamics of the order book. In particular, we distinguish two goals for this research, which both share a common objective:

1. learn to generate synthetic LOB data;

2. learn to predict an LOB given its previous state(s).

Both of these goals have in common that we should be able to construct a model for the limit order book. This model, however, comes with constraints: the model should be interpretable and explainable. Note the subtle difference between interpretability and explainability: the former suggests that the cause and effect of a model can be observed, while the latter suggests that the internal mechanics of the model can be explained in human terms. The financial market is a fast-moving one: many decisions should be quickly made and sufficiently substantiated. If a model outputs a certain decision, a trader should be able to look into the assumptions and parameters the model has learned to build confidence in its decision. Hence, the part of the model that makes the final decisions for the generation or prediction parts should not be a black-box model.

Although it might be obvious why we would want a model to predict a limit order book might be obvious, namely for the prediction of the price movement of an asset to gain a better understanding of how the asset moves and to realize more profits or to cut losses. The reason why we would want to generate synthetic limit order book data might be less obvious. In [2] it is noted that through synthetic data generation, we would be able to train more advanced machine learning methods, since they require large data sets to train. Furthermore, the lack of historical data for certain events in the financial market, such as flash crashes, recessions, and new trading behaviours, can be compensated by this kind of data. Also, through centralizing the data used in research within the field, we are able to keep all researchers on the same level for data accessibility, since not all data sets are publicly available.

In this work, we aim to learn the underlying distribution in the probability distribution of price movements: the probabilities for up, down, and stagnation movement. We tackle this problem by proposing a new kind of random walk based on Markov theory. Through this random walk, we are able to learn these probabilities. Furthermore, we aim to learn the rates of price movements. Combining these two properties, along with a generative adversarial network (GAN), we aim to generate synthetic LOB data. By slightly altering the network and loss functions, we shift our model's purpose from generation to prediction.

1.2. CONTRIBUTIONS

We list the main contributions of our work as follows:

- we propose a comprehensive mix between a deep learning model and a stochastic model to generate and predict order books. This model is not limited to be used for order books only and can be used to generate and predict other spatio-temporal time series as well;
- we formalize order books and propose losses for machine learning methods to utilize them;
- we propose a new kind of random walk to learn transition matrices with short mixing times for the purpose of longer walks, compared to standard random walks through Markov chains, through backpropagation;
- we have gathered one million order books for the bitcoin cryptocurrency with a depth of ten levels, each of them separated by 100 milliseconds, to be used by

other researchers as well. The data, along with the code used in the research, can be acquired from https://github.com/metahexane/learning_lob;

- we propose metrics to evaluate the quality and diversity of generated order books;
- we conduct a thorough sensitivity analysis and ablation study of our model to learn more about its behaviour and the used data sets.

1.3. OUTLINE

In the following chapter, we shall discuss the related work and the theoretical background needed for this work. Here, aside from discussing research within the field, we will give a primer on neural networks, generative models and Markov chains. In Chapter 3 we dive into the proposed stochastic and machine learning models used for generation and prediction. There we introduce the deep learning architectures, along with the mathematical models attached to them. We also introduce novel LOB-specific losses in the same chapter. In Chapter 4, we talk about the data sets that we will be using for our experiments and about the experiments themselves. We also discuss the setup for the sensitivity analysis and ablation study. The results of these experiments are presented, discussed, and thoroughly analyzed in Chapter 5. Finally, we conclude with a summary of the achieved performances and discuss future work in Chapter 6.

2

BACKGROUND

In this chapter, we will be discussing the relevant literature more in-depth. Furthermore, we will be introducing the theory behind this research, where we will be explaining deep learning and probabilistic modeling concepts that our research involves. We shall start off from the base of deep learning, namely multilayer perceptrons, and build our way forward towards generative adversarial networks. Finally, we conclude with a primer on Markov chains.

2.1. RELATED WORK

The work on limit order book modelling may be grouped into two fields: stochastic models and machine learning models. Even though there is a lot of work done in the former field, there is also a rise of the latter field, because of the novel inventions done within.

Most work on modeling the dynamics of LOBs, as mentioned, are stochastic models [11, 10], which usually come with a handful of assumptions, such as the fact that limit order and market order arrival times each arrive at independent, exponential times, and both of these events happen mutually independent as well. In addition, the stochastic models usually utilize other unvalidated assumptions, unobservable parameters or may not be computationally efficient [5, 6, 14, 40].

Although various machine learning methods have been deployed to learn the dynamics of limit order books (LOBs), they usually end up predicting the stock price movement [58, 50, 31], failing to converge [56], or deliver promising generative results, but remain a black-box model [33]. In the first case, the authors claim that the machine learning method learned the dynamics of the order book. Rather, the machine learning method has learned to recognize patterns within the order book, and learned to match those patterns to certain outputs. This is not the same as learning the dynamics of an order book, since they should be able to prove that the method has learned the dynamics, in any other way than showing that the predictions are more accurate compared to traditional methods.

The field of spatio-temporal prediction through GANs has been thoroughly explored by [15]. They mention that novel architectures and loss functions may be explored for advancements in the field. In addition, it is mentioned that evaluation metrics for GANs still pose a problem, as they do in non-spatio-temporal prediction related fields that still utilize GANs. The work of [45] showed the applicability of a spatio-temporal prediction GAN to taxi demand within a city and its ability to surpass traditional as well as deep learning-based prediction methods. There is a slight pitfall in their experimental setup: the deep learning models they used for their comparisons are quite shallow, compared to their own network. Their network consists of multiple 3D and long-short term memory (LSTM) convolutional networks (CNN) and multilayer perceptrons (MLP), while the compared models are CNNs or MLPs with a few layers. In addition, their used architecture still lacks quite some explanation in their paper about the layers, which makes their paper, in its current state, irreproducible. Furthermore, the work of [59] utilizes basic indicators of a stock to predict the close price of the next time step using a GAN. This research fails to capture more data on the stock, such as deeper levels in the order book and volumes per level. The model also remains a black-box model, not revealing much about its inherent dynamics.

As we shall see, we will treat the LOB as an image. This is not common practice in literature. The main reason for it is that researchers often do not use models that are able to process images. Usually, these models are either mathematical or machine learning models not capable of processing highly dimensional input. This is the consequence of the fact that models capable of processing images, like CNNs, are newly on the rise and it might not be outright obvious to use models that are often used for object detection [16], segmentation [35], and digit classification [47] to process LOB data. The works of [13, 8] are one of the first works to use CNNs to process LOB data, but they lack a thorough

understanding of the workings of CNNs. They argue that CNNs are better suited for processing LOB data solely based on their improved results, without trying to exploit the advantages of CNNs: namely to exploit local patterns in the input data. The work of [58] is actually the only work, to our knowledge, that tried to exploit the workings of the CNN by adjusting their settings to that of their LOB structure. However, as we shall see in Section 3.2, their setup for the convolutional layers cannot be deemed correct according to their description.

2.2. THEORETICAL BACKGROUND

In this section, we shall dive into the theory behind deep learning and its branches into computer vision and generative models. We shall start off at the heart of deep learning: artificial neural networks, and build our way towards generative adversarial networks. We shall also briefly discuss some theory behind Markov chains, which hold a vast position in probabilistic modeling. Most of the mathematics that deep learning is considered with and is used in our research is discussed here and in the next chapter.

2.2.1. MULTILAYER PERCEPTRON

The artificial neural network (ANN) lays the basis for deep learning and is widely used in all kinds of applications. Here we shall consider the ANN in its most basic form, which is the multilayer perceptron (MLP). The multilayer perceptron consists, as the name suggests, of multiple layers of neurons, where a layer would look like the one depicted in Figure 2.1. Given an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and a weight vector $\boldsymbol{w} \in \mathbb{R}^n$, the output y of a neuron would be

$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b), \tag{2.1}$$

where b is a bias term and $\sigma(\cdot)$ is a non-linear activation function, such as ReLU ($\sigma(x) = \max(0, x)$) or leaky ReLU, defined as

$$\sigma(x) = \begin{cases} x, & \text{if } x > 0 \\ cx, & \text{otherwise} \end{cases} \tag{2.2}$$

for a pre-defined $c \in \mathbb{R}$. Note that the same activation function is used for each neuron in the same layer and that it usually does not contain learnable parameters, but variations exist [19]. We shall also consider the logistic activation function and the softmax functions, which are both activation functions, since they are used throughout our work as well. The logistic activation function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.3}$$

and maps an input value $x \in \mathbb{R}$ to an output $\sigma(x) \in (0, 1)$. The softmax function is applied to a vector $\boldsymbol{x} \in \mathbb{R}^n$, as opposed to the logistic activation function, and is used to transform its values to ones that make it sum to one, such that $\sum_{k=1}^{n} \sigma(\boldsymbol{x})_i = 1$, through

$$\sigma(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_{k=1}^{n} e^{\boldsymbol{x}_k}}. \tag{2.4}$$

Figure 2.1 might be confusing at first, however, one should realize that the weights are implicitly defined as the edges between the nodes and not as the nodes themselves. The weight and biases, respectively denoted as $\boldsymbol{w} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$, are often combined into one parameter $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$ in literature, and are learned through optimizing a given loss function $\mathscr{L}(\boldsymbol{\theta}) : \mathbb{R}^n, \mathbb{R}^m \to \mathbb{R}$, for some $n, m \in \mathbb{Z}^+$. In particular, the derivatives of this loss function with respect to the network parameters are calculated and the parameters are then updated accordingly through gradient descent at step t throughout the optimization process, defined as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}}, \tag{2.5}$$

where α is the learning rate, which determines the magnitude of the updates for the network parameters. A problem with this straightforward approach is that the input data might become quite large, as is often seen in machine learning models. Utilizing gradient descent becomes less efficient, since the gradients for all input data points should be calculated. Instead, we might specify a batch size and calculate the average gradient of the batch instead to update our weights. This batch would then be randomly sampled from our input data. This process is called stochastic gradient descent (SGD) and many variations to ensure a more stable training period exist, such as RMSProp [23] and Adam [32]. In this work, we will be using only Adam as our optimization algorithm, so we shall discuss its workings. Adam computes individual learning rates for the model's parameters through the computation of the first and second moments of the gradients. The algorithm consists of a few equations:

$$m_t = \frac{1}{1-\beta_1^t}\left(\beta_1 m_{t-1} + (1-\beta_1)\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}}\right), \tag{2.6}$$

$$v_t = \frac{1}{1-\beta_2^t}\left(\beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}}\right)^2\right), \tag{2.7}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}, \tag{2.8}$$

with $\epsilon \to 0$ and $m_0 = v_0 = 0$. As we can see from Eqs. 2.6 and 2.7, the method computes a moving average of the first and second moments of the gradients of the parameters of the model, while using bias correction to compensate for the first few data points. The parameters α, β_1, and β_2 are manually set beforehand and are hence not learned throughout the optimization.

Finally, we shall introduce the Kullback-Leibler (KL) divergence, which is often used within loss functions. The KL divergence measures the difference between two probability distributions. To give an intuition behind the idea of this measure being used in loss functions, imagine a neural network, denoted by a function $h : \mathbb{R}^n \to \mathbb{R}^n$. Its input will be $x \in \mathbb{R}^n$ drawn from a uniform distribution $U(0,1)$. Its output, $h(x)$, will be a vector $\mathbb{R}^n$ that should resemble a Gaussian distribution. For simplicity, take our true data to be $y \in \mathbb{R}^n$ where each element is drawn from $\mathcal{N}(0,1)$. Hence, our network h will serve as a mapping between a uniform and normal distribution. To make our network learn this mapping, we will try to learn our weights and biases such that the KL divergence is

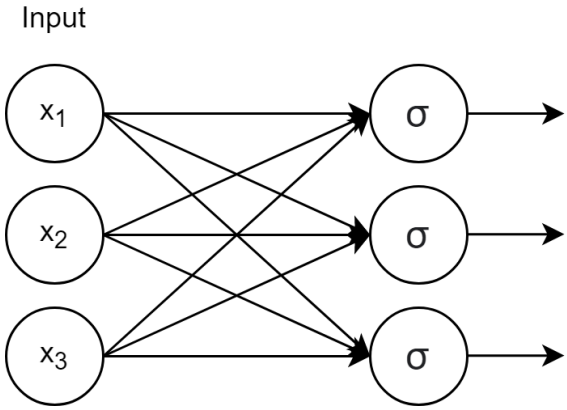

Figure 2.1: An example of an single layer perception with three hidden neurons and an input of dimension 3. Each neuron receives the input vector as its input. The outputs of the neurons can be fed into the next layer of neurons, in which case each neuron in the next layer would similarly receive the output vector of this layer.

minimized. In particular, if we let P and Q be two probability mass functions within a probability space $\mathscr{X}$, the KL divergence may then be defined as

$$D_{KL}(P||Q) = \sum_{x \in \mathscr{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{2.9}$$

The KL divergence can be interpreted as the expected logarithmic difference between probabilities P and Q, taken over the probabilities of P. The KL divergence will also be utilized later on in a pairwise manner between multiple probability distributions. This shall be discussed in Section 3.7.

2.2.2. CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNN) are networks that are often applied in computer vision tasks because of their ability to exploit spatial structures in the data. They are mainly composed of convolutional layers (along with an activation function) and pooling layers, usually followed by a fully connected MLP. In practice, a convolution operation, which is applied in a convolutional layer, is often implemented as a cross-correlation operation, since they are similar operations and the latter is more computationally efficient. A cross-correlation operation results in an output matrix G, given an input matrix H and filter matrix F and is denoted as

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v) H(i+u, j+v), \tag{2.10}$$

for a given filter size $2k+1$. In a convolution operation, the filter matrix F would be flipped first, such that we would have $F(i-u, j-v)$ in Eq. 2.10. In a CNN, the filter matrix F is learned through training the network: the weights in F are optimized through minimizing a given loss function. In a convolutional layer, we have multiple filters that all learn different weights. The number of filters are called channels and are often depicted

as C. Eq. 2.10 is applied over the whole input matrix H by sliding the filter over the input in both dimensions, such that we calculate $G(i+k, j+l)$. The variables k and l are the step sizes and are called the strides. They determine how much the filter moves when it slides over the input. Often, the stride of a layer is denoted by (k, l). Sometimes, an entry i, j in the input matrix H may be involved in more cross-correlation operations than an entry u, v. To compensate for this, we apply padding to the input matrix H. This means that we pad H with zeroes around the edges. The padding size determines how much we pad the input. Similar to stride, the padding of a layer is denoted by a tuple (p, q) to indicate the padding in both dimensions. The kernel size, stride, and padding all influence the output dimension of G. Finally, after the convolutional layer, a non-linear activation function, such as the (leaky) ReLU function, is usually applied to each entry of the filter.

Furthermore, there are pooling layers which involve a sliding window of a given size over the input matrix H. At each step, depending on the type of pooling, the maximum, average, or minimum (alternatives exist) value of the values within the sliding window is taken. Similar to the convolutional layers, we also have strides and paddings for pooling. This way the dimensionality of the input can be reduced by summarizing the information within a window. For example, an input matrix H of dimensions 4×4 can be reduced, using a sliding window of size 2×2, to an output of 2×2, given that the window slides in steps of 2 as well, meaning that we set our stride to 2 in both dimensions, resulting in $(2, 2)$.

Aside from the non-linear activation function, layers often also include a normalization layer, such as batch normalization [27], layer normalization [4], and group normalization [57]. These methods aim to normalize the outputs of each layer to reduce the internal covariate shift: the change in the distribution of neuron activations due to a change of the network's weights. In addition, the weights of the network become unbiased to smaller or larger values of the input since they get normalized to a zero mean and unit variance. Batch normalization normalizes each feature of the input across the batch dimension, whereas layer normalization, as the name suggests, is applied across the feature dimension of a network layer. Group normalization works similarly to layer normalization, but instead of applying the normalization across all layers, it splits them up in groups first.

Next to normal convolutions, there also exist transposed convolutions. Transposed convolutions are able to upsample an input matrix to a matrix with larger dimensions. They do this by applying padding to the original input, after which an operation as depicted in Eq. 2.10 is applied to the padded input. This process is exactly depicted in Figure 2.2. The weights of a transposed convolutional layer are hence learned in the same way as that with weights in a convolutional layer.

2.2.3. GENERATIVE ADVERSARIAL NETWORKS

A generative adversarial network (GAN) is, as the name suggests, a generative model. Before we dive into GANs, we shall first introduce the concept of generative models. As the name suggests, generative models are models that are able to generate data. Given the underlying distribution of some input data, say X_{data}, a generative model shall try to approximate this distribution. As a result, the model will yield X_{model}, which is the model's approximation of X_{data}.

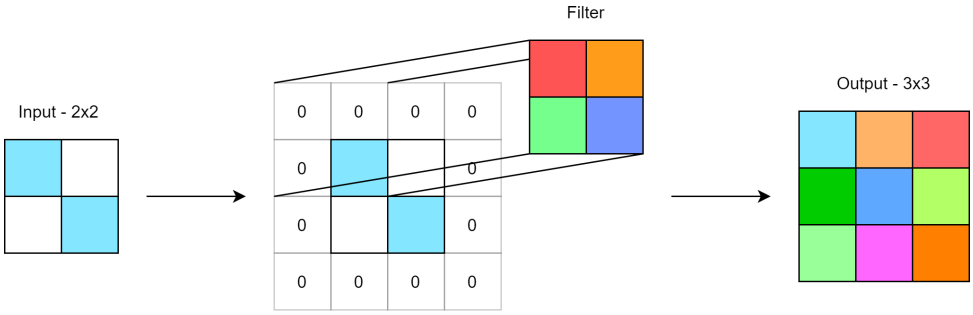

Figure 2.2: A transposed convolution operation on an input matrix of 2×2. It gets padded with zeroes and a 2×2 filter is applied with stride 1. The resulting output is a 3×3 matrix.

The simplest case of a generative model would be one where we would construct a histogram from the input data and sample from it accordingly. The pitfall of this method is that it does not scale well to higher dimensions. To see why, let us take an example from images. Suppose that we have 10×10 images within our data set and that every pixel in the image is either dark or light. Let us take an example of 100, not necessarily unique images where there is one arbitrarily chosen dark pixel in each image. Plotting out a histogram for this data set, where we would see the frequency of each image in our 100 samples, would enable us to calculate the probability of sampling a given image from the data set. However, to be able to finely approximate these probabilities, we would need more samples. Take for example a data set where we take every possibility of dark-light colored 10×10 image. This means that we need at least 2^{100} unique images in our data set to start calculating probabilities. Even then we could only get as far as a uniform distribution, so we would need more samples to construct better approximations of the distribution. This is where modern generative models come to the rescue.

A GAN consists of two separate networks, called the generator (G) and the discriminator (D). These two networks can either be multi-layer perceptrons or convolutional networks, but are not limited to those: any differentiable function or ANN would suffice, as long as both models fulfill their purpose. In short, the generator will try to generate samples that look similar to samples from a given data set. The discriminator, in its turn, shall try to distinguish fake samples from the generator from real samples from the data set.

Initially, the generator receives an input vector $\boldsymbol{z} \in \mathbb{R}^w$ for some specified dimension $w > 0$. This vector is randomly sampled from a prior distribution $\mathbb{P}_{\boldsymbol{z}}$. Through various layers of the network, this vector gets transformed to the desired output size $\mathbb{R}^{c \times c}$, in the case where the generator will try to generate square images of dimension $c \times c$. Hence, G is a function defined as $G : \mathbb{R}^w \to \mathbb{R}^{c \times c}$. Take for example a generator that tries to generate 64×64 images of cats. In this case, the generator will receive a vector $\boldsymbol{z}$ and upsample it through transposed convolutional layers. The final result $G(\boldsymbol{z})$ will then be a matrix of dimensions 64×64.

The discriminator receives its input from two sources: generated data $G(\boldsymbol{z})$ from the generator and real data from the data set that we want to generate images of. In

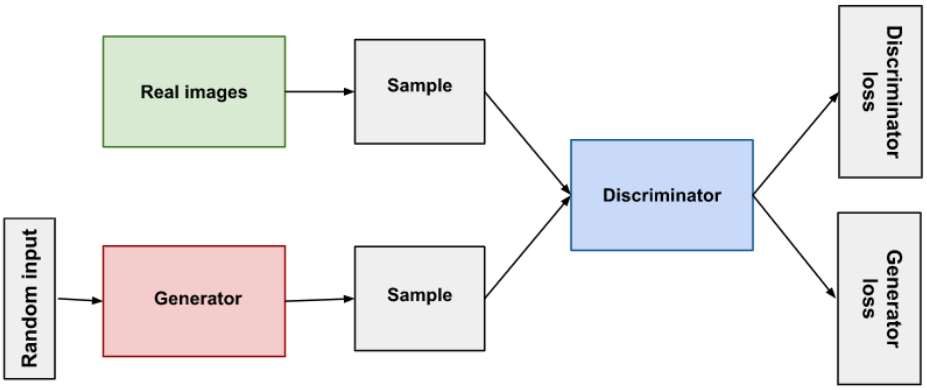

Figure 2.3: An overview of the workings of a GAN. Source: Google Developers

our example, the discriminator would receive both $G(\boldsymbol{z})$ and real 64×64 images of cats. Through multiple convolutional layers, the discriminator will finally output a scalar value between 0 and 1, representing the probability of the realness of the input cat image. Hence, the discriminator is actually a classifier, defined as a function $D : \mathbb{R}^{c \times c} \rightarrow \mathbb{R}$.

