Tampere University

Klaus Uhle

PREDICTING STOCK MARKET LIQUIDITY USING NEURAL NETWORKS

Faculty of Engineering and Natural Sciences
Master of Science Thesis
October 2020

ABSTRACT

Klaus Uhle: Predicting Stock Market Liquidity Using Neural Networks
Master of Science Thesis
Tampere University
Industrial Engineering and Management
October 2020

This thesis proposes a long-short term memory prediction model for stock market liquidity. The prediction task was defined as a time series regression problem of the next step limit order book quantity. Level 1 depth and multi-level depth of the limit order book were used as a measure of the stock liquidity. The objective of the thesis was to research the prediction capabilities of neural networks in this prediction task.

Several popular neural networks were investigated for time series prediction and stock quantities from NASDAQ stocks were analyzed to build long-short term memory prediction model. The used dataset included intraday limit order book data from five stocks during five full trading days in 2014. The used stocks were Apple, Facebook, Google, Intel, and Microsoft. The prediction model was first optimized using Apple stock data and then tested with all the stocks. The performance of the long short-term memory prediction model was compared against a naïve prediction model that was used as a benchmark.

The long short-term memory prediction model performed better than the benchmark model in the case of the multi-level depth liquidity prediction for Apple stock. Level I depth prediction was not found suitable for the regression prediction task. The long short-term memory prediction model proved prediction capabilities only for the Apple stock that was used to optimize the model, but it was not able to generalize the prediction capability for the other stocks. For them, the naïve model outperformed the long short-term memory prediction model. This thesis provides evidence of the prediction capability for the optimized neural network prediction model but does not show any generalization capability.

Key words: Neural networks, limit order book, stock market liquidity, time series prediction

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Klaus Uhle: Osakemarkkinoiden likviditeetin ennustaminen neuroverkoilla
Diplomityö
Tampereen Yliopisto
Tuotantotalouden tutkinto-ohjelma
Lokakuu 2020

Tämä diplomityö esittää uuden pitkän lyhytaikaisen muistin neuroverkkoennustemallin osakemarkkinoiden likviditeetin ennustamista varten. Ennustustehtäväksi määritettiin regressiivinen aikasarjaennuste, jossa tavoitteena on ennustaa sekä tarjouskirjan ensimmäisen tason tarjousten määrää, että useamman tason tarjousten määrää seuraavan tarjouskirjatapahtuman hetkellä. Diplomityön tavoitteena oli tutkia neuroverkkojen ennustuskykyä tässä ennustetehtävässä.

Työssä tutkittiin useiden tunnettujen neuroverkkojen sopivuutta NASDAQ:in osakkeiden tarjousmäärien aikasarjaennustamiseen, minkä perusteella päädyttiin rakentamaan pitkän lyhytaikaisen muistin ennustemalli. Käytetty tietoaineisto sisälsi viiden eri osakkeen päivänsisäistä tarjouskirjadataa viiden päivän ajalta vuodelta 2014. Tietoaineistoon kuuluvat osakkeet olivat Apple, Facebook, Google, Intel ja Microsoft. Ennustemalli optimoitiin käyttämällä Applen osakedataa ja myöhemmin mallia testattiin käyttämällä muiden osakkeiden dataa. Lopuksi pitkän lyhytaikaisen muistin ennustemallin suorituskykyä verrattiin naiiviin ennustemalliin.

Pitkän lyhytaikainen muistin ennustemalli suoriutui ennustamisesta naiivia ennustemallia paremmin Applen osakkeen tarjouskirjan usean tason tarjousten määrää ennustettaessa. Ensimmäisen tason ennustaminen ei sen sijaan soveltunut regressiotehtäväksi. Pitkän lyhytaikainen muistin ennustemalli osoitti kykyä ennustaa vain Applen osakkeen datalla, jota oli myös käytetty ennustemallin optimoinnissa. Se ei kuitenkaan kyennyt yleistämään ennustuskykyä muille osakkeille, vaan naiivi ennustemalli suoriutui paremmin niiden ennustamisessa. Tämä diplomityö osoittaa ennustuskykyä optimoidulle neuroverkkomallille, mutta ei osoita sen yleistettävyyttä.