Having these two networks, we are finally able to train both of them. Training a GAN, as done originally in [17], is done through optimizing the following function:

$$J(G,D) = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_z}[\log(1 - D(G(\boldsymbol{z})))]. \tag{2.11}$$

In this equation $\boldsymbol{x} \sim \mathbb{P}_r$ represents the provided 'true' data, whereas $\boldsymbol{z} \sim \mathbb{P}_z$ represents the noise vector (z) which the generator had sampled from a prior distribution $\mathbb{P}_z$. In practice, the expectations translate into taking the averages of the values in Eq. 2.11 over the batch dimension, in which case we assume that all samples $\boldsymbol{x}$ and $\boldsymbol{z}$ are equally important: we do not prioritize any input sample $\boldsymbol{x}$ or $\boldsymbol{z}$ over any other sample. The GAN is then modeled as a minimax game with value function $J(G,D)$, where the generator tries to minimize it, and the discriminator tries to maximize it. The weights of both networks are then updated accordingly using stochastic gradient descent, or any other optimization algorithm, like Adam. Note that the generator has no influence on the first part of Eq. 2.11, hence it can only optimize the latter part. In addition, the generator updates its weights through the output of the discriminator, so the it never sees the real data it tries to approximate. A complete overview of the workings of a GAN can be seen in Figure 2.3.

GANs often suffer from a problem called mode collapse. Mode collapse is a phenomenon where the generator maps multiple vector samples to the same output, reducing its output diversity [26, 52]. The reason for this is that the generator learns to generate a sample that almost always succeeds in fooling the discriminator. The discriminator will then learn to always reject that specific sample, however, it may get stuck in local optimum while doing so. As a result, the generator will be able to keep generating that specific sample or another sample up until the discriminator is not able to account for them. As a solution to mode collapse, researchers have proposed novel architectures

[12, 24] and robust loss functions [1, 38]. For this work, we aim to keep the architectures of our generator and discriminator simple, while adding in a mathematical model to the generator. Hence, we choose to tackle this problem through a loss function. In particular, we will be using the Wasserstein GAN (WGAN) [1] because of its wide adoption [55].

There are a couple of changes to WGAN compared to the standard GAN. The largest change is the loss function, which is defined as Eq. 2.11 for GANs, whereas WGAN uses

$$J_{WGAN}(G,D) = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]. \tag{2.12}$$

This equation is a transformation of the Wasserstein distance, also known as the earth mover's (EM) distance, and is a measure for the distance between two probability distributions. A constraint of the WGAN is that the discriminator should be a 1-Lipschitz function. A 1-Lipschitz function is a function $f : \mathbb{X} \to \mathbb{Y}$, along with distance functions d_X and d_Y on the sets X and Y, respectively, that satisfies

$$d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2), \tag{2.13}$$

for $x_1, x_2 \in \mathbb{X}$. The original authors of WGAN satisfy this constraint by clipping the weights of the discriminator to $[-c, c]$ for $c \in \mathbb{R}^+$. This is, however, as the authors have mentioned, a terrible way to enforce the Lipschitz constraint, since if c is too large, it may take a long time for the weights to reach their limit. If c is too small, then we might have vanishing gradients with more deeper networks. In addition to the loss function, the discriminator gets renamed to a critic in WGAN, since its job slightly changes: instead of outputting a probability on the realness of the input, it outputs an unbounded scalar score that reflects the realness of the input, instead of a probability. Hence, the critic is not a classifier anymore. Furthermore, the critic is trained n_{critic} times more often than the generator.

In the work of [20], the authors have proposed a gradient penalty (GP) to enforce the Lipschitz constraint as a replacement for weight clipping. In particular, the WGAN-GP loss becomes

$$J_{WGAN-GP}(G,D) = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original WGAN loss}} + \underbrace{\lambda\,\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_{\tilde{\boldsymbol{x}}}}[(||\nabla_{\tilde{\boldsymbol{x}}}D(\tilde{\boldsymbol{x}})||_2 - 1)^2]}_{\text{Gradient penalty}}, \tag{2.14}$$

where λ determines the intensity of the penalty and $\tilde{\boldsymbol{x}}$ is a random combination of generated data $G(\boldsymbol{z})$ and real data $\boldsymbol{x}$, defined as $\tilde{\boldsymbol{x}} = Qx + (1-Q)G(\boldsymbol{z})$ for $Q \sim U(0,1)$. For WGAN-GP, the authors have also removed batch normalization in the critic and replaced it with layer normalization. This is because batch normalization introduces correlation between samples, which invalidates the fact that the penalty applied in Eq. 2.14 is independent per sample.

2.2.4. MARKOV CHAINS

Markov chains (MC) are stochastic models that describe the occurrence of events, where the probability of an event happening depends only on the state that the event that happened before led to. An MC consists of different states, that may be visualized as nodes

within a graph. A directed connection between two states, for example from state A to state B, implies that there is a transition probability $p_{A\to B} > 0$ to transition from A to B. MCs are often characterized by their transition matrix $\boldsymbol{T} \in \mathbb{R}^{n\times n}$, where n is the number of states in the chain. In order for $\boldsymbol{T}$ to be a valid transition matrix, it has to have each entry $0 \leq t_{ij} \leq 1$ and that all rows sum to one, such that for a row i we have $\sum_{j=1}^{n} t_{ij} = 1$. Matrices that satisfy these two constraints are called stochastic matrices.

Theorem 1. *A square matrix $T \in \mathbb{R}^{n\times n}$, for $n \in \mathbb{Z}^+$, with entries t_{ij} such that $0 \leq t_{ij} \leq 1$ and a row t_i for $i = 1,...,n$ sums to one, such that $\forall i, \sum_{j=1}^{n} t_{ij} = 1$ is called a **stochastic matrix**.*

Stochastic matrices have nice properties, such that the matrix product of two stochastic matrices is still a stochastic matrix and that the weighted sum of multiple stochastic matrices, where the weights sum to one, is also a stochastic matrix. We proceed by stating these two properties and give a formal proof for them.

Theorem 2. *If given a weighted sum of stochastic matrices $T^{(w)} = \sum_{i=1}^{n} w_i T_i$ with weights $W = \{w_i \mid 0 \leq w_i \leq 1\}$, such that $\sum_{w\in W} w = 1$ and stochastic matrices $\mathcal{T} = \{T \mid T$ is a stochastic matrix$\}$, with $|W| = n$, then $T^{(w)}$ is a stochastic matrix.*

Proof. We reduce the problem to proving that a weighted sum of the same weights W of a row vector t_i of $T_i \in \mathcal{T}$, for which $\sum_j t_{ij} = 1$, such that $t_i^{(w)} = \sum_{i=1}^{|W|} w_i t_i$ still sums to 1. Hence, as follows

$$\begin{aligned}
\sum_{j=1}^{n} t_{ij}^{(w)} &= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} w_i t_i \right)_j \\
\sum_{j=1}^{n} t_{ij}^{(w)} &= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} w_i t_{ij} \right) \\
&= \sum_{i=1}^{n} \left(w_i \sum_{j=1}^{n} t_{ij} \right) \\
&= \sum_{i=1}^{n} w_i \cdot 1 \\
&= 1
\end{aligned} \tag{2.15}$$

We switch the summation since w_i is independent of j and the summations are finite. The third and last steps are reduced through $\sum_{j=1}^{n} t_{ij} = 1$ and $\sum_{i=1}^{n} w_i = 1$. Since each row of T_i sums to 1 and the weights multiply each row, we can generalize this proof for a stochastic matrix T_i. □

Theorem 3. *Given two stochastic matrices T and U, then the matrix multiplication $Q = TU$ results in a stochastic matrix as well.*

Proof. Given is, since both T and U are stochastic matrices, that $\forall i, \sum_j T_{ij} = 1$ and $\forall i, \sum_j U_{ij} = 1$. The matrix multiplication implies for a row i of Q

$$\begin{aligned}\sum_{j=1}^{n} Q_{ij} &= \sum_{j=1}^{n} (TU)_{ij} \\ &= \sum_{k=1}^{n} \left(\sum_{j=1}^{n} T_{ij} Q_{jk} \right) \\ &= \sum_{j=1}^{n} \left(T_{ij} \left(\sum_{k=1}^{n} Q_{jk} \right) \right) \\ &= \sum_{j=1}^{n} T_{ij} \cdot 1 \\ &= 1\end{aligned} \tag{2.16}$$

Again, we can make the switch of the summations in the third step since T_{ij} is not dependent on k and the summations are finite. □

Given a starting state in the chain, one can traverse the chain by following the transition probabilities at each state. This is called a random walk. In particular, for a countably infinite number of events happening and for time moving in discrete time steps we have a discrete-time MC (DTMC). In DTMCs, we often talk about a probability state $\boldsymbol{p}_n$ at time step n, where we specify an initial state $\boldsymbol{p}_0$. A nice property of DTMCs is that we may acquire the probability state $\boldsymbol{p}_k$ after k time steps, given the initial state, through $\boldsymbol{p}_k = \boldsymbol{p}_0 \boldsymbol{T}^k$ or recursively through $\boldsymbol{p}_t = \boldsymbol{p}_{t-1} \boldsymbol{T}$.

Some DTMCs have an additional nice property, called the stationary distribution. In particular, for MCs that admit to being ergodic, it is guaranteed that there exists a stationary distribution. An MC is said to be ergodic when, for a given number N, any state can be reached from any other state in N time steps or less. The stationary distribution $\boldsymbol{\pi}$ may then be defined as $\boldsymbol{\pi} = \boldsymbol{\pi} \boldsymbol{T}$. An interesting field of research is the time it takes for a chain to converge to its stationary distribution. This time is called the mixing time and we will be utilizing this concept later on as well.

3

Model

In this chapter, we shall dive into our model. We will start off with a formalization of the LOB and proceed with a transformation of the LOB to a matrix that we can feed into our model. The larger part of this chapter will be dedicated to the explanation and discussion of the mathematical model that drives the generator and enables us to extract useful features from the underlying data. Both models for generation and prediction will be discussed here as well, where the changes to the model for the predictive setting will be explained later on. We will also talk about additional loss functions for our model and the critic's architecture as well.

3.1. FORMALIZATION OF THE LOB

Before we dive into the actual model, we have to look into what defines a valid LOB. First and foremost, we start off with a formalization of the LOB. A limit order book is composed of two types of orders: bid orders and ask orders. An order is composed of a price and a volume. When a trader submits an order for a financial asset, for which no match is found in the LOB, the order gets submitted in the LOB itself. If an order with the same price already exists, the volume gets added to the volume of the price level in the LOB. Otherwise, a whole new row is inserted in the LOB. We will follow the conventions of [18] for its mathematical formulation. At a given time t, we have price vectors $\boldsymbol{P}_a(t)$ and $\boldsymbol{P}_b(t)$, respectively the prices of bid and ask orders in the LOB. In these vectors $p_a^{(1)}(t)$ and $p_b^{(1)}(t)$ respectively represent the lowest sell and highest bid orders. In general $p_a^{(\ell)}(t)$ and $p_b^{(\ell)}(t)$ represent the ask and bid orders at level ℓ. In addition to the price vectors, we also have the volume vectors $\boldsymbol{V}_a(t)$ and $\boldsymbol{V}_b(t)$, respectively representing the volumes of the ask and bid orders. Their definitions $v_a^{(\ell)}(t)$ and $v_b^{(\ell)}(t)$ follow analogously.

Because of the obvious nature of an LOB, there is no formal definition of the rules that define a valid LOB found in literature, to our knowledge, hence we look into formalizing the LOB ourselves. We do this in terms of three axioms. Two of these axioms are spatial ones, meaning these axioms are applied only on the spatial dimension of the LOB. The last axiom is a temporal and weak one, meaning it is applied to the time dimension of the LOB. In addition, this is a weak axiom, since it should be usually valid, but exceptions exist. Finally, these axioms only apply to non-crossed order books. Crossed order books are books where the bid price may lie higher than the ask price. These books occur when the market is closed and are not considered in our model. We continue by summing up our axioms:

Axiom 1 (Auto-Spatial: not crossed)**.** *Given* $p_a^{(\ell)}(t)$ *and* $p_b^{(\ell)}(t)$*, then* $\forall \ell, t \in \mathbb{N}^+$*:* $p_a^{(\ell)}(t) > p_b^{(\ell)}(t)$.

Axiom 2 (Cross-Spatial: valid levels)**.** *Given* $p_a^{(\ell)}(t)$ *and* $p_a^{(\ell+1)}(t)$*, then* $\forall \ell, t \in \mathbb{N}^+$*:* $p_a^{(\ell+1)}(t) > p_a^{(\ell)}(t)$*. This also applies for* $\forall \ell, t \in \mathbb{N}^+$*:* $p_b^{(\ell)}(t) > p_b^{(\ell+1)}(t)$*. This axiom reduces axiom 1 to* $p_a^{(1)}(t) > p_b^{(1)}(t)$*, since it takes care of the other levels.*

Axiom 3 (Cross-Temporal: bounded rates)**.** *Given* $p_a^{(\ell)}(t)$ *and* $p_a^{(\ell)}(t+1)$*, then* $\forall \ell, t$ *and time horizon* $T \in \mathbb{N}^+$*:* $\frac{1}{T}\sum_{t=1}^{T} \frac{p_a^{(\ell)}(t+1)}{p_a^{(\ell)}(t)} = 1 + \alpha_t$*, where* $\alpha_t \in [-d_\alpha, d_\alpha]$ *and* $d_\alpha \to 0$*. This also applies for* $p_b^{(\ell)}(t)$ *and* $p_b^{(\ell)}(t+1)$*:* $\frac{1}{T}\sum_{t=1}^{T} \frac{p_b^{(\ell)}(t+1)}{p_b^{(\ell)}(t)} = 1 + \beta_t$*, where* $\beta_t \in [-d_\beta, d_\beta]$ *and* $d_\beta \to 0$.

To summarize our axioms: axiom 1 refers to the fact that at each level and each time step, the ask price should be higher than the bid price, since we would have a crossed book otherwise. Axiom 2 states that the ask price at a certain level should be higher than any subsequent lower levels. It also states that the bid price should be higher than any subsequent higher levels. Axiom 2 makes sure the levels in the order book are valid levels. Combining axiom 2 with axiom 1 reduces axiom 1 to $p_a^{(1)}(t) > p_b^{(1)}(t)$, since axiom 2 takes care of the other levels. We shall, however, keep the definition of axiom 1 as is for

the sake of explicit notation. This shall also play a role in the definition of our losses in Section 3.7.

Finally, weak axiom 3 states that the average change over a time horizon T should be bounded by a small number, close to 0. This ensures that there are, on average, no large changes in the ask and bid prices. Without axiom 3, a valid order book could be defined as one where the prices are drawn from a uniform distribution $U(10^{-2}, 10^{10})$, in a way that axioms 1 and 2 apply, yet the price progression is uniformly random and therefore unrealistic. We shall give a proof of the theoretical existence of an order book that has an uniformly random price progression, but satisfies axioms 1 and 2, to clearly indicate the need for axiom 3.

Theorem 4. *Given a starting distribution $U(u, v)$, such that $u, v \in \mathbb{R}, v > u$ and $u > L$, where L is the number of levels in the order book. Furthermore, define a price state of an LOB at time t, such that $L_t = (\boldsymbol{P}_a(t), \boldsymbol{P}_b(t))$, where $\boldsymbol{P}_a(t), \boldsymbol{P}_b(t) \in \mathbb{R}^L$, then there exists an LOB progression $L = (L_1, L_2, ..., L_T)$ for which the price movements throughout the time steps are uniformly random, yet each LOB satisfies axioms 1 and 2.*

Proof. We shall show that we can construct $\boldsymbol{P}_a(t)$ and $\boldsymbol{P}_b(t)$ using an initial uniform distribution $U(u, v)$ through Algorithm 1.

Algorithm 1: Constructing L_t through uniform distributions

Input: u and v, such that $u, v \in \mathbb{R}$, $v > u$ and $u > L$
Result: $\boldsymbol{P}_a(t)$ and $\boldsymbol{P}_b(t)$ that satisfy axioms 1 and 2
$\ell \leftarrow 1$;
Sample $p_a^{(\ell)}(t) \sim U(u, v)$;
Sample $p_b^{(\ell)}(t) \sim U(u-1, p_a^{(\ell)}(t) - 1)$;
while $\ell < L$ **do**
  $\ell \leftarrow \ell + 1$;
  Sample $p_a^{(\ell)}(t) \sim U(p_a^{(\ell-1)}(t) + \ell, v + \ell + 1)$;
  Sample $p_b^{(\ell)}(t) \sim U(u - \ell - 1, p_b^{(\ell-1)}(t) - \ell)$;
end

Note that the algorithm is independent of t, as axioms 1 and 2 are as well, so we can apply this algorithm for all $L_t, t = 1, ..., T$. In fact, we are not limited to give the same inputs of u and v for each L_t, since we will still satisfy axioms 1 and 2 either way. This will introduce even more stochasticity in the price movements. Hence, the price progression is uniformly random. □

Hence, Axiom 3 ensures real-looking order books. Yet, there may be a period T for which the changes are relatively large, for example right after the market opening [21, 34], which is the reason why we define it as a weak axiom. In literature, there is often a distinction made between jump models and continuous models in models for assets. As the names suggest, a jump model will take in stochastic processes that try to model sudden jumps in the price, whereas continuous models, of which an example was given by the GBM in Eq. 1.1, have a path continuity where all prices seem closely related to each other.

Note that all these axioms are defined for prices only, since they do not apply to volume: the demand might be higher than the supply at any level in the order book (or vice-versa). In addition, v_a or v_b might suddenly increase because of a shift in the price, resulting in a sell or buy wall that was already set in place. Take for example Figure 1.1 and imagine that the volume of the ask price \$7 was ten times as large. In formal terms, the volume of the best ask price would then shift from 1 to 20. This implies that the volume might be better modeled by a model that incorporates more elements from a jump model than from a continuous one. We will not be adding additional axioms for volumes on which we will base our model, since there are no concrete behavioral properties of the volume, other than the empirically acquired ones. For example, in practice, we usually see higher levels of volume around the midprice [3], defined as

$$p_{mid}(t) = \frac{p_a^{(1)}(t) + p_b^{(1)}(t)}{2}. \tag{3.1}$$

Future work can explore the possibilities of defining additional axioms and implementing them into the model. We shall, however, leave it to the model to learn these empirical observations from the data itself.

3.2. LOB STRUCTURE

Now we got a clear formalization, we may move on to the translation of an order book to an image. Note that the transformation we will apply will actually result in a $2L \times 2T$ matrix, but we shall refer to it as the order book image for convenience, in accordance with the literature. We do lock the variables of L and T to 10 and 50 for generation, respectively, to reduce the complexity of the constructed model. Throughout the work, T is denoted in terms of 100 milliseconds. Preferably, we would have a smaller time scale, but the data available for our research does not admit a finer time scale. For prediction, L remains the same, but T becomes $T_{base} = 35$ and $T_{pred} = 15$, such that $T = T_{base} + T_{pred}$.

Before we get into the transformation, we have to substantiate our decision to transform the order book into an image, which will be processed by CNNs. Indeed, this is not the first work to feed order book data to a CNN. Earlier works, such as that of [37], have mentioned the applicability of CNNs to LOB data, based on the fact that previous research exploited the identification of volume barriers for trading decisions [43]. In addition, [44] tried to identify visual patterns in the market to predict market movements. All of these works have contributed to the fact that there exists a spatial structure in LOB data that can be exploited through deep learning, in particular through CNNs, as demonstrated by [37, 58, 13, 8]. Hence, we are convinced that we may transform the LOB into an image with good utility.

To get on to the actual transformation, we first look at what the approaches were of fellow researchers. First off, looking at the approach of [58], we see that they went for an image with dimensions $4L \times T$. In this image, they encoded, in respective order, p_a, v_b, p_a, and p_b in each column, for L levels and for T time steps. They argue that their chosen architecture for their CNN learns the micro-price, defined by

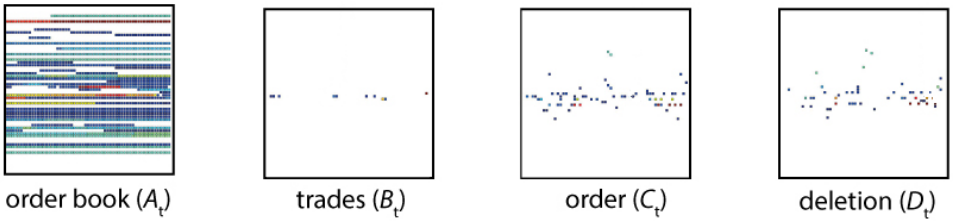

Figure 3.1: The matrices used for stock price movement prediction in [13]. Source: Doering et al. (2017)

$$p_{micro-price}(t) = I p_a^{(1)}(t) + (1-I) p_b^{(1)}(t)$$
$$I = \frac{v_b^{(1)}}{v_a^{(1)} + v_b^{(1)}}. \tag{3.2}$$

We, however, do not agree with their notion of learning the micro-price. They have implemented two convolutional layers, where each layer uses a 1×2 filter and a $(1,2)$ stride: the first layer summarizes the information between the price and volume for every (p_a, v_a) and (p_b, v_b) pair. The next layer summarizes the information of those two pairs together, hence, their feature maps contain values that are correlated with the micro-price, but not the micro-price itself: in their model, the weights I and $1-I$ in Eq. 3.2 get replaced by learnable weights w_1 and w_2, which are not necessarily equal to I and $1-I$.

In the work of [37], they disregarded the actual bid and ask prices completely, arguing that their underlying volume distributions and the number of incoming orders explain more about the price movement. Because of this, their pre-processing step might become tedious, and one might not have access to the resources to acquire the number of incoming orders for a given price. Since our work considers generating and predicting whole order books, we should take the prices into consideration in our transformation of the LOB.

Furthermore, [13] also uses the information of order arrivals to predict stock price movements using a CNN. In their case, however, they compute four different matrices: a matrix $\boldsymbol{A}_t$ for the order book state at time t, a matrix $\boldsymbol{B}_t$ for the trades that took place at time t, and two matrices $\boldsymbol{C}_t$ and $\boldsymbol{D}_t$, which represent the arrival and deletion of orders, respectively. Aside from the problem of not being able to acquire this information, a problem not mentioned in their research is that of sparse matrices. As mentioned in [28], CNNs are designed for dense input data, however, as can be seen from Figure 3.1, their matrices can become quite sparse, especially in the case of the latter three matrices. This is a direct consequence of their transformation methodology, which includes one-hot vector encoding along the columns of the input matrix.

In [8], they explore meaningful ways of transforming LOB data into a matrix that may be fed into the CNN. They do not seem to consider the volume, neither the price at various levels. Their only inputs to the pre-processing step are the open and close prices of the stock. They transform their time series data using a Gramian Angular Field (GAF)

[54] into a symmetric matrix. The transformation step applies the cosine to the summation of every possible pair in the time series. Formally, let $\boldsymbol{x} = \{x_i\}_{i=1}^{T}$ represent a time series with T data points x_i. Then, an entry $g(i,j)$ in the GAF transformed matrix would be $g(i,j) = g(j,i) = \cos(x_i + x_j)$. This transformation preserves the temporal correlations between the variables. In addition, they use two moving average mappings, which respectively use the open-close prices and the mean of the open-close prices. They compare their results to a matrix in which the candlesticks of the stock are depicted, which in its turn is quite sparse. In the long run, however, the maximum difference in accuracy between the worst performing method (candlesticks) and the best performing method (GAF) is about 2%, which implies that for a sparse matrix, the CNN was still able to achieve a similar accuracy to that of the GAF method. For a difference this small, we might even account it to a matter of random seeds set before the experiment. However, this probably has to do with the fact that the information contained with both the candlestick and GAF images remained the same. The problem for the GAF transformation then still remains: it is not quite compatible with human interpretability, while candlesticks are. A trader looking at the result of the network with a candlestick graph as its input might build trust in its decision, given its interpretable input. This is not the case for the GAF transformation.

Looking back at the methods used in literature, we may finally devise a transformation method of our own. In particular, we aim to transform the LOB in two main ways: 1) looking at the transformation, a trader should be able to understand what they are looking at, and 2) the transformation should be efficiently processable through a CNN. For the former part, we adopt a structure similar to that of [58], but slightly alter it to make it potentially more efficient for the network to compute. In particular, we model a tuple of variables (p_a, v_a, p_b, v_b) as a 2×2 matrix

$$\begin{bmatrix} p_a & v_a \\ p_b & v_b \end{bmatrix}.$$

Given L levels and T time steps, we repeat this matrix L times row-wise and T times column-wise, resulting in a $2L \times 2T$ matrix. As a result, we acquire a matrix

$$\begin{bmatrix} p_a^{(1)}(t) & v_a^{(1)}(t) & p_a^{(1)}(t+1) & v_a^{(1)}(t+1) & \dots & p_a^{(1)}(T) & v_a^{(1)}(T) \\ p_b^{(1)}(t) & v_b^{(1)}(t) & p_b^{(1)}(t+1) & v_b^{(1)}(t+1) & \dots & p_b^{(1)}(T) & v_b^{(1)}(T) \\ p_a^{(2)}(t) & v_a^{(2)}(t) & p_a^{(2)}(t+1) & v_a^{(2)}(t+1) & \dots & p_a^{(2)}(T) & v_a^{(2)}(T) \\ p_b^{(2)}(t) & v_b^{(2)}(t) & p_b^{(2)}(t+1) & v_b^{(2)}(t+1) & \dots & p_b^{(2)}(T) & v_b^{(2)}(T) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ p_a^{(L)}(t) & v_a^{(L)}(t) & p_a^{(L)}(t+1) & v_a^{(L)}(t+1) & \dots & p_a^{(L)}(T) & v_a^{(L)}(T) \\ p_b^{(L)}(t) & v_b^{(L)}(t) & p_b^{(L)}(t+1) & v_b^{(L)}(t+1) & \dots & p_b^{(L)}(T) & v_b^{(L)}(T) \end{bmatrix}.$$

By doing this, regardless of our chosen architecture for our CNN, we will be able to model cross-price correlations, cross-volume correlations, or price-volume correlations in one layer. This was not possible with the transformation of [58], where they could only model price-volume correlations, since either price or volume values would be in the way of the filter. A consequence of this chosen transformation is, given that axioms 1 and 2 apply, that we should see a growing row-wise contrast between p_a and p_b. This is better

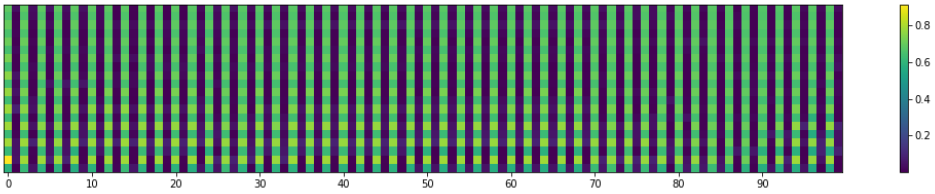

Figure 3.2: A valid collection of order books ($2L \times 2T$, $L = 10$, $T = 50$) with growing row-wise contrast for prices, depicting the growing bid-ask spread. Even columns depict the prices, whereas uneven columns depict the volumes.

depicted in Figure 3.2. Furthermore, it is suggested in [30] that there does not exist a correlation between the price and the volume of an asset, but there is proof of correlation between the price change and the volume of an asset. This is a reasonable assumption to make, since a large positive change in the price could be the result, or indication, of a transaction with a large volume, since it would draw the trader's attention. Our transformation still gives the network the opportunity to learn these correlations in the case of parties that tried to make the spread, since p_a and p_b are still within a filter's covered area. Making the spread implies that a party deploys a strategy to gain $p_a - p_b$, which is the bid-ask spread.

3.3. GENERATOR

As aforementioned, this work aims to construct an explainable model to learn the dynamics of the LOB. For the goal of generation, we use a WGAN-GP model, where we cut off a part of the generator and replace it by a mathematical model. In particular, we utilize transposed convolution layers to upsample the sampled noise to 8×8 feature maps. The way we have designed the architecture of the generator is by doubling the input dimension 1×1 up until a dimension of 8×8. We do this to preserve the correlation between all variables as much as possible. After this point, the networks splits up into two separate branches: one branch calculates the initial values of the first order book in the sequence of order books that will be generated and the other branch calculates the weights that we will be using as inputs to our model. The usage of these weights will become more apparent in Section 3.4. A schematic of the architecture is displayed in Figure 3.3, where the kernel, stride, and padding sizes are also given. All layers are initialized through the initialization [22], which means the weights and biases are sampled from $\mathcal{N}(0, \sigma^2)$, where $\sigma = \frac{\gamma}{\sqrt{f}}$. Here, γ is defined as the gain, which depends on the chosen activation function and f is the number of incoming connections. Since we will be mainly using the leaky ReLU activation function throughout our layers, we will set $\gamma = \sqrt{\frac{2}{1+c^2}} \approx 1.39$, since the slope c of our leaky ReLU is set to $c = 0.2$ [22].

After applying a transposed convolution to the 8×8 feature maps to receive $L \times 4$ feature maps, we keep reducing the features through 1×1 convolutions. We do this until we have 16 features, after which the next layer has only one feature. When applying a transposed convolution to go from 8×8 to $L \times 4$, we calculate the necessary kernel

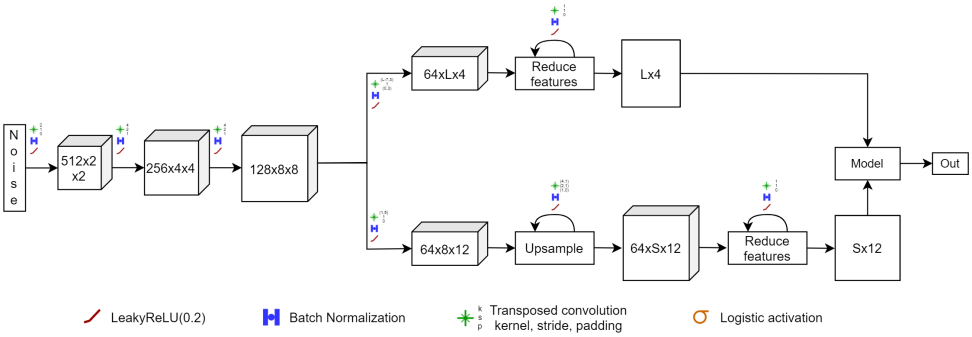

Figure 3.3: A schematic of the generator. The 3D blocks represent the matrices at various stages throughout the network, noted using $C \times W \times H$. The upper branch of the network represents the learned initial values and the lower branch of the network represents the learned weights for combining rates and transition probabilities.

size, given that the stride and padding are both fixed at 1 and 0, respectively. Applying a backwards calculation to acquire the necessary kernel size, we find that $L-7$ is the right one. As a result, we are left with initial values for the ask price p_a, bid price p_b, ask volume v_a, and the bid volume v_b, where each variable has L values that are generated. Hence, we acquire an $L \times 4$ matrix. For convenience, we utilize $x \in \{p_a, p_b, v_a, v_b\}$ in the continuation of our work and denote the initial values as $\boldsymbol{x}_0 \in \mathbb{R}^L$.

For the weights in the lower branch we first apply a transposed convolution to go from 8×8 to 8×12. This is because we will need three variables for each x: the prioritizing weights for upward movements $\boldsymbol{x}_{up}$, prioritizing weights for downward movements $\boldsymbol{x}_{down}$, and prioritizing weights for Markov chains $\boldsymbol{x}_M$. The exact usage of these weights will become clearer in Sections 3.4.1 and 3.4.2. The goal of this branch is to upsample the input matrix to a matrix of $64 \times S \times 12$, where S is a predefined value. The upsampling is done through transposed convolution layers that keep doubling the number of rows, up until the largest number of rows that is smaller than S. Inherent to the upsampling process, but not explicitly depicted in Figure 3.3, is also a final layer whose kernel size depends on S and the previously achieved number of rows, such that the output becomes exactly $64 \times S \times 12$. After that, the similar feature reduction as with the initial values is applied, through 1×1 convolutions. Finally, we are left with an $S \times 12$ matrix. This matrix will be split up into groups of three, such that we acquire the previously mentioned variables $\boldsymbol{x}_{up}, \boldsymbol{x}_{down}, \boldsymbol{x}_M \in \mathbb{R}^S$.

3.4. MATHEMATICAL FORMULATION

Before we explain what the outputs of the branches of the network are used for, we have to dive deeper into the model. In this section, we shall give a concrete mathematical formulation of the model, on which the continuation of this chapter will build. In particular, we model a time dependent variable y_t as a random variable, dependent on y_{t-1}, such that

$$y_t = y_{t-1} w_t. \tag{3.3}$$

Here, $w_t = 1 + r_t$ is the rate of change of y_t compared to y_{t-1}, where $r_t > -1$ is also a random variable. This is our second time dependent variable and it lies at the heart of our model. A larger part of the next sections will be dedicated to its calculation. In particular, we define the conditional expectation for y_t, given y_{t-1}, as

$$\begin{aligned} \mathbb{E}[y_t \mid y_{t-1}] &= \mathbb{E}[y_{t-1} \mid y_{t-1}]\mathbb{E}[w_t \mid y_{t-1}] \\ &= y_{t-1}\mathbb{E}[w_t \mid y_{t-1}] \end{aligned} \tag{3.4}$$

We set $\mathbb{E}[w_t \mid y_{t-1}] = 1 + \mathbb{E}[r_t \mid y_{t-1}]$, as the expected rate at time t and rewrite this function to

$$\mathbb{E}[y_t \mid y_{t-1}] = y_{t-1}(1 + \mathbb{E}[r_t \mid y_{t-1}]) \tag{3.5}$$

where $\mathbb{E}[r_t \mid y_{t-1}]$ is the expected rate at time step t, given y_{t-1}. Calculating this expectation requires a few ingredients, namely the probability of an up movement, a stagnation, or a down movement and the rate of an up or down movement, all given the previous value y_{t-1}. These values are learnable parameters and our work shall task itself with learning them. Note that the rate for a stagnation is already given: this is equal to one. We may now define $\mathbb{E}[r_t \mid y_{t-1}]$, using its event space $E = \{\text{up}, \text{stag}, \text{down}\}$, as

$$\mathbb{E}[r_t \mid y_{t-1}] = \sum_{e \in E} P(e_t \mid y_{t-1}) r_t^{(e)}. \tag{3.6}$$

In the following sections, we explain how we acquire $P(e_t \mid y_{t-1})$ and $r_t^{(e)}$, $e \in E$, which are the probabilities and rates, respectively, of an up movement, stagnation, and down movement.

3.4.1. PROBABILITIES

As is done in [25], we start by modeling the probabilities of an increase, decrease, and stagnation in the variable y_t as a discrete time Markov chain (DTMC), such that we can estimate the probability $P(e_t \mid y_{t-1})$ in Eq. 3.6. Although this will imply that we make the Markovian assumption over our variables, which has not been proven to apply to the targeted variables in the financial market, we shall try to keep our model as realistic as possible.

We define a learnable transition matrix for variable $x^{(\ell)}$ as $\boldsymbol{T}_{x,\ell} \in \mathbb{R}^{3\times 3}$, where each row (resp. column) represents the probability for an upward movement, stagnation and downward movement. If we denote movements $a, b \in E$, then an entry $\boldsymbol{T}_{x,\ell}(a,b)$ in the transition matrix means the probability of movement b occurring when movement a had occurred previously. Furthermore, we define a probability state at time t for variable $x^{(\ell)}$ at level ℓ as $\boldsymbol{p}_t^{(x,\ell)} \in \mathbb{R}^3$. The entries of the probability state indicate the probability of having a movement $e \in E$ at time step t at level ℓ in the order book.

Since $\boldsymbol{T}_{x,\ell}$ is a transition matrix and therefore a stochastic matrix, we should have that $\forall i \sum_j \boldsymbol{T}_{x,\ell}(i,j) = 1$. Learning a transition matrix with this constraint in mind through backpropagation can be a bit tricky, but we have a way around it. We initialize our 'transition matrices' from a uniform distribution and transform them into real transition matrices through applying the softmax function on each row of the transition matrix when

we need to utilize the matrix for calculations. The model itself learns the inputs to the softmax function. We define a step in the chain as the inner product between the probability state and the transition matrix:

$$\boldsymbol{p}_t^{(x,\ell)} = \boldsymbol{p}_{t-1}^{(x,\ell)} \boldsymbol{T}_{x,\ell}, \tag{3.7}$$

where we define

$$\boldsymbol{p}_0^{(x,\ell)} = \left[\tfrac{1}{3} \quad \tfrac{1}{3} \quad \tfrac{1}{3}\right], \tag{3.8}$$

since we do not know the previous state, assuming that all movements could have happened. This is our basic model for probabilities. There are, however, several problems with this, which we will address.

First, the model will not be able to generalize to different regions in the underlying data that exhibit a different variable movement. Take for example the price movement: right after the market opening, the price might fluctuate a lot more and the probability state might be more uniformly distributed at any given time t, or even the inverse: extremely skewed. At other times, the probability state might lie in between those two extremes. To account for the different distributions, we introduce another variable S, which denotes the estimated number of different distributions in the underlying data. In particular, we have S transition matrices per level of the order book and combine them into one transition matrix by taking a weighted average of all S transition matrices for each level, acquiring a set $\boldsymbol{T}_{x,\ell}^S = \left\{ \boldsymbol{T}_{x,\ell}^{(i)} \right\}_{i=1}^S$. These weights are given by $\sigma(\boldsymbol{x}_M)$, which is of dimension $\mathbb{R}^L$ and is learned in the lower branch shown in Figure 3.3, where $\sigma(\cdot)$ is the softmax function, such that the elements of the vector sum to one.

Even though the introduction of the S variable seems interesting, we have to draw some attention to this variable. Let S^* denote the real number of different probability distributions: if $S < S^*$, the network will learn to cluster probability distributions together, resulting in an underfit, following the pigeon hole principle; if $S > S^*$, however, the network will learn to distribute the same distribution over multiple matrices, resulting in an overfit. We calculate the weighted transition matrix as follows

$$\boldsymbol{T}_{x,\ell}^{(w)} = \boldsymbol{T}_{x,\ell}^{weighted} = \sum_{i=1}^{S} \sigma(\boldsymbol{x}_M)_i \boldsymbol{T}_{x,\ell}^{(i)}. \tag{3.9}$$

Note that since all rows of all transition matrices sum to one and the weights do too, the result will also be a valid transition matrix which sums to one. Also, since all values of the transition matrix are nonzero, we have a fully connected chain, resulting in an ergodic Markov chain. For ergodic Markov chains, there exists a stationary distribution $\boldsymbol{\pi}_t^{(x,\ell)}$ such that

$$\lim_{t\to\infty} \boldsymbol{\pi}_t^{(x,\ell)} = \boldsymbol{\pi}_t^{(x,\ell)} \boldsymbol{T}_{x,\ell}^{weighted}. \tag{3.10}$$

This is the second problem with the model. Let us draw some attention to the t variable, in particular, we are interested in the number of time steps it takes for the chain to settle on the stationary distribution. In literature, this is often referred to as the mixing time, denoted as t_{mix}. In our case, we want $t_{mix} > T$, such that the chain does not

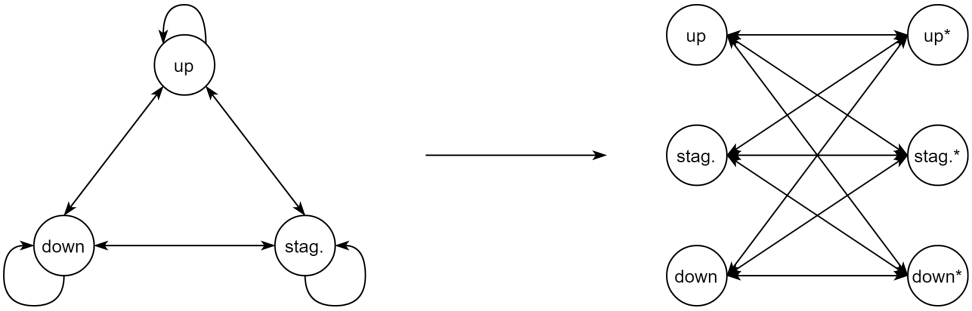

Figure 3.4: Translating a fully connected graph with three states to a fully connected bipartite graph with six states.

converge to its stationary distribution before the last order book entry is calculated in our $2L \times 2T$ matrix. If this constraint is not satisfied, then the chain will give the same transition probabilities for all $t \geq t_{mix} \leq T$. Having $t_{mix} = T$ will result in a stationary distribution on T, where the initial vector $\boldsymbol{p}_0^{x,\ell}$ would not matter anymore [46]. The stationary distribution is in fact only dependent on T. In particular, we want the model to reflect the uncertainty of the prediction mechanism as $t \to T$, since we should be able to predict more accurately for $t = 0$ than for $t = T$. We continue with a proof on the bounds of t_{mix} and thereafter discuss an alternative model for the probabilities.