Avainsanat: Neuroverkot, tarjouskirja, likviditeetti, aikasarjaennustaminen

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

PREFACE

I want to especially thank my supervisor Juho Kanniainen for giving me this opportunity to work on this thesis and for giving me comprehensive guidance. Another big thanks to Adamantios Ntakaris and Dat Tran for supporting me with the data understanding and the model development. Finally, thanks to my friends and colleagues for sharing ideas and inspiring machine learning conversations during lunchtime and after work hours.

Helsinki, 28th October 2020

Klaus Uhle

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

AAPL	Ticker for Apple stock
ANN	Artificial neural network
CNN	Concurrent neural network
FB	Ticker for Facebook stock
FNN	Feedforward neural network
GOOG	Ticker for Google stock
IFRS	International financial reporting standards
INTC	Ticker for Intel stock
LSTM	Long short-term memory
LOB	Limit order book
MAE	Mean average error
MLP	Multi-layer perceptron
MSE	Mean squared error
MSFT	Ticker for Microsoft stock
RNN	Recurrent neural network
OTC	Over-the-counter
SLP	Singe-layer perceptron
XLM	Exchange liquidity measure

b	bias term
C_t	cell state
D_t	depth at time t
f_t	forget gate
h_t	hidden state
I	number of input nodes
i_t	input gate
J	number of hidden nodes
L	loss function
μ	learning rate
∇	gradient
o_t	output gate
q_t	quantity at time t
$\sigma(x)$	non-linear activation function
S	bid-ask spread
s	standard deviation
U	hidden weight matrix
W	weight matrix
z	z-score normalization

.

1. INTRODUCTION

Liquidity plays a central role in the operation of financial markets. It can mean different things, but in the stock markets, the most important dimension of liquidity is the ease with which market participants can buy or sell stocks, or the ability of stock markets to absorb purchases or sales of large quantities without any remarkable effect on the stock prices. (Geithner 2007) Investors find stock market liquidity important, because it affects the return on their investment as illiquid stocks cost more and sell for less. Because of the imperfect markets, stocks are always illiquid to some extent, which can be observed in the bid-ask spread: the stocks cannot be bought and sold for the same price. Thus, the analysis of liquidity changes is crucial for asset managers as well as for ordinary investors in evaluating their trading activities. In addition to increased trading costs, the illiquidity is a source of risk for investors. (Faucalt et al. 2013, pp. 5) Investors will thus require compensation not only for the trading costs, but also for the additional risks related to the stock market liquidity. Analysis of stock market liquidity is a required tool to support investment decisions.

The stock market liquidity can be observed in the limit order book (LOB). As LOB is dependent on the past versions, the researchers have widely examined the LOB with the objective to predict the future market movements. The high-frequency limit order book is an intriguing research area due to the complex behavior of the financial markets and the possible gains and trading strategies yielded from the findings. Mid-price and mid-price movement prediction have been the most popular research topics (Palguna et al. 2016; Dixon 2016; Nousi et al. 2019). While the price component of the LOB is widely researched, the quantity component has received less attention. One previous research examined on how liquidity in LOB evolves around scheduled and nonscheduled company announcements (Siikanen et al. 2017). Other research topics have covered liquidity supply predictions, volatility changes and multi-level order-book imbalance (Elezovic 2009; Härdle et al. 2012; Kang 2018; Xu et al. 2019). The LOB data has also been used in the creation of a reinforcement learning agent to perform high-frequency trading (Wang et al. 2019).

The advances in computer science and technology have enabled fully automated high-frequency trading, boosting the profitability of the trading departments, and attracting interest in developing the technology further. In the last decade, with the development of

machine learning and time series prediction, methods have been shifting from statistical parametric models to data-driven machine learning approaches. Artificial neural networks have found success in numerous pattern recognition and machine learning contests, and they have proven to overcome the challenges represented with statistical models (Schmidhuber 2014). As statistical models often make unrealistic assumptions about the distribution of the data, machine learning techniques does not make any assumptions at all (Nousi et al. 2019). This increases in importance in the area of financial time series prediction, since LOB is a complex, dynamic and high dimensional entity, leading to modelling challenges that make statistical methods hard to cope with (Zhang et al. 2019). As the trading continues for full trading hours, the LOB includes a huge amount data that is constantly changing with the possibility of crucial patterns forming and deforming too quickly for human to observe (Kercheval & Zhang 2015). Automation and technical development of financial markets has significantly made information analysis more complex, which creates a demand for more complex models as well. (Ntakaris et al. 2019).