Theorem 5. *A discrete time Markov chain with transition matrix $\boldsymbol{T} \in \mathbb{R}^{3\times3}$, with an entry t_{ij} in the matrix $\forall i, j\ t_{ij} > 0$ and $\forall i \sum_{j=1}^{3} t_{ij} = 1$ has a mixing time $2 \leq t_{mix} \leq 35$.*

Proof. Given a transition matrix $\boldsymbol{T} \in \mathbb{R}^{3\times3}$, such that $\forall i, j\ t_{ij} > 0$ and $\forall i \sum_{j=1}^{3} t_{ij} = 1$, and a random walk with probability state $\boldsymbol{p}_t \in \mathbb{R}^3$ on time step t whose entries sum to 1 as well, we shall prove the bounds on the mixing time t_{mix} for the random walk $\boldsymbol{p}_{t+1} = \boldsymbol{p}_t \boldsymbol{T}$ and arbitrary $\boldsymbol{p}_0 \in \mathbb{R}^3$. As suggested in [9], the mixing time depends on the second largest eigenvalue modulus (SLEM) of $\boldsymbol{T}$. Given an eigenvalue ordering of $\boldsymbol{T}$ as $1 = \lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_{n-1} \geq -1$, the SLEM of $\boldsymbol{T}$ is defined by

$$\mu = \max_{i=2,\ldots,n} |\lambda_i| \tag{3.11}$$

We shall proceed by transforming $\boldsymbol{T}$ into a chain for which the SLEM is already known in literature. Since our chain is a fully connected one, along with self-loops, we may transform our chain into a complete bipartite graph, having equal number of states in both classes, hence $n = m = 3$, as shown in Figure 3.4. The transition matrix $\boldsymbol{T}^*$ for the bipartite graph then becomes

$$\boldsymbol{T}^* = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T} \\ \boldsymbol{T} & \boldsymbol{0} \end{bmatrix}, \tag{3.12}$$

where $\boldsymbol{0}$ denotes the 3×3 matrix with all zeroes. We continue our proof by proving that the set of absolute eigenvalues of $\boldsymbol{T}^*$ are equal to the set of absolute eigenvalues of $\boldsymbol{T}^*$. We do this by looking at the characteristic equations of both. If we set

$$|\boldsymbol{T} - \lambda \boldsymbol{I}| = 0 \tag{3.13}$$

to be the characteristic equation of $\boldsymbol{T}$ and use the property of block matrices from [49] to calculate the characteristic equation of $\boldsymbol{T}^*$

$$|\boldsymbol{T}^* - \lambda I| = -|\boldsymbol{T} + \lambda I||\boldsymbol{T} - \lambda I| = 0, \tag{3.14}$$

we can see that the eigenvalues of $\boldsymbol{T}^*$ contain both the eigenvalues and negative eigenvalues of $\boldsymbol{T}$, hence the absolute eigenvalues of both matrices are the same. In this sense, the SLEM as defined in Eq. 3.11, is also the same for both matrices. We can now use [7] to acquire the lowest possible SLEM for $\boldsymbol{T}^*$ to be $\mu = \frac{n}{n+2m} = \frac{1}{3}$. We may now use [9] to determine a lower bound for t_{mix}, resulting in $t_{mix} = \frac{1}{\log \mu^{-1}} \approx 2$. For the upper bound of t_{mix} we look at [29]. They give an upper bound for the (total variation) mixing time of a transition matrix $\boldsymbol{T}$

$$t_{mix}(\epsilon) = \arg\min_t ||\boldsymbol{\pi} - \boldsymbol{p}_0 \boldsymbol{T}||_{TV} < \epsilon, \tag{3.15}$$

where $\boldsymbol{\pi}$ is the stationary distribution of the chain. They define the total variation distance between two probability measures κ and ν on a state space $\mathscr{X}$ as

$$||\kappa - \nu||_{TV} = \frac{1}{2} \sum_{x \in \mathscr{X}} |\kappa(x) - \nu(x)|. \tag{3.16}$$

As a result, they define an upper bound on the total variation mixing time of the chain for $0 < \epsilon < 1$ to be equal to

$$t_{mix}(\epsilon) \leq 2N t_{rel} \log t_{rel} + 4(1 + \log 2) N t_{rel} + 2(\log \epsilon^{-1} - 1) t_{rel}, \tag{3.17}$$

where $t_{rel} = \frac{1}{1-\mu}$ and $N = 3$ in our case. As commonly chosen in literature for the total variation distance, we set $\epsilon = \frac{1}{4}$. Filling in Eq. 3.17 results in $t_{mix}(\frac{1}{4}) \leq 35$. Hence, we acquire lower and upper bounds for t_{mix} to be $2 \leq t_{mix} \leq 35$. □

Hence, the constraint on $t_{mix} > T$ is not satisfied for $T > 2$. However, we do aim to have $T = 50$, so this approach will not work.

Instead, we explore the possibilities of utilizing the stationary distributions of the Markov chains, since we still want to utilize Markov chains to learn the transition matrices and to estimate the conditional probabilities. In particular, we seek to do a random walk on the stationary distributions: given that $t_{mix} < T$, we can consider the stationary distributions as the 'characteristic' distributions of the transition matrices. We can manually shape this random walk to become more stochastic as $t \to T$. The stationary distributions are independent of $\boldsymbol{p}_0$, so they show the limiting behaviour of the variable on a given level. If we take this limiting behaviour to be the average behaviour of the variable on that level, since $T > t_{mix}$, then we can do a random walk through various stationary distributions. However, we will also aim to model cross-level interactions, such that the probabilities of a movement on a given level are also considered in the probabilities of another level. To model this interaction, we calculate a set $T_x = \left\{ \boldsymbol{T}^{(w)}_{x,i} \boldsymbol{T}^{(w)}_{x,j} \right\}_{i,j=1}^{L}$, which

contains the matrix products of all pairs of weighted transition matrices. This step considers both $p_u p_v$ and $p_v p_u$ movements, for $u, v \in E$ when calculating the matrix product, hence is able to model cross-level interactions that way. Our next step is to calculate the stationary distributions of these matrices. This calculation is done through finding the eigenvector that corresponds to the eigenvalue of 1, as denoted by Eq. 3.10. This results in a set of stationary distributions

$$\Pi_x = \left\{ \boldsymbol{\pi}_{ij}^{(x)} \;\middle|\; \boldsymbol{\pi}_{ij}^{(x)} \boldsymbol{T}_{x,ij}^{(w)} = \boldsymbol{\pi}_{ij}^{(x)} \text{ for } \boldsymbol{T}_{x,ij}^{(w)} \in T_x \right\}_{i,j=1}^{L},$$

which we refer to as the sampling set. A basic random walk through these distributions for a variable x on level ℓ on time step t is then a walk

$$\boldsymbol{\pi}_{x,\ell}^{(t)} = \Psi(\Pi_x, \ell, t), \tag{3.18}$$

where $\Psi(\Pi_x, \ell, t)$ is a function $\Psi : \Pi_x, \mathbb{Z}^+, \mathbb{Z}^+ \to \Pi_x$ that samples a stationary distribution from the sampling set, hence the name. An advantage of this method is that we may specify any function Ψ to sample a stationary distribution. This function may choose distributions uniformly to depict a more cross-level interacting focused walk, which is quite stochastic and might result in a more stochastic training period. However, one might also lay focus on $\boldsymbol{\pi}_{\ell j}$, and sample j randomly, in which case only the interactions of the current level are considered. An interesting instance arises when we consider a combination of the two methods, where we usually select $\boldsymbol{\pi}_{\ell j}$ for the variable at level ℓ, but also consider $\boldsymbol{\pi}_{ij}, i \neq \ell$ to take other cross-level interactions into account as well. Hence, there is much left to experiment with.

In fact, Ψ is not limited to sample from Π_x. It might also take in an additional learnable parameter $\theta_\Pi = (w_1, ..., w_{|\Pi_x|})$, $w_k \in \mathbb{R}$ to weigh stationary distributions and combine them through

$$\Psi(\Pi_x, \ell, t, \theta_\Pi) = \sum_{\substack{\pi \in \Pi_x \\ w \in \theta_\Pi}} w\pi. \tag{3.19}$$

The learnable parameter θ_Π can, for example, be acquired through an additional network branch in the generator that outputs the required number of weights. This way, the sampling function becomes differentiable and the network will be able to learn to weigh certain stationary distributions more than others. However, as long as no thorough analysis is executed, it will be unclear why the network outputs the weights it does. This is in fact the case with the branches calculating the initial values and the values of $\boldsymbol{x}_{up}$, $\boldsymbol{x}_{down}$, $\boldsymbol{x}_M$. However, as opposed to these branches, the sampling function offers flexibility in its definition, so we can offer up some of our model's ability of learning in return for more control and insight about the model's functioning.

Furthermore, note that for $T < |\Pi_x| = L^2$ it is impossible to consider all stationary distributions in the random walk, since t it will only go up to $t = T$. A chosen sampling method should consider this constraint as well. Also, depending on the sampling method, the random walk might become Markovian: one could let $\boldsymbol{\pi}_{x,\ell}^{(t)}$ depend on $\boldsymbol{\pi}_{x,\ell}^{(t-1)}$, for example through calculating the inner product between $\boldsymbol{\pi}_{x,\ell}^{(t-1)}$ and $\boldsymbol{\pi}_{x,\ell}^{(t)}$. Depending on the sampling method, one could let the process be periodic or even ergodic.

The stochastic properties of $\boldsymbol{\pi}_{x,\ell}^{(t)}$ depend on the sampling function, so the expectation and variance are derived through Eq. 3.18. However, we shall not extensively explore its properties for a multitude of sampling functions in this work, since they differ depending on the chosen sampling function. Note that the sampling function can be used for a purpose other than achieving better results: it might also serve as another source for feature extraction, where we will be able to learn more about the underlying data set through variables that can be defined within the function.

For this work, we shall only consider the case where we usually select $\pi_{\ell j}$, but also consider $\pi_{ij}, i \neq \ell$ as well. Hence, the sampling function that follows is

$$\Psi(\Pi_x, \ell, t) = \begin{cases} \boldsymbol{\pi}_{\ell j}, & \text{with probability } p \\ \boldsymbol{\pi}_{ij}, & i \neq \ell \text{ with probability } 1-p. \end{cases} \tag{3.20}$$

The exact implications of this sampling function are that we shall usually consider the stationary distributions of the chains that capture the interactions between levels ℓ and $i = 1, ..., L$ and also consider the stationary distributions of the chains that capture all other interactions.

For this sampling function, we can define its expectation, where we shall use a shorthand notation of $\Psi = \Psi(\Pi_x, \ell, t)$, as

$$\mathbb{E}[\Psi] = \frac{p}{L} \sum_{j=1}^{L} \boldsymbol{\pi}_{\ell j} + \frac{1-p}{(L-1)L} \sum_{i \neq \ell} \sum_{j=1}^{L} \boldsymbol{\pi}_{ij}, \tag{3.21}$$

since there are L elements of each level and there is an equal probability to sample from the same level, resulting in $\frac{p}{L}$ for $i = \ell$ and in $\frac{1-p}{L(L-1)}$ for $i \neq \ell$. Furthermore, we can define its variance as

$$\text{Var}[\Psi] = \frac{p}{L} \sum_{j=1}^{L} (\boldsymbol{\pi}_{\ell j} - \mathbb{E}[\Psi])^2 + \frac{1-p}{(L-1)L} \sum_{i \neq \ell} \sum_{j=1}^{L} (\boldsymbol{\pi}_{ij} - \mathbb{E}[\Psi])^2. \tag{3.22}$$

To make things even more concrete, we aim to sample $\boldsymbol{\pi}_{\ell j}$ ξ times more often than $\boldsymbol{\pi}_{ij}, i \neq \ell$. This means that p has to be ξ times larger than $1-p$, hence

$$\begin{aligned} p &= \xi(1-p) \\ p &= \xi - \xi p \\ p + \xi p &= \xi \\ p(1+\xi) &= \xi \\ p &= \frac{\xi}{1+\xi}. \end{aligned} \tag{3.23}$$

The variable ξ will be referred to as the sampling rate further on in this work. This sampling function is a good example of how we can utilize it from the perspective of feature extraction, as will be explicitly demonstrated in the sensitivity analysis in Section 5.3.1. The sampling rate can be used as a measure of how much a variable is correlated with the same variable on other levels within the order book: a lower sampling rate indicates a higher dependency on other levels and a higher sampling rate indicates a higher

dependency on the variable's own level. To actually see what the interaction between the probabilities of the levels within the order books in the data set is, we would have to look at whichever sampling rate would yield the best results. Note that we cannot set ξ to be a learnable variable, since the sampling function is not differentiable, but the sampled stationary distributions are differentiable, so we can learn the transition matrices that form these distributions.

As can be seen from Eq. 3.18, the specified random walk is quite stochastic, independent of t. To solve this issue, we introduce an exponential weight, scaled on t, such that we can increase the stochasticity of the walk as $t \to T$. This gives rise to

$$\boldsymbol{\pi}_{x,\ell}^{(t)} = \mu_\pi + (1-\gamma^t)(\Psi(\Pi_x, \ell, t) - \mu_\pi), \tag{3.24}$$

where $\mu_\pi = \frac{1}{N} e$ is the uniform distribution for $N(=3)$ states, e is the vector of all ones and γ^t is the exponential weight, which, for a constant $0 < \gamma < 1$, becomes smaller as $t \to T$. This way we are able to construct a model that is still able to capture cross-level interactions, and becomes more stochastic as $t \to T$. Note that for $t = 0$, $\boldsymbol{\pi}_{x,\ell}^{(t)} = \mu_\pi = \boldsymbol{p}_0$, just as in a random walk in the Markov chain, and that the sum over all entries in $\boldsymbol{\pi}_{x,\ell}^{(t)}$ is still 1, since Eq. 3.24 is a linear transformation on $\boldsymbol{\pi}_{ij} \in \Pi_x$, and the elements of $\boldsymbol{\pi}_{ij}$ already add up to one.

To visualize how the random walk looks for 10 initial transition matrices and different values for γ, we take a look at Figure 3.5, where we plot the walk from Eq. 3.24 with a uniform sampling function, instead of the function proposed in Eq. 3.20, for $T = 50$. The transition matrices are randomly sampled from a uniform distribution and the softmax function is applied to their rows. Indeed, we see that increasing γ leads up to less stochasticity early on and more as $t \to T$.

The proposed walk is an alternative to the random walk over the Markov chain. Through this walk, we are still able to learn the transition matrices of the variables at each level, which are representative of the data and can be used for later analysis. In addition, we omit the problem of the chains converging too fast, which would result in no variance in the probability states at each time step. Along with all these advantages, it also comes with flexibility in the chosen sampling method and retains the initial probability state $\boldsymbol{p}_0$ of the random walk on the Markov chain.

3.4.2. RATES

Learning the rates is actually simpler than learning the probabilities. We introduce two rate matrices, namely $\boldsymbol{R}_x^{up} \in \mathbb{R}^{L\times T}$ and $\boldsymbol{R}_x^{down} \in \mathbb{R}^{L\times T}$, respectively representing the rates of the variable increasing and decreasing for each time step at each level. One important aspect that we have to take into account has to do with axiom 3, which involves that the rates revolve around 1 and should not have too large deviations from it. We may enforce this constraint by putting the ratio matrices through the logistic activation function, defined by $\sigma(x) = \frac{1}{1+e^{-x}}$, such that the new matrices become $1 + \sigma(\boldsymbol{R}_x^{up})$ and $1 - \sigma(\boldsymbol{R}_x^{down})$. This way, we will always have up rates bounded in $[1, 2]$ and down rates bounded in $[0, 1]$. This still enables sudden jumps in the variables, so this way we still incorporate a jump model in our model. For ease of notation, however, we shall omit the notation of $\sigma(\cdot)$ when we talk about rates. Since the rates have to be put through the sigmoid function,

3

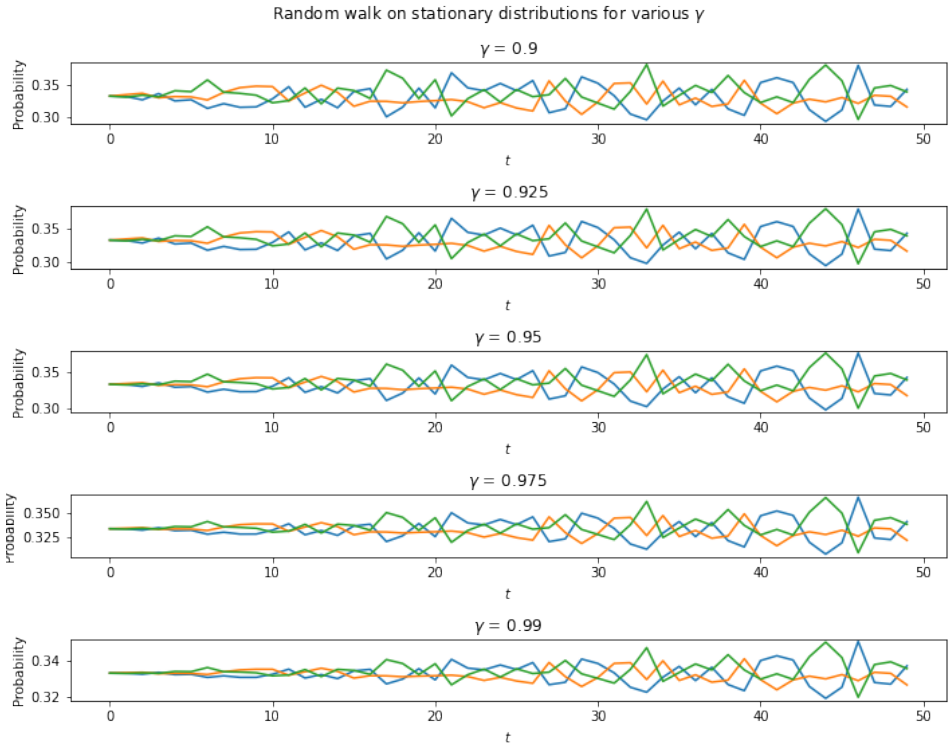

Figure 3.5: A random walk on the stationary distributions of ten different transition matrices for different values of γ. Up movements are depicted in green, down movements in orange and stagnations in blue.

we initialize them through a $U(-4,4)$ distribution, such that we acquire an initial range wide enough to just disclose smaller gradients.

Continuing on, we run into a similar issue as with the probabilities. The learned rates could only provide valid rates for one particular distribution, such as one where there are as intense upward movements as downward movements. However, there are of course scenarios where these two rates are not in balance, hence we should take that into account as well. We do this by introducing the same term S as with the probabilities. S carries the same concept for the rates as for the probabilities: the estimated number of different rate distributions. We define rate sets $R_x^{up,S} = \left\{\boldsymbol{R}_{x,i}^{up}\right\}_{i=1}^{S}$ and $R_x^{down,S} = \left\{\boldsymbol{R}_{x,i}^{down}\right\}_{i=1}^{S}$. Using vectors $\boldsymbol{x}_{up} \in \mathbb{R}^S$ and $\boldsymbol{x}_{down} \in \mathbb{R}^S$ we calculate the weighted averages of the rates in a similar fashion to that of Eq. 3.9.

3.4.3. CALCULATING THE EXPECTATION

We now have acquired the required ingredients to calculate $\mathbb{E}[r_t \mid y_{t-1}]$, as defined by Eq. 3.6. The probabilities $P(e_t \mid y_{t-1})$ at time t are obtained from the random walk of Eq. 3.24, and the rates $r_t^{(e)}$ are obtained through the learned rate matrices. Note that we can now also calculate

$$\mathbb{E}[y_t \mid y_{t-1}] = y_{t-1}(1 + \mathbb{E}[r_t \mid y_{t-1}]).$$

One last missing term is y_0. This has to do with $\boldsymbol{x}_0$, one of the outputs of the generator. We might simply take this output as y_0, however, the problem with this approach is that the model will have to learn to generate its initial values in the network itself and learn to predict the future values through the methodology described above. However, since we will be using gradient descent for learning, we want the gradients of the rates and probabilities to move in correlation with each other, which is most easily accomplished by computing $\mathbb{E}[y_0 \mid \boldsymbol{x}_0]$ in a similar fashion as mentioned before. If we simply set $y_0 = \boldsymbol{x}_0$, then the gradient descent step would only update the convolutional layers, instead of $r_1^{(\ell)}$ that is contained within the rate matrices, to accommodate for $\boldsymbol{x}_0$.

3.4.4. COMPUTATIONAL CONSIDERATIONS

Now we have the required formulas and ingredients, we need to provide a way to efficiently compute all of these values. We propose a way of embedding these formulas into matrices and calculate all expectations at once using matrix multiplication. Through this transformation we can calculate all values on our graphical processor instead of using for-loops. Our first step is to embed $\boldsymbol{x}_0$ into a diagonal matrix $\boldsymbol{X}_0 \in \mathbb{R}^{L \times L}$:

$$\boldsymbol{X}_0 = \begin{bmatrix} \boldsymbol{x}_0^{(0)} & 0 & 0 & \dots & 0 \\ 0 & \boldsymbol{x}_0^{(1)} & 0 & \dots & 0 \\ 0 & 0 & \boldsymbol{x}_0^{(2)} & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{x}_0^{(L)} \end{bmatrix}.$$

The next step is to construct an update matrix $\boldsymbol{U} \in \mathbb{R}^{L \times T}$. This matrix will contain the rates $1 + \mathbb{E}[r_t]$. Each row will represent the rates for that level of the order book. However,

since Eq. 3.5 is defined recursively, we cannot simply put the rates in their designated spots. For an entry $u(\ell, t)$ in matrix $\boldsymbol{U}$, the rates are defined by the product

$$u(\ell, t) = \prod_{k=1}^{t} 1 + \mathbb{E}[r_t^{(\ell)}]. \tag{3.25}$$

We may now define update matrix $\boldsymbol{U}$ as

$$\boldsymbol{U} = \begin{bmatrix} 1+\mathbb{E}[r_1^{(1)}] & (1+\mathbb{E}[r_1^{(1)}])(1+\mathbb{E}[r_2^{(1)}]) & \dots & \prod_{k=1}^{T} 1+\mathbb{E}[r_t^{(1)}] \\ 1+\mathbb{E}[r_1^{(2)}] & (1+\mathbb{E}[r_1^{(2)}])(1+\mathbb{E}[r_2^{(2)}]) & \dots & \prod_{k=1}^{T} 1+\mathbb{E}[r_t^{(2)}] \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 1+\mathbb{E}[r_1^{(L)}] & (1+\mathbb{E}[r_1^{(L)}])(1+\mathbb{E}[r_2^{(L)}]) & \dots & \prod_{k=1}^{T} 1+\mathbb{E}[r_t^{(L)}] \end{bmatrix}.$$

Finally, we may acquire all the values for the variable by calculating the inner product

$$\boldsymbol{Y} = \boldsymbol{X}_0 \boldsymbol{U}. \tag{3.26}$$

Note that we have to apply this process for all of our four variables p_a, p_b, v_a and v_b. For one variable, this step executes in $\mathcal{O}(L^2 T)$. Furthermore, what we did not discuss, is the computation of the eigenvectors for 3.18. Our transition matrix has dimensions $q \times q$ for $q = 3$, but we calculate the pairwise products, which in itself is $\mathcal{O}(L^2 q^3)$. Note that we set the dimension of our transition matrices to be a variable since they can vary for work on spatio-temporal time series other than order books. In addition, the matrices have already been weighed through Eq. 3.9, so we do not have to consider the variable S. We thereafter compute the eigenvectors for L^2 matrices, which is $\mathcal{O}(L^2 q^3)$ as well, but faster computations exist [51]. However, since the former operation takes as much time, we do not theoretically consider the faster computation. Depending on our value for T, assuming q is fixed, the time complexity of the model is defined as $\mathcal{O}(L^2 T)$ for $T > q^3 = 27$ and defined as $\mathcal{O}(L^2 q^3)$ for $T < q^3 = 27$.

Looking at the space complexity of the model, we have to keep track of all probability states for every level, at every time step, resulting in $\mathcal{O}(qLT)$ for the probabilities. However, we also need to store the $S \times L \times q \times q$ transition matrices, resulting in $\mathcal{O}(q^2 SL)$. We also have the matrices that keep track of the rates, which have dimensions $S \times L \times T$. All other matrices used in the process are either equal in dimensions to the aforementioned ones or smaller. Hence, the space complexity of the model depends on a few variables. Again, assuming that q is fixed and $S > q$, we can say that the space complexity is $\mathcal{O}(SLT)$ if $T > q^2$ and $\mathcal{O}(q^2 SL)$ if $T < q^2$, while the former will usually be the case.

3.5. PREDICTION

So far, we have talked about the mathematical model. This exact model is used for generative purposes, assuming that the input is a randomly initialized vector from some specified prior distribution. For the goal of prediction, however, we can supply more information to the model aside from just noise. We formulate the problem as follows: given a matrix of order books of size $2L \times 2T_{base}$, how can we predict the next T_{pred} number of

order books, such that the prediction becomes a matrix of $2L \times 2T_{pred}$, representing the continuation of the input.

Since we use mathematically tractable information, such as the rates and probabilities, we can calculate these values from the input and use it as prior information for the model. Before going into the calculation and the operations we apply to these values, we first show how we transform the first couple of layers of the generator. Since the generator will not receive random noise as input anymore, but an actual order book, the generator gets a predictive role, so we shall refer to the new generator as a predictive generator (PG).

Given an input matrix of dimensions $2L \times 2T_{base}$, we apply three convolutional layers, all including batch normalization and the Leaky ReLU activation function. The first layer consists of 2×2 filters to summarize the information of the prices and volumes, along with a stride of 2. The next layer consists of 1×2 filters to capture time dependent information, along with a stride of $(1,2)$. The final layer consists of a 3×2 filter to capture spatio-temporal information with a stride of $(1,2)$. Finally, we arrive at 8×8 feature maps. The model is continued as mentioned in Section 3.3, except for the fact that we do not need the branch for the initial values anymore, since we can read those from the last entries in the input.

We continue by going through the calculations and operations we apply to the input data to adapt the model to a predictive setting. We note that the model utilizes three learnable matrices for each variable: $\boldsymbol{R}^{up}$, $\boldsymbol{R}^{down}$ and T, representing the up rates, down rates and the transition probabilities. These are all values that we can infer from the input data, but there are a few hindrances.

First off, we can only extract rates for each time step, regardless of the rates representing up or down movements. We clearly distinguish between up and down rates in the generative model, such that each time step has an up and down ratio, weighed by their respective probabilities. Now, however, we may only obtain a ratio matrix $\boldsymbol{R}_{hist} \in \mathbb{R}^{L \times (T_{base}-1)}$, representing all rates in the order book. To obtain both up and down rates, we will make a uniform assumption that there is an indifference between the intensity of up and down rates. In particular, given a historical up ratio $r_t^{up} > 1$ for an arbitrary time step t, its counterpart, being r_t^{down} shall be calculated as $r_t^{down} = 2 - r_t^{up}$, such that the intensity of the movement would have been the same for a down movement. The same rationale applies to a historical down ratio $r_t^{down} < 1$: its counterpart r_t^{up} is calculated as $r_t^{up} = 2 - r_t^{down}$. This method finally gives us two matrices $\boldsymbol{R}_{hist}^{up}$ and $\boldsymbol{R}_{hist}^{down}$, both having dimensions of $L \times (T_{base} - 1)$.

Note that we need these ratio matrices to have dimensions of $L \times T_{pred}$, since they will be used for prediction. Towards this end, we utilize convolutional layers to summarize temporal information. There are no strict limitations bound to these layers, apart from the fact that they should output an $L \times T_{pred}$ matrix. Being able to design this network however we want is a strong advantage of this methodology, since we are able to specify which information the network should use to learn future weights. In our case, we take this network to consist of only one layer, having 1×5 filters with a stride of $(1,2)$, such that the information of every 5 time steps is summarized, followed by a batch normalization layer and a Leaky ReLU activation function. Note that the input for this layer has to be normalized as well, so we feed the matrices $\boldsymbol{R}_{hist}^{up} - 1$ and $\boldsymbol{R}_{hist}^{down} - 1$ into the layers. Since

both up and down movements might have different continuations after T_{base}, we also assign different layers to both of them.

The predictive model now is not a learnable model involving $\boldsymbol{R}^{up}$ and $\boldsymbol{R}^{down}$, however, we can change that by adding those matrices to the outputs of the convolutional layers. This operation acts as a learnable bias term that compensates for the errors made in the prediction of future rates. Finally, recall that we do not utilize a single learnable ratio matrix, but an S number of matrices, hence this same operation is repeated for all $\boldsymbol{R}_i^{up} \in R^{up,S}$ and $\boldsymbol{R}_i^{down} \in R^{down,S}$. To summarize, given the calculated historical up and down ratio matrices $\boldsymbol{R}_{hist}^{up}$ and $\boldsymbol{R}_{hist}^{down}$, we acquire the matrices $\boldsymbol{R}_{pred}^{up}$ and $\boldsymbol{R}_{pred}^{down}$ that will be used in the model through

$$\boldsymbol{R}_{i,pred}^{up} = \text{ConvNet}(\boldsymbol{R}_{hist}^{up} - 1) + R_i^{up} \tag{3.27}$$

and

$$\boldsymbol{R}_{i,pred}^{down} = \text{ConvNet}(\boldsymbol{R}_{hist}^{down} - 1) + R_i^{down}. \tag{3.28}$$

Through these operations, we acquire the new ratio sets for prediction $\boldsymbol{R}_{pred}^{up,S} = \left\{\boldsymbol{R}_{i,pred}^{up}\right\}_{i=1}^{S}$ and $\boldsymbol{R}_{pred}^{down,S} = \left\{\boldsymbol{R}_{i,pred}^{down}\right\}_{i=1}^{S}$.

Aside from the rates, we can also use information about the transition probabilities within the input data. In particular, we can construct a historical transition matrix $T_{hist}^{(\ell)} \in \mathbb{R}^{3\times3}$ for each level ℓ by counting the number of combined movements. Given that a movement m_t at time step t can either be $m_t \in \{\text{up}, \text{stag}, \text{down}\}$, we calculate the frequency of m_t followed by m_{t+1} for all possible pairs. Through these frequencies, we can construct the empirical transition matrix $\boldsymbol{T}_\ell$, of which its rows sum to one. Since the dimensions of the transition matrices are spatio-temporal independent, we do not have to put in additional efforts to resize these matrices. However, we still need to make this step learnable since we are predicting forward, as the given transition matrices might not accurately reflect the transition matrices needed for prediction. Towards this end, we utilize the learnable matrix $\boldsymbol{T}_\ell$. In addition, recall that we do not learn a single transition matrix, but S ones. Therefore, we calculate the matrix product between $\boldsymbol{T}_{hist}^{(\ell)}$ and $\boldsymbol{T}_\ell^{(i)} \in \boldsymbol{T}_\ell^S$, such that we acquire $\boldsymbol{T}_{pred,i}^{(\ell)} = \boldsymbol{T}_{hist}^{(\ell)} \boldsymbol{T}_\ell^{(i)}$. Note that since both $\boldsymbol{T}_{hist}^{(\ell)}$ and $\boldsymbol{T}_\ell^{(i)}$ are stochastic matrices, their product will also result in a stochastic matrix. We then acquire a new set of transition matrices for prediction, defined as $\boldsymbol{T}_{pred,S}^{(\ell)} = \left\{\boldsymbol{T}_{pred,i}^{(\ell)}\right\}_{i=1}^{S}$. This way, the model stays learnable while utilizing extracted information from the input data. A complete schematic of the generator in the predictive setting is displayed in Figure 3.6.

3.6. CRITIC

Since we are using a WGAN-GP, the discriminator gets renamed to 'critic', since it does not output a probability anymore. Rather, it outputs a scalar value, representing the realness of the input. The critic gets an order book of size $2L \times 2T$ as its input and consists of a combination of convolutional layers. Initially, we apply a 2×2 kernel over the order

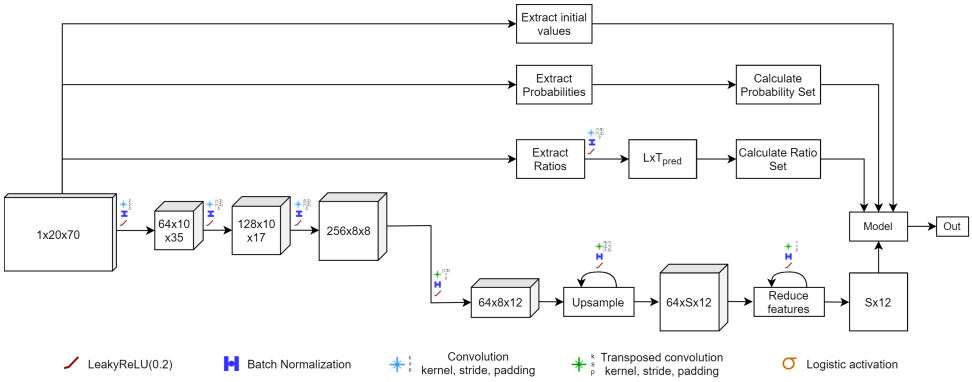

Figure 3.6: A schematic of the generator in the predictive setting. The 3D blocks represent the images at various stages throughout the network, noted using $C \times W \times H$.

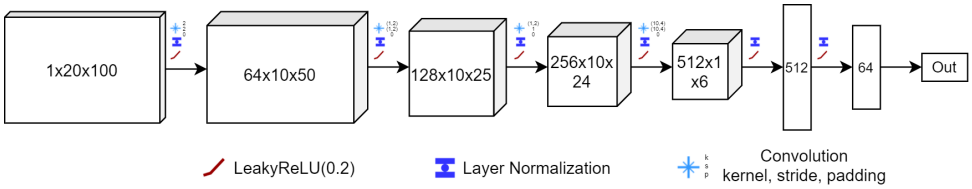

Figure 3.7: A schematic of the critic. The 3D blocks represent the images at various stages throughout the network, noted using $C \times W \times H$. The flat rectangles represent vectors of dimension C.

book, with a stride of 2, to summarize the information of the prices and volumes for each level and time step. Thereafter, we apply a 1×2 kernel over the summarized info to capture temporal correlations and reapply this kernel one more time to extend the captured correlation range. Finally, we summarize the captured information over a 10×4 filter to reduce the dimensions, and output the critic value after two fully connected MLP layers. After every layer, except for the last one, we apply both layer normalization and a leaky ReLU activation function, again with slope 0.2. The last layer does not have a layer normalization layer, nor an activation function, as required by the WGAN-GP architecture. Finally, the critic outputs a scalar value. A schematic of the critic is displayed in Figure 3.7, where exact kernel, stride and padding sizes are depicted. All layers are, similar to the generator, initialized through He initialization [22].

3.7. LOSSES

Since we are mainly using a WGAN-GP architecture, our losses for the discriminator (critic) and the generator follow from [20]. The loss of the critic consists of the earth mover's distance between the outputs of the critic for the real and fake data. This loss, however, requires the critic to be a 1-Lipschitz function. Towards this goal, the original WGAN paper [1] proposes to clip the weights of the critic between $[-c, c]$ for some constant $c \in \mathbb{R}^+$. This value may, however, be hard to tune for various critic architectures. In [20], a gradient penalty is issued for the critic, such that the Lipschitz constraint is

satisfied. Therefore, the loss of the critic $\mathscr{L}_C$ becomes

$$\mathscr{L}_C = \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] + \lambda_{GP}\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g}[(||\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})||_2 - 1)^2]. \tag{3.29}$$

The first two terms define an approximation of the earth mover's distance: $\tilde{\boldsymbol{x}} \sim \mathbb{P}_g$ defines a fake sample $\tilde{\boldsymbol{x}}$ from the generated data $\mathbb{P}_g$ and $\boldsymbol{x} \sim \mathbb{P}_r$ defines a real sample $\boldsymbol{x}$ from the real data $\mathbb{P}_r$. The last term is the gradient penalty, where λ_{GP} is the weight of the penalty and $\hat{\boldsymbol{x}}$ represents a sample which is a mix between a real and a fake sample from $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Since the generator can only influence one term in Eq. 3.29, namely the first one, its loss $\mathscr{L}_{G-WGAN}$ becomes

$$\mathscr{L}_{G-WGAN} = -\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})]. \tag{3.30}$$

Hence, the loss function of the generator is unchanged for WGAN-GP. We will, however, not leave it at that, as done by [33]. There is information about the order book that we know beforehand, hence we may exploit that to generate more realistic LOBs, faster. In particular, we will be looking at satisfying axioms 1 and 2, defined in Section 3.1. Axiom 1 states that $p_a^{(\ell)}(t) - p_b^{(\ell)}(t) > 0$, for all levels ℓ and time steps t. We may enforce Axiom 1 by setting $s(\ell,t) = p_a^{(\ell)}(t) - p_b^{(\ell)}(t)$ and calculate the maximum likelihood using $P(s(\ell,t) > 0) = \sigma_k(s(\ell,t))$. In this case, we define $\sigma_k(\cdot)$ as the k-squeezed logistic sigmoid function

$$\sigma_k(x) = \frac{1}{1+e^{-kx}}. \tag{3.31}$$

This enables us to squeeze (or widen) the curve of the logistic sigmoid function as we see fit. This way, since we know that both $p_a^{(\ell)}(t)$ and $p_b^{(\ell)}(t)$ are normalized to lie between $[0,1]$, we will have $-1 \le s(\ell,t) \le 1$. Using $\sigma_k(\cdot)$, we will then ensure larger gradients when using gradient descent, compared to the standard logistic sigmoid function. We then define the maximum likelihood estimation for parameters θ_G of the generator as

$$\mathscr{L}_{A1} = \prod_{\ell,t} \sigma_k(s(\ell,t)). \tag{3.32}$$

However, to make it numerically stable, we will utilize the negative log-likelihood loss instead, hence, by transforming Eq. 3.32 through logarithms, we acquire

$$\mathscr{L}_{A1} = -\sum_{\ell,t} \log(\sigma_k(s(\ell,t))). \tag{3.33}$$

Through the same logic, we can also satisfy Axiom 2, using $s_a(\ell,t) = p_a^{(\ell+1)}(t) - p_a^{(\ell)}(t)$ and $s_b(\ell,t) = p_b^{(\ell)}(t) - p_b^{(\ell+1)}(t)$ for the ask and bid prices, respectively. Following the same procedure, we arrive at the loss to enforce Axiom 2

$$\mathscr{L}_{A2} = -\sum_{\ell,t} \log(\sigma_k(s_a(\ell,t))) - \sum_{\ell,t} \log(\sigma_k(s_b(\ell,t))). \tag{3.34}$$

We may now construct the semi-final loss of our generator as

$$\mathscr{L}_G = \mathscr{L}_{G-WGAN} + \mathscr{L}_{A1} + \mathscr{L}_{A2}. \tag{3.35}$$

We say semi-final, because we have experienced multiple instances of mode collapse, even though we utilize the WGAN-GP architecture. This has to do with the fact that the distributions of the weights and probabilities became the same, which resulted in $\boldsymbol{x}_{up}$, $\boldsymbol{x}_{down}$ and $\boldsymbol{x}_M$ not making much differences, since the underlying distributions were the same anyways. To solve this issue, we defined a loss for the pairwise Kullback-Leibler (KL) divergence [48] in terms of the KL divergence as defined in Eq. 2.9. The pairwise KL-divergence measures the overall similarity of multiple distributions from specified target distributions and is defined as

$$D_{KLP}(W) = \frac{1}{S(S-1)} \sum_{i,j=1}^{S} D_{KL}(W_i || W_j). \tag{3.36}$$

In this case, $W \in W_S = \{T^S_{x,\ell}, R^{up,S}_x, R^{down,S}_x\}$. This way, we ensure that the S learned distributions are different from each other, such that our weights $\boldsymbol{x}_{up}$, $\boldsymbol{x}_{down}$ and $\boldsymbol{x}_M$ make a difference. Note that we aim to maximize Eq. 3.36, instead of minimizing it, which is commonly done in literature, since we want different distributions. This translates mathematically into minimizing the negative value of Eq. 3.36. We may now define the loss

$$\mathscr{L}_{KL} = -\sum_{W \in W_S} D_{KLP}(W). \tag{3.37}$$

Note that this loss directly and only affects the weights and probabilities. This might result in unstable learning curves, taking into account the other losses that attempt to update these matrices as well. This is why we introduce a weight λ_{KL} for $\mathscr{L}_{KL}$, such that we might limit its update influence on the matrices, but still attempt to generate different distributions. Our loss for the generator in Eq. 3.35 then becomes

$$\mathscr{L}_G = \mathscr{L}_{G-WGAN} + \mathscr{L}_{A1} + \mathscr{L}_{A2} + \lambda_{KL}\mathscr{L}_{KL}. \tag{3.38}$$

Finally, the parameters θ_C and θ_G of the critic and the generator, respectively, get updated by using gradient descent, so the main updates become

$$\boldsymbol{\theta}_C = \boldsymbol{\theta}_C - \alpha_C \frac{\partial \mathscr{L}_C}{\partial \boldsymbol{\theta}_C} \tag{3.39}$$

and

$$\boldsymbol{\theta}_G = \boldsymbol{\theta}_G - \alpha_G \frac{\partial \mathscr{L}_G}{\partial \boldsymbol{\theta}_G}, \tag{3.40}$$

for learning rates α_C and α_G. This is the standard gradient descent update rule, however, we will be using the Adam optimizer for our update, as described in Section 4.2.

4

EXPERIMENTS

In this chapter we shall go through the data sets and experimental setups that we have utilized for our experiments in the generation and prediction parts. The section will start off with a discussion of our data sets and will continue with the experiments for generation and prediction. Finally, we will also perform a sensitivity analysis and ablation study to look at the effects of our losses on the model.

4.1. DATA SETS

For the data set part, there were not many options for us to choose from. Order book data is usually kept private by organizations to optimize their models with and there are a few open data sets for academic purposes. One of those data sets is the FI-2010 data set [39] and is to our knowledge the most used one in literature. The data set consists of training data for nine days, where each day consists of the order books of five stocks. These order books have a depth of 10 levels and are in sorted order, so one is able to distinguish the order books of one stock from the other. The authors have utilized three separate normalization methods, being z-score normalization, which for a sample x_i with mean μ_x and standard deviation σ_x applies the following transformation

$$\bar{x}_i^{z-score} = \frac{x_i - \mu_x}{\sigma_x}, \tag{4.1}$$

min-max (MM) normalization, defined by

$$\bar{x}_i^{MM} = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \tag{4.2}$$

and decimal precision (DP) normalization, defined by

$$\bar{x}_i^{DP} = \frac{x_i}{10^k}, \tag{4.3}$$

where $k = \arg\max_k |\bar{x}_i^{DP}| < 1$. For each normalization method, they also provide a data set containing the auction period of the market and one without the auction period. Since there is a risk of running into crossed order books in the auction period, we will use the non-auction data set. The question now arises as to which normalization method is most optimal to use.

To see which data set contains valid order books, we apply axioms 1 and 2 from Section 3.1 to all three normalization methods. It seems that only the decimal precision approach resulted in valid order books: the other two normalization order books did not satisfy either one of the axioms. This is an important observation, since this data set is being used widely in LOB/HFT literature. Indeed, if the data set does not contain valid LOBs, then the machine learning method shall learn to match patterns that do not represent valid LOBs to some output data.

There is one more point that has to be addressed regarding this data set, being the scale of the data. Since the data set contains the data for five stocks, the scale of the data is varying. A machine learning method that has to process this data should then also be able to take that into consideration, which is not very efficient, since the model then also has to learn a transformation for the underlying data. Instead, we can do this transformation before we feed the data to the model. The transformation is done in three steps: 1) identify the indices of all five stocks, 2) normalize all prices and volumes using Eq. 4.2 of each stock to scale the values between 0 and 1 and finally 3) concatenate the stocks together. Note that the normalization is applied over the prices and volumes separately. This transformed data set still satisfies the first two axioms. There is one final note to be made about this data set, which is that it is one of a few years back, in a time where markets were not very liquid. To account for this, we introduce another data set.

Using the Binance API, we gathered one million order book entries for the BTC/USDT ticker. This ticker tracks the value of the bitcoin cryptocurrency denoted in US dollars. Every order book arrived 100 milliseconds after the other and has a depth of 10 levels as well. The data is gathered from the 1st of March 2021 to the 14th of March 2021: a very liquid and volatile period for BTC/USDT. This data is formatted in a similar order as that of the FI-2010 data set, namely using $\{p_a^{(\ell)}, v_a^{(\ell)}, p_b^{(\ell)}, v_b^{(\ell)}\}_{\ell=1}^{10}$ for each row.