Artificial neural networks are machine learning algorithms. They have adaptive learning behavior, where the algorithm is capable of learning from previous samples and adapting to new input parameters. (Kyaw & Xiang-Qun 2015) These algorithms are especially useful for handling non-linear behavior such as financial time series (Speck-Planche & Cordeiro 2015). Financial time series predictions are considered challenging prediction tasks since there is not any straightforward method to define the nature of the financial time series. For example, a time series used to describe stock index is generally non-linear, non-Gaussian, non-deterministic and non-stationary. (Konar & Bhattacharya 2017, pp. 2) Artificial neural networks are one of the few methods that have found broad success in time series prediction (Graves 2012).

1.1 Objective of the thesis

The topic of this thesis is to predict stock market liquidity using neural networks. Even though stock market liquidity and artificial neural networks are widely researched separately, there is not any existing research combining these two topics. In the previous research, some statistical prediction models are presented. However, to the best of knowledge, none of the existing research has predicted stock market liquidity using neural networks. The objective of this thesis is to develop prediction models using neural networks to predict stock market liquidity with LOB data and to clarify the prediction capabilities of neural networks in liquidity prediction. This thesis aims to answer the two main research questions:

RQ1: Can stock market liquidity be predicted using neural networks with LOB data?

RQ2: How does the proposed model perform against the naïve prediction model?

The first research question underlines the novelty of this thesis and the approach is to search widely and experiment different prediction models to find evidence for prediction capabilities. The second research question looks to evaluate the findings against the simple prediction model to find out the scale of prediction capabilities, if any. This thesis aims to describe the final solution in detail to enable reproduction attempts. Also, to help further research, the challenges of the prediction task and model development are explained to a greater extent.

1.2 Organization of the thesis

The structure of the thesis is the following: First, Chapter 2 provides the relevant theory background related to the stock market liquidity and LOB. Then, Chapter 3 gives a brief introduction of time series prediction following a discussion of machine learning and artificial neural networks. After that, Chapter 4 describes the data used in the modelling. It is followed by Chapter 5 analyses on the built models and prediction results. Finally, Chapter 6 concludes the thesis.

2. LIMIT ORDER BOOK AND MARKET LIQUIDITY

2.1 Limit order book

In market microstructures, trading mechanisms are separated to order-driven and quote-driven markets. Quote-driven markets are exchange systems in which market makers provide the buy and sell prices to the investment products. Markets for government bonds, currencies and commodities are usually quote-driven. In an order-driven market, prices are created by market participants posting orders to buy and sell the investment products. The orders made by market participants are then submitted to a LOB by market makers. (Schwarz & Weber 1997) Stock markets are typically order-driven, for example, New York Stock Exchange (NYSE) is an order-driven market.

In the order-driven markets, traders post two kind of orders, limit orders and market orders. A limit order is an order to trade at the price level that equals or is better than the limit price. The trade will not be executed, if market price does not reach the limit price. This means that the limit orders have execution uncertainty. (Yang et al. 2016) A market order is an order that is executed instantly with the best possible price, assuming that the LOB is not empty. The best possible price is only determined at the time of execution, which means that market orders have price uncertainty. LOB includes all the outstanding limit orders, which are presented aggregated and are available for market participants. The limit orders remain in the LOB until they are executed against market order or cancelled. The LOB is a dynamic environment, and it is constantly changing due the new orders and cancellations during the trading hours.

In the order-driven stock markets, a limit order is an order to sell or buy a certain number of shares with a certain price. There are two sides in the LOB, one for ask orders and one for bid orders, and the two main components are price and volume. The ask order is an order to sell and the bid order is an order to buy. The LOB is constructed as follows. The bid and ask limit orders are on the different sides of the LOB and they are sorted by the price so that the levels are separated from each other by at least the minimum tick size. The level 1 is the best price level, meaning the highest bid price and the lowest ask price. The quantity of the order is a natural number of the stocks that the market participant is willing to trade. If more than one limit order has the same price, their quantities are summarized together. LOB consist of all the outstanding orders until they are executed or cancelled. The structure of top five levels of a LOB is illustrated in Figure 2.1

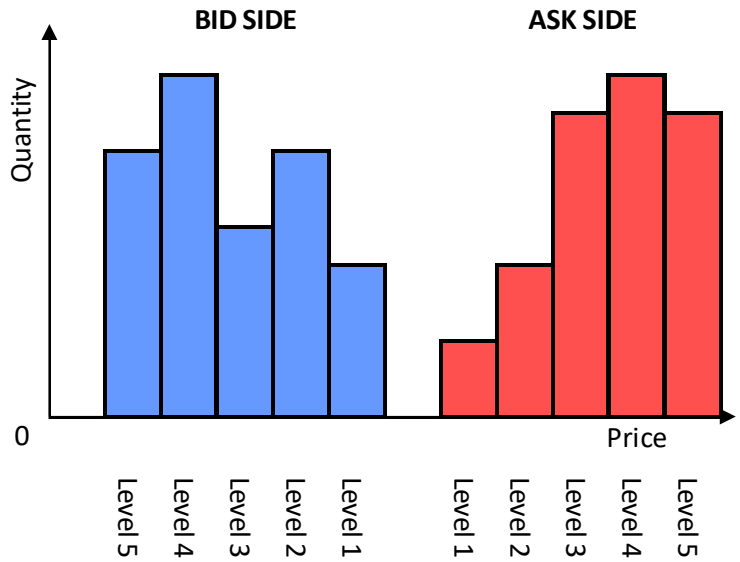

Figure 2.1. *Structure of top five levels of the LOB.*

The level 1 on the bid side consist of the orders that have the highest price, and the sum of their quantities aggregate the level 1 bid quantity. Similarly, the level 1 on the ask side consists of the orders that have the lowest price, and their sum is the level 1 ask quantity. The bid-ask spread is the difference between the level 1 prices, represented by the gap between the ask and bid side. An example of a LOB of Apple stock is represented in Figure 2.2.

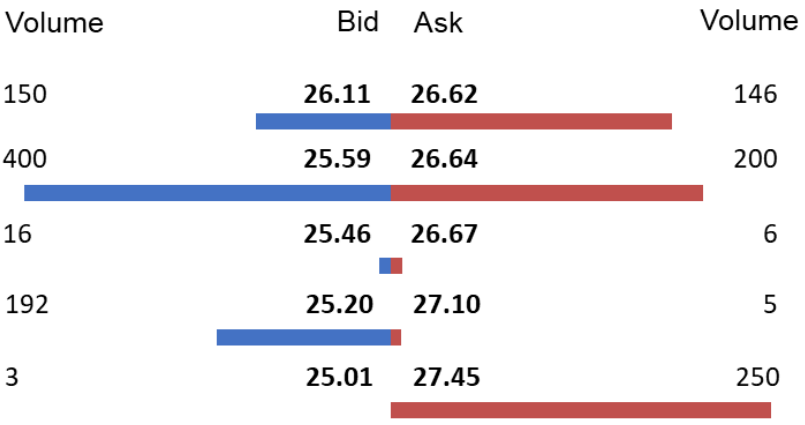

Figure 2.2*. Top five levels of the LOB of Apple stock on April 14, 2014.*

This is the representation of LOB that trading platforms usually use. The bid side of the LOB is represented in the left side, ask side being on the right. The bid-ask spread can be calculated from the top row, being currently 0.51. The bars represent the relative size of the order quantities.

The advantage of the order-driven market compared to the quote-driven markets is that it accepts both, limit orders and market orders (Zhang et al. 2019). When a market order

is submitted, the buy or sell action happens instantly at the best currently available price. On the contrary, when submitting a limit order, the trader is willing to buy or sell a financial instrument under a certain price, but the trade is left unexecuted. The limit orders represent unexecuted trading activity until a matching market order arrives or the trade is cancelled. The unexecuted limit orders are divided into levels, which then construct the LOB. The LOB accepts orders from market participants. Market participants who submit limit orders provide liquidity to the market and market participants who submit market orders consume liquidity from the market. (Ntakaris et al. 2019) Contrarily, liquidity is the advantage of quote-driven market since the market makers are required to transact on their quoted prices so there is a guarantee of order fulfillment.