For both data sets, we shall use 150K samples for training and another 50K samples for testing. Note that the samples for testing are only utilized in the predictive setting. Even though we have access to more data samples for both data sets, we will only consider the given number of samples due to computational resources. The samples of the FI-2010 data set used for testing originate from the day after the last day of the training set. The testing data for the BTCUSDT data set are 50K samples that come right after the 150K training samples, hence, they originate from the first thirteen hours after the training data. In Figure 4.1 and 4.2 we depict properties of the FI-2010 and BTCUSDT data sets, respectively. In particular, we see that the return distribution, calculated through

$$r_t = \frac{p_{t+1} - p_t}{p_t} \tag{4.4}$$

for midprices p_t at time t, of the BTCUSDT data set is more volatile compared to the FI-2010 data set. A very interesting comparison is that of the distribution of the probabilities of incoming limit orders (LOs). We see a distribution that is Poisson-like in that of the FI-2010 data set, which nicely fits the assumption of [11] that the arrival of limit orders is distributed as a Poisson-point process. However, when we look at the same distribution of BTCUSDT, we see that the probability of incoming LOs is almost invariant regardless of the level within the order book. This is a practical example of how statistical assumptions do not always hold in reality. Finally, we see the midprice movements depicted in the figures as well. We see a large amount of variance in the FI-2010 data set, which is the result of it containing multiple stocks. When examined more closely, one can clearly distinguish the different stocks through the sudden drops and rises in the midprice.

4.2. GENERATION

For our generation procedure, we experimented with both data sets, and have run a total of three models: deep convolutional GAN (DCGAN), WGAN-GP, and order book GAN (OBGAN), which is our generative model. Since there is not much research into the generation of whole order books, we will be using those two models as our baseline. All models have roughly the same number of parameters, being 3 million (M), and the architecture for their generators is depicted in Figure 4.3. The discriminator's architecture stays the same as with OBGAN, except for the output and normalization layers for DCGAN: the output gets put through a logistic activation function and the layer normalization becomes batch normalization. We use the Adam optimizer with a learning rate $\alpha = 10^{-3}$, $\beta_1 = 0$ and $\beta_2 = 0.9$ for our models and train each model for 50 epochs, regardless of the data set used. Through a grid search, we have found the parameters for $\lambda_{KL} = 10^{-1}$, $\lambda_{GP} = 10$, $\gamma = 0.95$, $n_{critic} = 5$, and $S = 16$ to be the best performing ones. We will use $L = 10$ and $T = 50$ to get order book matrices of 20×100. For the sampling

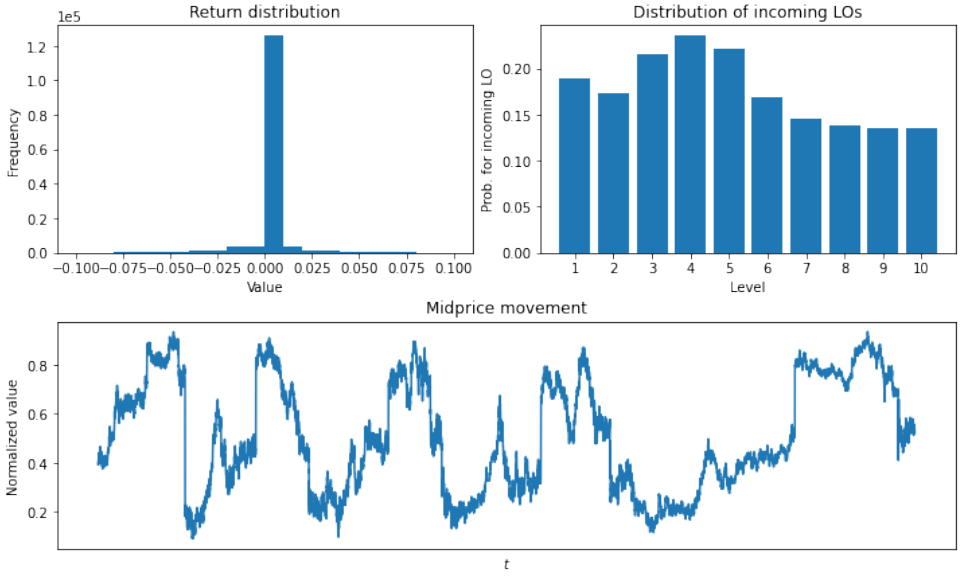

Figure 4.1: The return distribution, distribution of incoming limit orders (LOs) and midprice movement of the FI-2010 data set.

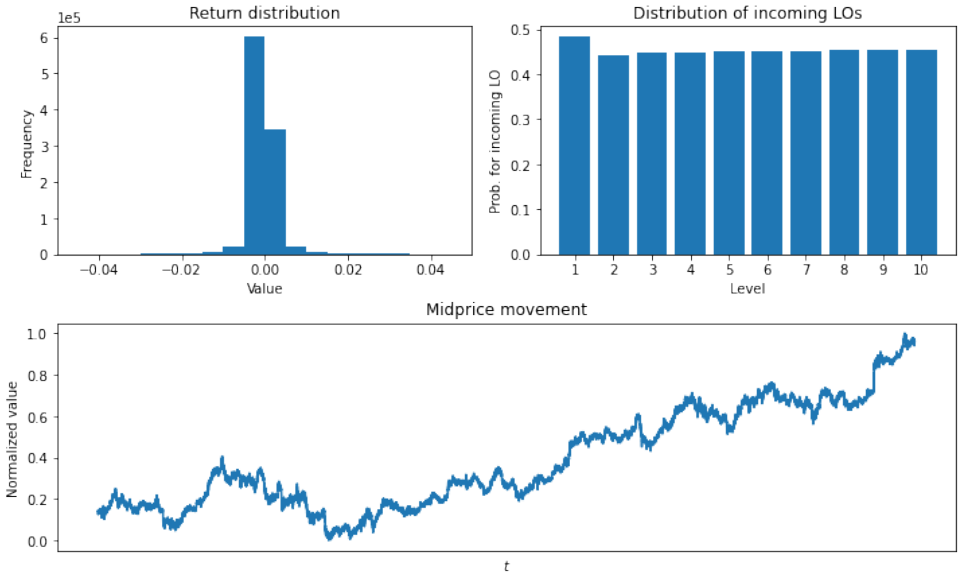

Figure 4.2: The return distribution, distribution of incoming limit orders (LOs) and midprice movement of the BTCUSDT data set.

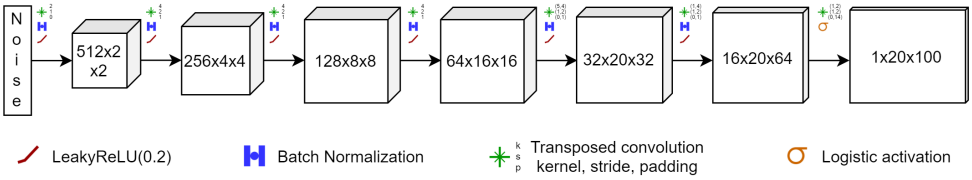

Figure 4.3: The architecture of the generators of DCGAN and WGAN in the generative setting. The 3D blocks represent the images at various stages throughout the network, noted using $C \times W \times H$. The flat rectangles represent vectors of dimension C.

procedure described in Section 3.4.1, we utilize a sampling function Ψ to sample π_{ij} 3 times more often for $i = \ell$, compared to $i \neq \ell$, as denoted by Eq. 3.20. Hence, we set our sampling rate $\xi = 3$. Furthermore, we utilize a batch size of 1024 and acquire an input vector in $\mathbb{R}^{100}$ for the generator sampled from a $\mathcal{N}(0,1)$ distribution. All experiments discussed in this work have been run on a single Tesla P100 16GB on Google Colab.

Evaluating the performance of a GAN has always been an active research topic. This is also a challenge in our case, since there are no guidelines to evaluating the realness of an order book. We can, however, utilize axioms 1 and 2 as measures for the structural realness. This would translate into axiom accuracies $\mathscr{A}$ of the form

$$\mathscr{A}(f) = \frac{1}{|\mathscr{W}|} \sum_{(i,j) \in \mathscr{W}} \mathbb{1}_{f(i,j)}, \tag{4.5}$$

where $(p_a, p_b) \in \mathscr{W}$ is the set of all bid-ask price pairs for axiom 1 in the order book, such that $f_1(p_a, p_b) = p_a - p_b > 0$. For axiom 2, we would have $(p_a^{(\ell+1)}, p_a^{(\ell)}) \in \mathscr{W}$ to be the set of all ask price pairs for each level, such that $f_{2a}(p_a^{(\ell+1)}, p_a^{(\ell)}) = p_a^{(\ell+1)} - p_a^{(\ell)} > 0$. Similarly, for the bid price pairs, we would have $(p_b^{(\ell+1)}, p_b^{(\ell)}) \in \mathscr{W}$, such that $f_{2b}(p_b^{(\ell+1)}, p_b^{(\ell)}) = p_b^{(\ell)} - p_b^{(\ell+1)} > 0$. Note that $0 \leq \mathscr{A}(f) \leq 1$ and that $0 \leq \frac{1}{3}\Big(\mathscr{A}(f_1) + \mathscr{A}(f_{2a}) + \mathscr{A}(f_{2b})\Big) \leq 1$. These equations define the accuracies that will measure how real the structure of the generated order book looks.

We shall then also compare three more metrics: the distribution of the returns, the price distribution and the volume probability distribution. The latter is done by counting the number of times that the volume has gone up, stagnated, and has went down and dividing it by the total number of time steps. We do this per level, such that we get the probability of a movement in the volume per level, representing the addition of limit or market orders, and the cancellation of orders. In terms of Eq. 4.5, we can define the probability for an up movement in the volume $v_\ell^{(t)}$ at level ℓ and time step t as $f_{up}(v_\ell^{(t)}, v_\ell^{(t+1)}) = v_\ell^{(t+1)} > v_\ell^{(t)}$, a stagnation as $f_{stag}(v_\ell^{(t)}, v_\ell^{(t+1)}) = |v_\ell^{(t+1)} - v_\ell^{(t)}| < \epsilon$ for $\epsilon \to 0$ and down movement as $f_{down}(v_\ell^{(t)}, v_\ell^{(t+1)}) = v_\ell^{(t+1)} < v_\ell^{(t)}$. Having the estimated probabilities for both the real order books and the generated order books, we quantify the error between the two by means of the mean squared error, defined as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (p_i - \hat{p}_i)^2, \tag{4.6}$$

for the generated probability $\hat{p}_i$ and true probability p_i. This loss is calculated per level and is finally averaged over all levels.

As for the difference in the distributions, we measure it per means of the Chebyshev distance between two empirical distribution functions $F_{1,n}$ and $F_{2,m}$ with n and m samples respectively, defined as

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|. \tag{4.7}$$

This distance is also calculated per level, since the distributions may differ for each level. Hence, for the final score, we average out the scores over all levels. Furthermore, since we can only generate order books up to $t = T$, it would be hard to compare the distributions with the underlying data. Therefore, we generate μ number of samples for each model and take the average resulting scores. We take μ to be a variable, since it is interesting to see which model can best approximate the underlying data distribution with as little samples as possible. In particular, we take $\mu \in \{512, 1024, 4096, 16384\}$.

There is another aspect of generation that we will be covering, which is the output diversity. GAN evaluation still remains a challenge and the applied measures differ per field. The Inception Score [42] is able to measure how good the quality and diversity of the output of a GAN is. However, this score is only applicable to generated images, containing actual objects. Our research does not fall into that category, so we need to resort to other means of measuring the diversity of our output, since we already cover the quality part through our previously mentioned measures. In particular, we will be creating a standard deviation map (SDM) of our order books, where we will measure the standard deviation of every variable at each time step at each level and plot a heatmap of the acquired standard deviations. This way, we will be able to see how large the standard deviation is of each variable: the closer they are to zero, the more probable it is that we have encountered mode collapse. In contrast, the larger they are, the larger the variety is of the produced output.

4.3. PREDICTION

As for our experiments on prediction, we utilize the model proposed in Section 3.5. We use the same hyperparameters and sampling function for the model as with generation, but change $\gamma = 0.99$, for which we shall see the reason in Section 5.3.1. Since we are now considering prediction, we will be comparing our model not only to adversarially trained models, but also to supervised models and traditional time series prediction models. For the adversarial models, we will be using both DCGAN and WGAN-GP, with a different architecture since they will not be generating order book anymore, but taking in an order book of $L \times T_{base}$ and outputting an order book of $L \times T_{pred}$. The architecture for these generators is depicted in Figure 4.4, along with kernel sizes, paddings, strides, activations and normalization methods.

For the supervised models, we will be comparing our results with a multilayer perceptron (MLP) containing 512, 256, 128 and 64 hidden units. Since an MLP takes in a one dimensional input, we flatten the $L \times T_{base}$ matrices to an $L * T_{base}$ vector. We also use a CNN with five convolutional layers: the first three layers have kernel sizes 1×11 and the last two layers 1×6, with all strides and paddings set to one and zero, respectively. As

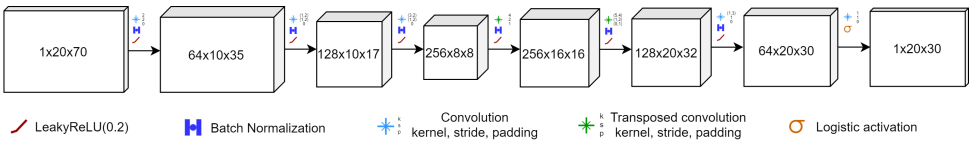

Figure 4.4: The architecture of the generators of DCGAN and WGAN in the predictive setting. The 3D blocks represent the images at various stages throughout the network, noted using $C \times W \times H$. The flat rectangles represent vectors of dimension C.

can be seen from our kernel sizes, the convolutions are only applied to the time dimension, such that the spatial dimension is kept in tact. Both MLP and CNN contain batch normalization followed by the Leaky ReLU activation function with a slope of 0.2 for all layers, except for the last one, which is only followed by the logistic activation function.

Furthermore, we also use a traditional forecasting method for time series. Since our work considers multivariate prediction, we resort to vector autoregression (VAR). A VAR(1) model is able to capture dependencies between the variables of the previous time step through optimizing

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix}. \tag{4.8}$$

This is the VAR(1) equation for two variables, where the intercept c and the coefficient matrix A are calculated through optimizing the least squares fit. The error term e is a vector white noise process with zero mean. Note that the VAR model is only able to capture linear dependencies between variables [60]. A generalization of this model is called a VAR(p) and extends the lag range of this equation to p, such that the dependencies between the variables of the previous p time steps is considered as well. Using matrix and vector notation for convenience, the resulting equation becomes

$$\boldsymbol{y}_t = \boldsymbol{c} + \sum_{k=1}^{p} \boldsymbol{A}_k \boldsymbol{y}_{t-k} + \boldsymbol{e}_t, \tag{4.9}$$

where $\boldsymbol{y}_t$ is the vector of variables at time step t, the intercept vector $\boldsymbol{c}$ contains an intercept for all variables, $\boldsymbol{A}_k$ is the k-th coefficient matrix and $\boldsymbol{e}_t$ is the white noise vector at time step t. For our experiments, we will be using a VAR(4) model which we acquired through a search for p where we measured the model with the lowest Akaike information criterion (AIC), defined as $AIC = 2k - 2\ln\hat{L}$ for k parameters and the maximum value of the likelihood function $\hat{L}$. Utilizing this model enables us to capture information across four lag steps per variable and enables us to compare it with our model which only incorporates information from the previous variable. There are quite distinct, but discrete differences between our model and the VAR model, other than the fact that we utilize a deep learning architecture. Whereas the VAR model learns a linear combination of the variables to predict one for the next time step, our model learns the actual rates, probabilities and with that, the expectation of the variable for the next time step.

Finally, as an addition to the supervised model, we also utilize our prediction model from Figure 3.6 and train it in a supervised manner as well, without a discriminator and title it order book network (OBNet). All supervised models, including this one, are

Model	No. of Learnable Parameters
VAR(4)	6.5K
OBGAN/OBNet	340K
MLP	930K
CNN	1.24M
WGAN/DCGAN	1.9M

Table 4.1: A list of all models along with their number of learnable parameters in the predictive setting.

trained through minimizing the MSE loss. This is done to measure the performance of the models alone, without taking the losses into account. We will conduct an ablation study to measure the impact of the different losses in Section 5.3.2.

We measure the MSE of the predicted order books to compare the models, as defined in Eq. 4.6. In addition, we measure the generalizibility of all networks by testing them on the data sets that they were not trained on: training on FI-2010 and testing on BTCUSDT and vice versa.

To give an overview of the used models for prediction, we list them in Table 4.1, along with the number of learnable parameters per model. VAR(4) has the least number of parameters, which is expected from a purely analytical model which only fits its parameters once. However, if we keep look further, we see a huge decrease in the number of parameters for our model, which got reduced by an order of magnitude, while the WGAN and DCGAN models still remain relatively large. This decrease is caused by the removal of the branch in Figure 3.3 that learns the initial values and by the smaller matrices that our model uses for prediction, since per definition of our model as described in Section 3.4 $T = T_{pred}$. Our model also has less parameters compared to an MLP with four hidden layers and a CNN with four convolutional layers. Note that we did not include the number of parameters of the discriminator for the networks that are trained adversarially since the discriminators are all the same.

4.4. SENSITIVITY ANALYSIS & ABLATION STUDY

To broaden our insights into the model, we shall perform a sensitivity analysis on both the generative and predictive models, respectively OBGAN and OBNet. Since the task for both models are quite different, we expect the parameters for both models to differ as well, which is the reason why we run the analysis on both models. For this analysis, we shall use the one-factor-at-a-time (OFAT) method, where we vary the hyperparameters of the model individually, while keeping the other parameters fixed. In particular, we shall consider the following parameters: the number of estimated distributions S, the sampling rate ξ and the walk discount γ for both models. For the generative model, we shall also consider the weight for the pairwise KL loss λ_{KL}, since the KL loss is not used in the predictive setting. The domain of each variable is listed in Table 4.2. We run our sensitivity analysis solely on the BTCUSDT data set, since this data set is not mixed up like FI-2010, so the data will not bias our model.

The case of OBNet is more fruitful than the case of OBGAN, since OBNet is trained directly, without the use of a critic, which might affect the way the other parameters

Variable	Domain	Model
S	{ 8, 16, 32, 64, 128 }	Gen./Pred.
ξ	{ 1, 2, 3, 4, 5 }	Gen./Pred.
γ	{ 0.9, 0.925, 0.95, 0.975, 0.99 }	Gen./Pred.
λ_{KL}	{ 1, 0.1, 0.01, 0.001, 0.0001 }	Gen.

Table 4.2: The tweaked parameters along with the domain used for the sensitivity analysis.

of the model are learned. Rationalizing about the tasks of both networks, we suppose that additional stochasticity due to the parameters favours the generative model and less stochasticity favours the predictive model. The parameters influencing the stochasticity of the model are mainly ξ and γ. Lower values for ξ let the model consider cross-level interactions and lower values of γ make the walk in Eq. 3.24 more stochastic. Hence, higher values of both variables make the model more deterministic. We suspect that these variables are the most interesting ones, after which the number of estimated distributions is also interesting, as it determines and underfit or overfit on the data. The λ_{KL} variable should influence the learning process the most, since higher values might disturb the other losses and thereby make the training process less stable.

We shall also conduct an ablation study, where we start off with the WGAN loss as defined in Eq. 3.30 and add the losses of Eq. 3.33, Eq. 3.34, and Eq. 3.37 respectively until we finally reach the full loss of Eq. 3.38, such that we can measure the impact of each loss on the model. Note that the ablation study shall only be conducted on the generative network (OBGAN) and not on OBNet, since the latter does not utilize any other losses than the MSE loss. We suspect that each axiom's loss will contribute to the corresponding axiom accuracy that we measure, since we almost directly optimize for those scores. The last loss, we suspect, shall contribute to a more stable learning process to omit the problem of mode collapse.

5

RESULTS

In this chapter, we shall go through the results that we have acquired from our experiments. We shall first discuss the results of the generative and predictive settings. In both instances, we look deeper into the model and visualize its learned features. We continue by discussing of how these graphs can be interpreted and what the role of the various model parameters are. The latter is discovered in the sensitivity analysis and ablation study, where we also measure the effect of the different losses on the model.

Model	μ	Axiom 1	Axiom 2: PA	Axiom 2: PB	Avg. Axioms	Return Dist	Price Ask Dist	Price Bid Dist	Volume Ask Dist	Volume Bid Dist
WGAN-GP	512	0.95	0.76	0.50	0.74	0.68	0.36	0.35	0.21	0.22
	1024	0.95	0.76	0.50	0.74	0.67	0.36	0.35	0.21	0.22
	4096	0.96	0.76	0.50	0.74	0.67	0.35	0.35	0.21	0.22
	16384	0.95	0.76	0.50	0.74	0.67	0.35	0.35	0.21	0.22
DCGAN	512	1.00	0.74	0.75	0.83	0.70	0.42	0.38	0.21	0.22
	1024	1.00	0.75	0.75	0.83	0.69	0.42	0.38	0.21	0.22
	4096	1.00	0.75	0.75	0.83	0.70	0.42	0.38	0.21	0.22
	16384	1.00	0.75	0.75	0.83	0.70	0.42	0.38	0.21	0.22
OBGAN (ours)	512	1.00	0.92	0.87	0.93	0.61	0.27	0.28	0.21	0.22
	1024	1.00	0.96	0.87	0.95	0.62	0.25	0.27	0.21	0.22
	4096	1.00	0.94	0.88	0.94	0.62	0.26	0.27	0.21	0.22
	16384	1.00	0.93	0.88	0.94	0.62	0.27	0.27	0.21	0.22

Table 5.1: Results for generation for the FI-2010 data set. For the axioms a higher score is better and for the returns, prices and volumes a lower score is better. The relative goodness of the scores is depicted by the greenness of the cell.

5.1. GENERATION

5.1.1. RESULTS

We depict the results for the generative setting using the FI-2010 data set in Table 5.1. Since the scores for the axioms are given by accuracies, a higher score means a better one. For the returns, prices, and volumes we either use the distance given by Eq. 4.7 and the MSE loss as given by Eq. 4.6. The MSE loss is only used for the volumes. The better the score, the greener the cell is.

First looking at the axiom scores, we see that our model is able to generate structurally more realistic order books compared to the other two models. We attribute this to the fact that we optimize the axiom losses, such that the network pays more attention to the structures of the order books it generates. We are also able to better approximate the return and price distributions compared to the other two models, however, there is almost no difference in the scores for the volumes. This has to do with how the model works: the model assumes an increase or decrease in the variable, which is bounded between two given values. Indeed, this makes sense for the price variables, as very large jumps in the price are uncommon. However, a relatively large volume transaction might occur at any time, which would result in an increase or decrease in the volume at a given level in the order book. Hence, our model is unable to accurately adapt to the volume processes given in the data, resulting in no difference compared to the other models. Taking the different test sizes μ into consideration, we see that all models seem to perform consistent regardless of the generated amount of samples.

Having better scores is not the only advantage that our model brings. The core value of the model lies in its interpretability, since we learn probability transition matrices and rates, which a trader might use to better understand the market and build confidence in their choices.

We continue by looking at Table 5.2 for the results of the BTCUSDT data set. We see that the two other models had a harder time generating more realistic-looking order books, while the performance of our model is still consistent. The price distributions are also better approximated, however, WGAN-GP seems to be able to keep up with the return distribution. This is attributed to the fact that the BTCUSDT data set, as the name suggests, only contains the BTCUSDT ticker, while the FI-2010 data set is a mix of five different stocks. This time around, we see that WGAN-GP performs slightly better in approximating the volumes. This has to do with the same argument that we mentioned

Model	μ	Axiom 1	Axiom 2: PA	Axiom 2: PB	Avg. Axioms	Return Dist	Price Ask Dist	Price Bid Dist	Volume Ask Dist	Volume Bid Dist
WGAN-GP	512	0.81	0.51	0.50	0.61	0.62	0.48	0.48	0.003	0.002
	1024	0.81	0.51	0.50	0.61	0.62	0.48	0.48	0.003	0.002
	4096	0.81	0.51	0.50	0.61	0.62	0.48	0.48	0.003	0.002
	16384	0.81	0.51	0.50	0.61	0.62	0.48	0.48	0.003	0.002
DCGAN	512	0.81	0.50	0.62	0.64	0.70	0.50	0.50	0.034	0.032
	1024	0.81	0.50	0.62	0.64	0.71	0.50	0.50	0.033	0.031
	4096	0.81	0.50	0.62	0.64	0.70	0.50	0.50	0.034	0.032
	16384	0.81	0.50	0.62	0.64	0.70	0.50	0.50	0.034	0.032
OBGAN (ours)	512	0.94	0.80	0.74	0.83	0.61	0.35	0.32	0.006	0.008
	1024	0.95	0.82	0.74	0.84	0.60	0.34	0.31	0.007	0.007
	4096	0.93	0.86	0.72	0.84	0.60	0.34	0.32	0.007	0.008
	16384	0.93	0.81	0.70	0.82	0.61	0.34	0.32	0.007	0.008

Table 5.2: Results for generation for the BTCUSDT data set. For the axioms a higher score is better and for the returns, prices and volumes a lower score is better. The relative goodness of the scores is depicted by the greenness of the cell.

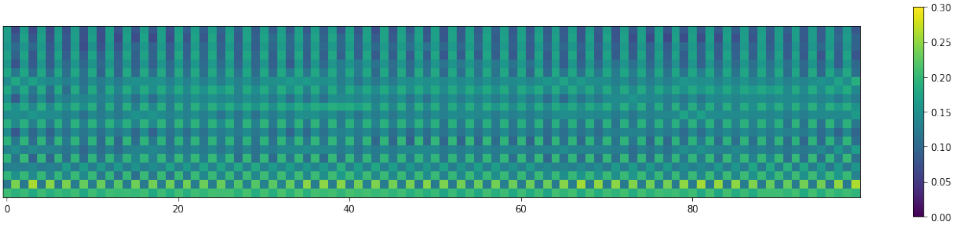

Figure 5.1: The SDM of WGAN-GP

before: a network consisting of only convolutional layers is better able to capture the stochasticity of the volumes since there are no inherent assumptions made in the model. Finally, we see that all models seem to perform consistently across all generated amount of samples μ.

We now turn our attention to the SDMs, depicted in Figures 5.1, 5.2, and 5.3, for WGAN-GP, DCGAN, and OBGAN, respectively. We directly notice the lack of variance in the SDM of DCGAN, which likely suffered from mode collapse, as it is almost completely dark compared to the other two SDMs. The other two SDMs look quite similar, where we can clearly distinguish the two at lower levels in the order books: the ask price gets a progressively lower standard deviation in OBGAN, while the lower variables in the lower levels of WGAN get a larger standard deviation. The former case, however, is more realistic, since the variables at lower levels are less likely to vary more, compared to the variables at higher levels, since the trades take place at higher levels. This is partially depicted by both networks through a horizontal line above the middle level of the SDM, where we see that all standard deviations remain bright throughout the whole order book, indicating that these variables vary the most.

5.1.2. DISCOVERING THE MODEL

As we have mentioned earlier, the OBGAN model is explainable and interpretable. We can extract useful information from it, such as the learned rates and transition probabilities. Here, we will take the ask price variable p_a as an example in the FI-2010 data set and demonstrate the various properties that we can extract from the model to gain more insight into the underlying data.

First of all, let us take a look at Figure 5.4 where the distribution of up rates ($r_t > 1$)

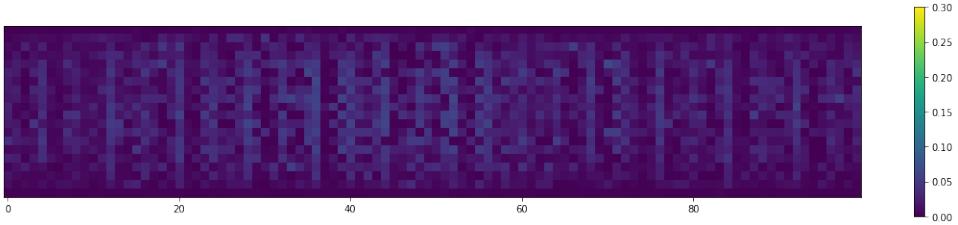

Figure 5.2: The SDM of DCGAN

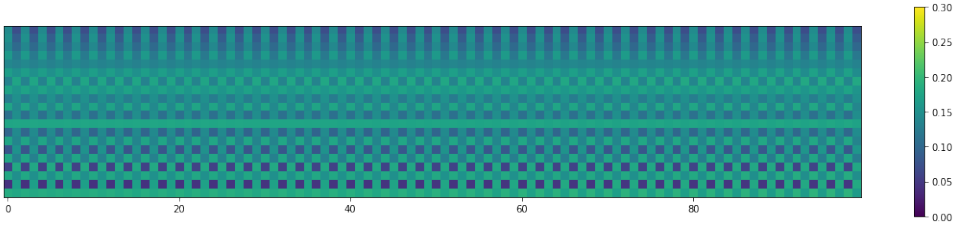

Figure 5.3: The SDM of OBGAN

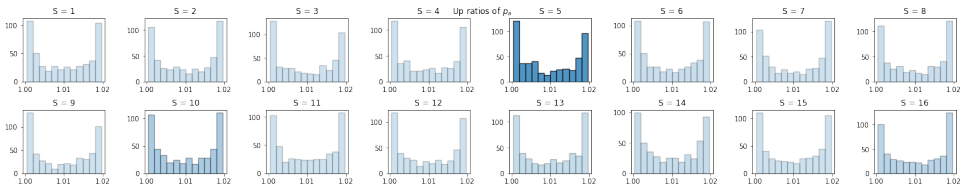

Figure 5.4: Distribution of up rates ($r_t \geq 1$) for p_a for all feature channels ($S = 16$). Plots with a higher opacity are the ones that the network weighs more.

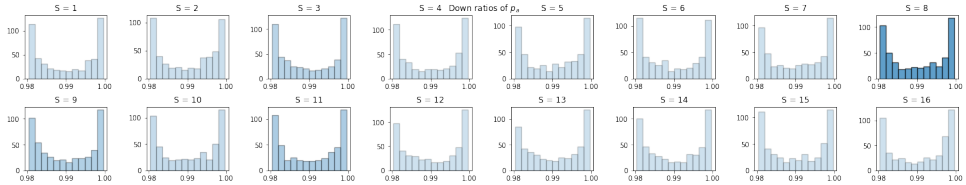

Figure 5.5: Distribution of down rates ($r_t \leq 1$) for p_a for all feature channels ($S = 16$). Plots with a higher opacity are the ones that the network weighs more.

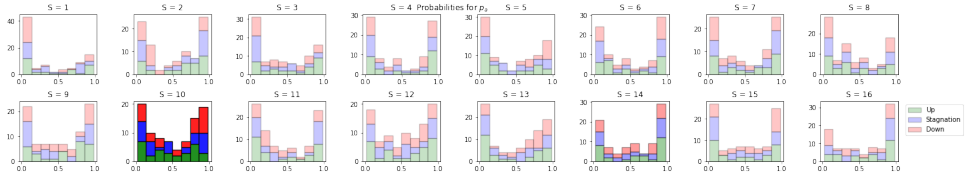

Figure 5.6: Distribution of probabilities for up and down movements and stagnations for p_a for all feature channels ($S = 16$). Plots with a higher opacity are the ones that the network weighs more.

are depicted for p_a for all feature channels. We have also measured the mean values of the learned weights $\boldsymbol{x}_{up}$, which come from the lower branch in Figure 3.3, and have assigned a higher opacity to those which got a higher mean weight from the network. As we can see, all plots seem to have the U-shape in common, where there are more values around 1 and towards the extreme of 1.02. However, the frequencies of the values in between differ, where some distributions look more uniform in the middle and the other ones look more biased towards the extremes. We see that the distribution of $S = 5$ is weighted the most, which has a larger bias towards the lower extreme and the lowest values of all distributions in the middle. It gets more interesting, however, if we also take a look at Figure 5.5, where the down rates ($r_t \leq 1$) are depicted in a similar fashion to that of Figure 5.4. Here we see that the values are more biased towards lower extremes in the most favored distribution by the model ($S = 8$). Indeed, if we imagine the distributions from these figures next to each other, where they meet at $r_t = 1$, we see that the two distributions would form a tri-modal distribution, with the modes being at both the lower and upper extreme and in the middle. This can be seen as a normal distribution that admits fatter tails, along with modes at the extremes.

Finally, we draw our attention the Figure 5.6, where the probabilities of up and down movements and stagnations are depicted in a similar fashion to that of the previous plots. We see that the movement with the highest frequency for large probabilities is the down movement, as depicted by the favored distribution by the model ($S = 10$). In most distributions, the down movement is mostly biased against the extremes, meaning that there are certain down movements or no down movements at all. Stagnations also seem to share this property. This differs from up movements, which seem to be more frequent compared to the other movements in the middle.

Model (FI-2010)	FI-2010 MSE	BTCUSDT MSE
WGAN-GP	0.0072	0.0134
DCGAN	0.0104	0.0190
OBGAN (Adv.)	0.0069	0.0095
VAR(4)	0.0670	0.0958
MLP	0.0110	0.0164
CNN	0.0057	0.0010
OBNet (Sup.)	0.0050	0.0008

Table 5.3: Results for prediction where the models have been trained on the FI-2010 data set.

5.2. PREDICTION

5.2.1. RESULTS

We list the results for the predictive setting of the networks trained on the FI-2010 data set in Table 5.3. Firstly note that out of the adversarially trained networks, OBGAN performs the best, achieving a slightly better score compared to WGAN-GP for the FI-2010 test set, but acquiring half as much MSE for the generalization test on the BTCUSDT test set. In fact, the only model that is able to outperform OBGAN, other than OBNet, is the CNN, which is also able to generalize quite well. However, CNN is outperformed on the FI-2010 data set by OBNet, while achieving a similar, but better generalization result. These results are promising, since OBNet uses far fewer parameters compared to the other models and also supplies the user with additional features about the underlying data, which can be used to make sense out of the prediction.

The worst result is achieved by the MLP. This has to do with the high dimensionality of the input and the fact that a multivariate time series is fed into the network. VAR(4) is able to exploit this latter fact, although it being a simpler model than its competitors. Almost all the other models involve convolutional layers, which are able to exploit the spatio-temporal dimensions of the data, resulting in better scores.

We continue by looking at the scores of the models on the BTCUSDT data set in Table 5.4. A similar case arises where OBGAN outperforms the other adversarially trained models, but this time is outperformed by both VAR and CNN. The lowest MSE on the data set and the lowest generalization result is, however, acquired by OBNet. In fact, if we compare the results of OBNet in Table 5.4 and Table 5.3, we find that the scores look quite similar.

This either suggests that the data in BTCUSDT and FI-2010 is somewhat correlated or that OBNet is able to learn a part of the underlying dynamics in order books that apply to both data sets. Indeed, if the latter was the case, then the results of the other models should have reflected this as well. This implies that OBNet is able to learn the transition matrices and rates that are implicit in both data sets, regardless of which data set is fed into it. Note that we say that OBNet learns the transition matrices and rates, but unlike generation where we directly learn the transition matrices and rates of the underlying data, in the predictive setting we learn the weights for those matrices and weigh them accordingly. The result is then a matrix that depicts the transition matrices and rates of the future.

Model (BTCUSDT)	BTCUSDT MSE	FI-2010 MSE
WGAN-GP	0.0062	0.0478
DCGAN	0.0108	0.0564
OBGAN (Adv.)	0.0041	0.0167
VAR	0.0033	0.0359
MLP	0.0148	0.0170
CNN	0.0003	0.0096
OBNet (Sup.)	0.0002	0.0058

Table 5.4: Results for prediction where the models have been trained on the BTCUSDT data set.

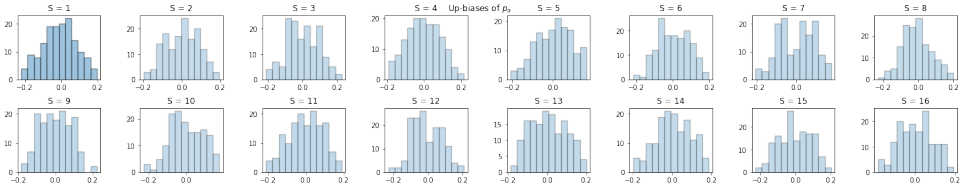

Figure 5.7: Distribution of biases for up movements ($r_t \geq 1$) for p_a for all feature channels ($S = 16$) of OBNet in the predictive setting. Plots with a higher opacity are the ones that the network weighs more.

In both data sets, the OBNet model outperforms all other models, including VAR(4) and CNN. The former result suggests that, although our model does not consider variables from larger time lags in its mathematical definition, it is still able to incorporate this information. This has to do with the deep learning architecture that comes before the mathematical model. In fact, the model maps the information between the variables to the prioritization vectors $\boldsymbol{x}_{up}$, $\boldsymbol{x}_{down}$, and $\boldsymbol{x}_M$ and models their correlations that way. This is of course different for the CNN, which does not make this mapping and instead computes the predicted variables in a direct manner.

5.2.2. DISCOVERING THE MODEL

As with generation, we shall now turn our attention to the learned model parameters. Instead of learning the rates and probabilities directly, the predictive model of OBNet learns the ratio biases and weights for the probability transition matrix, which we shall showcase now. It is indeed possible to acquire the resulting transition matrices and rates, but since this differs per sample in a batch, we resort to a showcase of the learned features instead.

If we look at Figure 5.7, where the distributions of all features for biases of up movements are depicted, we see that they resemble a normal distribution. We have again calculated the mostly preferred distributions by measuring the outputs of the network in Figure 3.6 throughout the test set for FI-2010. We see that $S = 1$ has the highest weight, while the other distributions all look quite similar in their weights. All biases also have a range of [-0.2, 0.2], where some of them look more uniform in the middle and some of them have a distinct mode and sometimes even multiple modes.

Looking at Figure 5.8, we see that the biases for downwards movements look quite

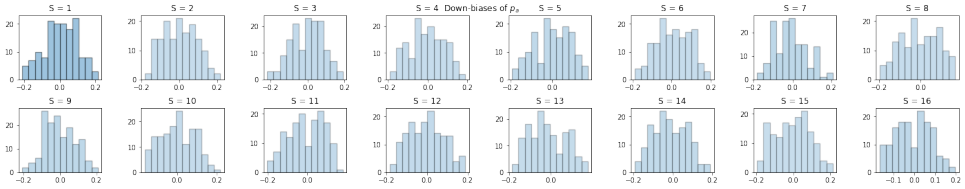

Figure 5.8: Distribution of biases for down movements ($r_t \leq 1$) for p_a for all feature channels ($S = 16$) of OBNet in the predictive setting. Plots with a higher opacity are the ones that the network weighs more.

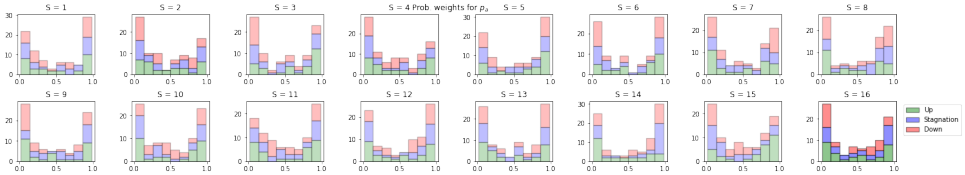

Figure 5.9: Distribution of the probability weights for p_a for all feature channels ($S = 16$) of OBNet in the predictive setting. Plots with a higher opacity are the ones that the network weighs more.

similar compared to the biases for upwards movements. This time around, the network also prefers the $S = 1$ distribution. Note, however, that this is not coincidental, since the network learns the weights for the distributions for both movements, including probabilities, in the same convolutional branch in Figure 3.6. The other distributions show a similar structure as with the upwards biases, where they still range in [-0.2, 0.2]. This means that, regardless of the movement, the bias stays within the same range, as does roughly the distribution of the bias.

Finally, we turn our attention to Figure 5.9. Here we see the weights for the extracted probability transition matrices of the inputs in the predictive setting. The distribution for $S = 16$ is mostly favored by the model, which shows a bias towards less down movements and more stagnations and upward movements. This would imply that the predictive model will be biased towards stagnations and upwards movements when predicting the data for FI-2010 and might also signal an imbalance in the training data set.

All of these properties are certainly interesting to analyze to further comprehend the inner dynamics of the order books within the training data. This explicitly shows that our model learns useful properties about the input data. Note that even though we have used order books as our targeted data set, the model might also be deployed on other multivariate data sets. The only assumptions that we have made are the ones that came forth from the axioms in Section 3.1, of which the novel loss functions were the results. One might construct their own loss functions depending on the underlying data, or not use any at all, yet still utilize the same models to generate and predict for a multivariate time series.

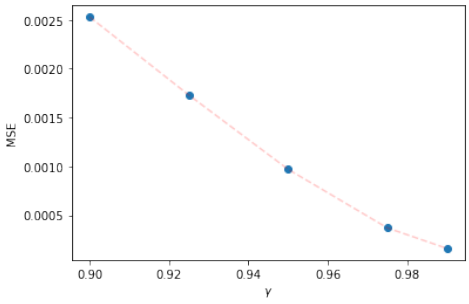

Figure 5.10: Various values of gamma versus the MSE in the sensitivity analysis of OBNet

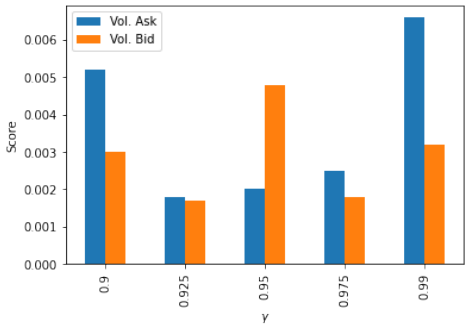

Figure 5.11: The effect of γ on the volume scores

5.3. SENSITIVITY ANALYSIS & ABLATION STUDY

5.3.1. SENSITIVITY ANALYSIS

We shall now discuss the results of our sensitivity analysis and ablation study. We have to note that comparing the models was much easier in the predictive setting compared to the generative setting, since the generative setting considers more than just one output measure.

In the predictive setting, we found the γ variable to be the most influential to the resulting MSE. In particular, a higher value for γ yielded a lower MSE, as depicted in Figure 5.10. This is the result of increased determinism in the model, as was depicted earlier in Figure 3.5. Since the model can be more consistent with its output, it is easier to fit the data in a supervised manner. The model seemed to perform consistently across all other values for S and ξ: their influence on the model was negligible in the predictive setting for the set domains. It is probable that the behavior of the model would have changed if we had searched for more extreme values of S and ξ, but as we will see in the generative setting, this was not needed to see the effect of these variables on the model.

In fact, in the generative setting, the γ variable had the least effect on the results. The most significant changes for a changing γ were in the volume measures, which are depicted in Figure 5.11. In contrast to the predictive setting, a higher γ resulted in worse scores for the volume measures, but so did a too low γ. There exists a sweet spot revolving around $\gamma = 0.95$ for which the model would not be too stochastic and not too deterministic either, where the scores for the volume would be best. This indicates that volume is more sensitive to the temporal stochasticity of the model, compared to the price and returns.

The most interesting cases are now the cases that were not interesting in the predictive setting: the sampling rate ξ and the amount of estimated distributions S.

For a sampling rate too high, the volume scores worsened, while the price and return scores got better, as can be seen from Figure 5.12. This implies that the price and returns favor more cross-level interaction, while the volume variable has a clear bound on the cross-level interaction. Furthermore, the axiom scores, relating to price, seemed at their best around $\xi = 3$. This indicates that a balanced sampling rate, even though resulting in less accurate price and volume distributions, was resulting in structurally more realistic

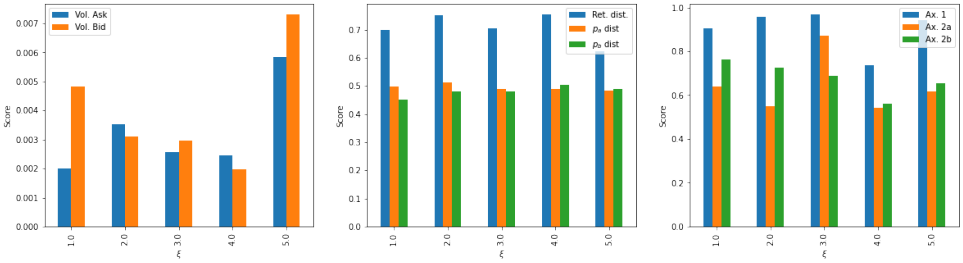

Figure 5.12: From left to right: the effect of the sampling rate ξ on the volume, return and price, and axiom scores

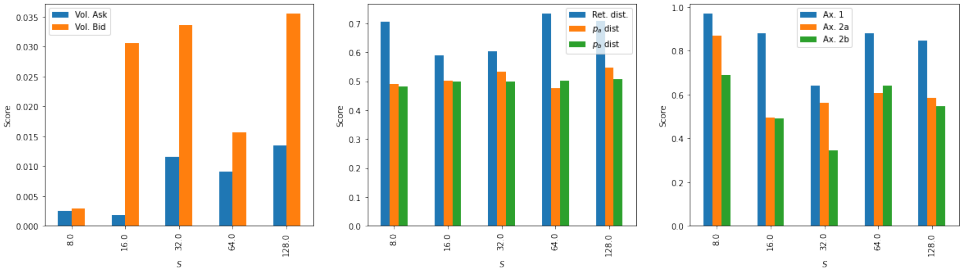

Figure 5.13: From left to right: the effect of the estimated number of distributions S on the volume, return and price, and axiom scores

order books.

Furthermore, looking at the estimated number of distributions S in Figure 5.13, we note a quite interesting case. We see extremely low scores for volume for $S = 8$, however, these scores do not reflect for the price and returns. In fact, the price and returns become lower for $16 \leq S \leq 32$. This indicates that $S < 16$ and $S > 32$ are both underfits and overfits, respectively, on the price and return distributions, while $S > 8$ results in an overfit for the volume distributions. Hence, the number of distributions for price and volume differ in the underlying data. Even though this particular case of S remains interesting, we remain confident that the study of the S variable as a whole remains interesting and encourage future researchers to dig into its properties and analytical derivations. Looking at the axiom scores, we see that they become best around $S = 8$, which even though is an underfit for the prices, still results in more realistic-looking order books. This is the second time we make such an observation, where a variable tuned worse towards the axioms seemed to perform better for the distributions. This shines light upon the idea that there is a strong link between the structural realness and the distributions, which is not accurately captured by the model. This is another interesting case, since this would mean that a structurally real order book comes at the cost of worse distribution approximations, while these two properties seem to be disjoint from each other. This is apparently not the case and calls for further investigation by future researchers as well.

We continue by looking at Figure 5.14, where we have depicted the loss of the generator $\mathscr{L}_G$ over the training iterations. We see that $\mathscr{L}_G$ is quite unstable for $\lambda_{KL} = 0.1$

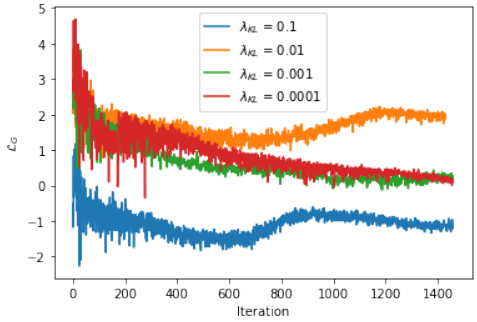

Figure 5.14: The loss of the generator $\mathscr{L}_G$ over the training iterations for various values of λ_{KL}

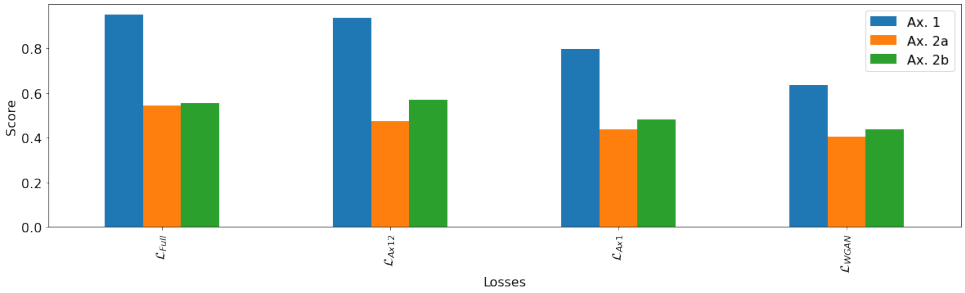

Figure 5.15: Axiom scores for various loss functions

and $\lambda_{KL} = 0.01$. Note that $\lambda_{KL} = 1$ is not included within this plot, because it falls far below all the others, but its structure resembles that of the previously mentioned two. The training for the other two values of λ_{KL} seems to remain stable. The output of the generator remains indifferent of the used value for λ_{KL}, hence, only the training process is altered. Even though we might not notice a difference now, a longer training session might get hurt from a larger value of λ_{KL}, actually resulting in worse output results.

5.3.2. ABLATION STUDY

We finalize our analysis with the ablation study. We analyze the results of the scores while keeping out one of the losses at each run. Initially, we start off with the full loss $\mathscr{L}_{Full}$, remove $\mathscr{L}_{KL}$ to get $\mathscr{L}_{Ax12}$, which contains the WGAN-GP loss, as well as the loss for both axioms. Then we remove the loss for axiom 2 to get $\mathscr{L}_{Ax1}$ and finally remove the axiom 1 loss as well to get $\mathscr{L}_{WGAN}$. The results of our ablation study are listed in Figure 5.15. Note that we only depict the axiom scores: the scores of the return, price, and volume distributions did not change. However, we did get the change in axiom scores that we expected. Not having the axiom losses resulted in a direct decrease in the axiom scores as well. Interesting to see is that removing the axiom 2 loss had an influence on the axiom 1 score as well, and vice-versa. It seems that these two losses work well together.

We also proceed with our analysis from the previous section of λ_{KL} since we can also evaluate the SDMs without $\mathscr{L}_{KL}$ now. Instead of plotting out all SDMs, we will take the

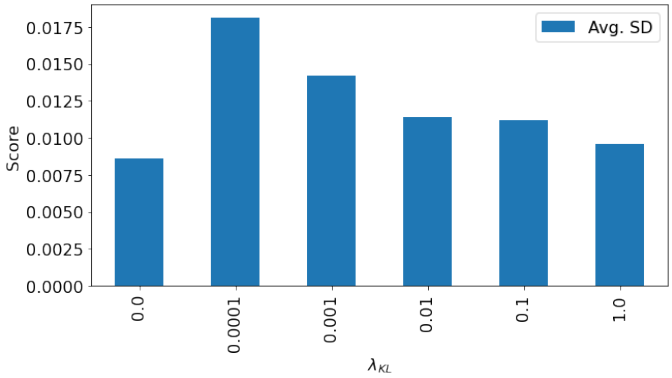

Figure 5.16: The average standard deviation (SD) for different values of λ_{KL} including the ablation of $\mathscr{L}_{KL}$

average value of the SDMs and depict them accordingly in Figure 5.16. We see a clear trend in the average standard deviations (ASD): the smaller λ_{KL}, the higher the ASD. However, completely removing $\mathscr{L}_{KL}$ results in a worse ASD compared to the smallest value for λ_{KL}. We see that the addition of λ_{KL} more than doubles the ASD score, hence improves the diversity of the model's output.

6

Conclusion

In this chapter, we will summarize our work and list the advantages and disadvantages of the work. We will also quite elaborately talk about future work since this work leaves a lot to be discovered.

6.1. SUMMARY

In this work, we aimed at constructing a deep learning-based mathematical model for the goal of learning the inner dynamics of the limit order book. This model was constructed with interpretability and explainability in mind and as a result, we acquired a model that was able to capture interesting features from the underlying data, both in the generative and predictive setting. Although the OBGAN model had the same number of parameters in the generative setting, it had an order of magnitude fewer parameters in the predictive setting as OBNet. In the generative setting, the model outperformed both WGAN and DCGAN in almost every aspect, except for approximating the volume distributions, in which all models performed similarly. In the predictive setting, the OBGAN model outperformed both WGAN and DCGAN and the OBNet model performed better or similar to the other models. The most interesting takeaway is that the model performed consistently in the case of transfer learning, where we tested the model on a different data set than it was trained on. This implies that the model was able to capture an underlying mechanic that was both data sets had in common, which the other models were not able to show.

In addition, we have shown that our model, whether it be OBGAN or OBNet, was able to capture interesting features that might be used by traders or algorithms to make sense of the results, since the rates and transition matrices are explicitly defined in both models.

Through our sensitivity analysis, we have found that our model's parameters influence the variables of the order book. This analysis partially confirmed our hypothesis about the model hyperparameters and also gained us more insight into the underlying data. It also shed light on concepts that we were not aware of, but revealing those concepts through our analysis opened up new pathways for future researchers.

6.2. STRENGTHS, WEAKNESSES, OPPORTUNITIES AND THREATS

We finally summarize the model through a SWOT analysis. This analysis proves to be useful since it provides a clear view of what the model is capable of and of what it is not.

We start off with the strengths of the model and the work:

- it is able to deliver accuracy, as well as extracting useful features from the underlying spatio-temporal time series;
- it is the first work, to our knowledge, to implement both generation and prediction through the use of the same core model for LOBs;
- novel LOB specific losses are implemented and novel measures to assess the quality and diversity of the generated order books are proposed for other researchers to use;

Furthermore, we continue with the weaknesses of our model:

- although many interactions are modeled explicitly, the model was not able to accurately capture the link between structural realness and the quality of the generated distributions;

- the variables of L and T, as well as T_{base} and T_{pred}, have been fixed throughout the study in order to reduce the complexity of the model. This, however, comes at the cost of not being able to measure the performance of the model for larger or short time horizons and different depths of the order books;
- although the model has interpretable parts in it, it is still unclear, without further analysis, what the branches in the network exactly learned that resulted in its outputs. However, as shown in Sections 5.1.2 and 5.2.2, we are able to retrieve information about the inner workings of the mathematical model that do tell us more about the final result of the model. In theory, we have mapped the outputs of the network to a more interpretable model from which we can read off useful features of the data set.

Then, we proceed to the opportunities of the model:

- the model does not make explicit assumptions, except for the proposed losses, about LOBs. Although not explored, this means that the model can be used for other spatio-temporal time series as well, while it remains accurate and transparent;
- the model offers a flexible sampling function, which can be used to model prior knowledge about the underlying data and to test assumptions through it;
- the proposed deep learning architecture is, compared to other deep learning research, quite shallow. A deeper and perhaps wider network might result in state-of-the-art performance in spatio-temporal time series prediction, while the model still extracts useful features from the underlying data.

Finally, we list the threats to our model:

- the financial market keeps evolving quite rapidly. Even though we have used data from a relevant cryptocurrency, it might not reflect the actual data from stock exchanges, which we were not able to access. The model might not be able to keep up with other and perhaps even future data sets;
- work in the machine learning field is often not published due to its potential earnings. A party finding out about a change or addition to the proposed model to deliver better results will likely not publish it, slowing down the progress in the field as they do. In addition, people using this model will make their steps predictable by others as the model is a public one.

6.3. FUTURE WORK

This work still leaves a lot of future work to be done, where the most interesting case lies in using this model on other multivariate time series, different from order books, since the general model does not make any assumptions regarding the underlying data. The model that we have constructed utilizes four variables, namely the ask price p_a, bid price p_b, the ask volume v_a, and the bid volume v_b. However, these variables are not

necessarily modeled explicitly in our formalization. Any combination of variables could be used, where a correlation between variables can be modeled through the sampling function. The sampling function, regardless of its applied domain, is still a subject worthy of further analysis. There are many possibilities for the sampling function and there are many situations in which we might prefer one sampling function over the other.

The opposite could also be done, namely to extend the current model by adding additional axioms that relate to volumes. There are ample empirical observations on the behavioral patterns of v_a and v_b. Along with new axioms, new loss functions can be constructed and the sampling function might be altered to differ per variable, to take different behaviours into account.

Furthermore, the variable S, representing the number of learned distributions for both rates and probabilities, is acquired through a search for the best performing model in our work. However, it should be possible to create a model that is able to approximate S by searching for different means and variances in the underlying data, where perhaps a search for the former would be enough. The research on S does not stop there, however. It is still a question on how S and the other hyperparameters influence both the axiom scores and the approximated return and price distributions in a counter-intuitive way.

In addition, the models that we have deployed are quite shallow compared to state-of-the-art models. This is the result of a lack of computational resources. Future research might hence utilize a deeper and wider network to acquire better results.

BIBLIOGRAPHY

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[2] Samuel Assefa. "Generating synthetic data in finance: opportunities, challenges and pitfalls". In: *Challenges and Pitfalls (June 23, 2020)* (2020).

[3] Marco Avellaneda and Sasha Stoikov. "High-frequency trading in a limit order book". In: *Quantitative Finance* 8.3 (2008), pp. 217–224.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[5] Bruno Biais, David Martimort, and Jean-Charles Rochet. "Competing mechanisms in a common value environment". In: *Econometrica* 68.4 (2000), pp. 799–837.

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

[7] Stephen Boyd et al. "Fastest mixing Markov chain on graphs with symmetries". In: *SIAM Journal on Optimization* 20.2 (2009), pp. 792–819.

[8] Jou-Fan Chen et al. "Financial time-series data analysis using deep convolutional neural networks". In: *2016 7th International conference on cloud computing and big data (CCBD)*. IEEE. 2016, pp. 87–92.

[9] Min Chen. "Mixing time of random walks on graphs". In: *Mathematics Department, University of York* (2004).

[10] Rama Cont and Marvin S Mueller. "A stochastic PDE model for limit order book dynamics". In: *arXiv preprint arXiv:1904.03058* (2019).

[11] Rama Cont, Sasha Stoikov, and Rishi Talreja. "A stochastic model for order book dynamics". In: *Operations research* 58.3 (2010), pp. 549–563.

[12] Antonia Creswell et al. "Generative adversarial networks: An overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65.

[13] Jonathan Doering, Michael Fairbank, and Sheri Markose. "Convolutional neural networks applied to high-frequency market microstructure forecasting". In: *2017 9th Computer Science and Electronic Engineering (CEEC)*. IEEE. 2017, pp. 31–36.

[14] Thierry Foucault, Ohad Kadan, and Eugene Kandel. "Limit order book as a market for liquidity". In: *The review of financial studies* 18.4 (2005), pp. 1171–1217.

[15] Nan Gao et al. "Generative adversarial networks for spatio-temporal data: A survey". In: *arXiv preprint arXiv:2008.08903* (2020).

[16] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[17] Ian J Goodfellow et al. “Generative adversarial networks”. In: *arXiv preprint arXiv:1406.2661* (2014).

[18] Martin D Gould et al. “Limit order books”. In: *Quantitative Finance* 13.11 (2013), pp. 1709–1742.

[19] Mohit Goyal, Rajan Goyal, and Brejesh Lall. “Learning Activation Functions: A new paradigm for understanding Neural Networks”. In: *arXiv preprint arXiv:1906.09529* (2019).

[20] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *arXiv preprint arXiv:1704.00028* (2017).

[21] E Han Kim and Vijay Singal. “Stock market openings: Experience of emerging economies”. In: *The Journal of Business* 73.1 (2000), pp. 25–66.

[22] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[23] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent”. In: *Cited on* 14.8 (2012).

[24] Saifuddin Hitawala. “Comparative study on generative adversarial networks”. In: *arXiv preprint arXiv:1801.04271* (2018).

[25] Jui-Chieh Huang et al. “Applying a Markov chain for the stock pricing of a novel forecasting model”. In: *Communications in Statistics-theory and Methods* 46.9 (2017), pp. 4388–4402.

[26] Ferenc Huszár. “How (not) to train your generative model: Scheduled sampling, likelihood, adversary?” In: *arXiv preprint arXiv:1511.05101* (2015).

[27] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[28] Maximilian Jaritz et al. “Sparse and dense data with cnns: Depth completion and semantic segmentation”. In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 52–60.

[29] Daniel Jerison. “General mixing time bounds for finite markov chains via the absolute spectral gap”. In: *arXiv preprint arXiv:1310.8021* (2013).

[30] Jonathan M Karpoff. “The relation between price changes and trading volume: A survey”. In: *Journal of Financial and quantitative Analysis* (1987), pp. 109–126.

[31] Alec N Kercheval and Yuan Zhang. “Modelling high-frequency limit order book dynamics with support vector machines”. In: *Quantitative Finance* 15.8 (2015), pp. 1315–1329.

[32] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[33] Junyi Li et al. “Generating realistic stock market order streams”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 727–734.

[34] Qi Li. “Market Opening and Stock Market Behavior: Taiwan’s Experience”. In: *International Journal of Business and Economics* 1.1 (2002), p. 9.

[35] 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*. 2015, pp. 3431–3440.

[36] Jacob Loveless. “Barbarians at the Gateways”. In: *Communications of the ACM* 56.10 (2013), pp. 42–49.

[37] Jaime Niño et al. “CNN with Limit Order Book Data for Stock Price Prediction”. In: *Proceedings of the Future Technologies Conference*. Springer. 2018, pp. 444–457.

[38] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. “f-gan: Training generative neural samplers using variational divergence minimization”. In: *arXiv preprint arXiv:1606.00709* (2016).

[39] Adamantios Ntakaris et al. “Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods”. In: *Journal of Forecasting* 37.8 (2018), pp. 852–866.

[40] Christine A Parlour. “Price dynamics in limit order markets”. In: *The Review of Financial Studies* 11.4 (1998), pp. 789–816.

[41] Sheldon M Ross. “Variations on Brownian motion”. In: *Introduction to Probability Models* (2014), pp. 612–614.

[42] Tim Salimans et al. “Improved techniques for training gans”. In: *arXiv preprint arXiv:1606.03498* (2016).

[43] Javier Sandoval. “Empirical shape function of the Limit-Order books of the USD/COP spot market”. In: (2013).

[44] Javier Sandoval et al. “Detecting informative patterns in financial market trends based on visual analysis”. In: *Procedia Computer Science* 80 (2016), pp. 752–761.

[45] Divya Saxena and Jiannong Cao. “D-gan: Deep generative adversarial nets for spatio-temporal prediction”. In: *arXiv preprint arXiv:1907.08556* (2019).

[46] Rinaldo B Schinazi. “Stationary Distributions for Discrete Time Markov Chains”. In: *Classical and Spatial Stochastic Processes*. Springer, 2014, pp. 105–129.

[47] Pierre Sermanet, Soumith Chintala, and Yann LeCun. “Convolutional neural networks applied to house numbers digit classification”. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE. 2012, pp. 3288–3291.

[48] Andrea Sgarro. “Informational divergence and the dissimilarity of probability distributions”. In: *Calcolo* 18.3 (1981), pp. 293–302.

[49] John R Silvester. “Determinants of block matrices”. In: *The Mathematical Gazette* 84.501 (2000), pp. 460–467.

[50] Justin A Sirignano. “Deep learning for limit order books”. In: *Quantitative Finance* 19.4 (2019), pp. 549–570.

[51] Oliver K Smith. “Eigenvalues of a symmetric 3× 3 matrix”. In: *Communications of the ACM* 4.4 (1961), p. 168.

[52] Lucas Theis, Aäron van den Oord, and Matthias Bethge. “A note on the evaluation of generative models”. In: *arXiv preprint arXiv:1511.01844* (2015).

[53] Avraam Tsantekidis et al. “Forecasting stock prices from the limit order book using convolutional neural networks”. In: *2017 IEEE 19th Conference on Business Informatics (CBI)*. Vol. 1. IEEE. 2017, pp. 7–12.

[54] Zhiguang Wang and Tim Oates. “Imaging time-series to improve classification and imputation”. In: *arXiv preprint arXiv:1506.00327* (2015).

[55] Maciej Wiatrak, Stefano V Albrecht, and Andrew Nystrom. “Stabilizing Generative Adversarial Networks: A Survey”. In: *arXiv preprint arXiv:1910.00927* (2019).

[56] Magnus Wiese et al. “Quant gans: Deep generation of financial time series”. In: *Quantitative Finance* 20.9 (2020), pp. 1419–1440.

[57] Yuxin Wu and Kaiming He. “Group normalization”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[58] Zihao Zhang, Stefan Zohren, and Stephen Roberts. “Deeplob: Deep convolutional neural networks for limit order books”. In: *IEEE Transactions on Signal Processing* 67.11 (2019), pp. 3001–3012.

[59] Xingyu Zhou et al. “Stock market prediction on high-frequency data using generative adversarial nets”. In: *Mathematical Problems in Engineering* 2018 (2018).

[60] Eric Zivot and Jiahui Wang. “Vector autoregressive models for multivariate time series”. In: *Modeling Financial Time Series with S-Plus®* (2006), pp. 385–429.