2.2 Stock market liquidity

In the perfect markets, buyer and sellers immediately find each other and all the trades are executed instantly at frictionless prices, generating infinite market liquidity. Real markets however fall short on delivering such efficiency. Market participants can be constrained by reasons such as conflicts of interests, information asymmetry and fragmented markets. (Biais et al. 2016) Due the constrains, all the orders are not instantly executed, but instead they are submitted to LOB as outstanding limit orders.

Liquidity is the stock market's feature whereby a market participant can sell or buy an asset without having a far-reaching effect in the stock's price. In a liquid stock market, selling quickly and large amounts of stocks cannot be observed from the stock's price. On the contrary, in an illiquid market, selling quickly and large amounts will require cutting the price by observable amount. The same logic applies when buying quickly and large volumes. In a liquid market, buying quickly and large volumes will not increase the price much and in an illiquid market buying quickly and large volumes will require an increased price by some amount. Real markets are always illiquid to some extent, since buying a stock and selling it immediately would cause loss even without trading costs because of the bid-ask spread. In a stock market, the market liquidity can be observed in the LOB. Liquidity is supplied to the LOB by inflow of limit orders provided by market participants. The non-executed orders constitute the LOB, the consolidated source of liquidity. (Frey & Gamming 2005) According to Black (1971), liquid stock market should hold the following conditions:

1. There are always bid and ask prices for the investor who wants to buy or sell a small amount of stocks immediately.
2. The difference between the bid and ask prices is always small.

3. An investor who is buying or selling a large amount of stock, in the absence of special information, can expect to do so over a long period of time at a price not very different, on average, from the current market price.
4. An investor can buy or sell a large block of stock immediately, but at a premium or discount that depends on the size of the block.

In stock market, liquidity is supplied by the limit orders. The second condition by Black means that there should always be limit orders in the LOB in liquid market. The liquidity increases as limit orders are submitted and the liquidity decreases as market orders as submitted and the limit orders are executed. The liquidity supply is dependent of the quantities and prices of the limit orders. The higher the volume and the smaller the spread, the greater the liquidity supply. When the liquidity supply is high, the market participant can buy or sell the desired quantity of shares immediately with the best price. If the liquidity supply is lower, the market participant will not be able to buy or sell the desired quantity with the best price. In this scenario, the market participant would have to wait until more limit orders flow in or buy or sell the desired quantity with lower price level, which is naturally a less favorable condition for market participants. The lower liquidity supply is creating challenges for market participants and they may have pressure to complete the orders and avoid decreasing their profits. One instance of the lowering profit is when the lower liquidity supply forces the market participants to perform multiple transactions instead of one, which will increase the transaction costs.

All the liquidity is not observable in the stock market because there exists hidden liquidity that is not visible in the LOB. There are two reasons for hidden liquidity. The first one is the fragmented markets. Liquidity is often posted on various stock exchanges and other markets may intervene such as dark trading that happens outside visible trading books and over-the-counter (OTC) trading (Degryse et al. 2014). The second reason is the behavior of the existing trading algorithms. The algorithms employ trading strategy that consists of splitting orders with large quantities into small batches that replenish the orders immediately as they are executed. They are also referred as iceberg orders for the purpose of hiding the actual order quantity. (Stoikov et al. 2013)

At some point, liquidity is a basic need of all investors, but some investors have more essential need for liquidity than the others. For example, investors with long maturity liabilities have less liquidity risk, since the risk of suddenly requiring transaction immediacy in short term is relatively low. These investors could then be able to collect premium by supplying liquidity for investors who need it short term. (Chacko et al. 2016)

During this millennium, equity markets have experienced revolutionary institutional and technological changes. Decimalization leading to a smaller tick size, increased amount of algorithmic trading, explosions in sub-second order submission and cancellation, and trading volume in general have increased the liquidity but also changed the nature of the stock market liquidity. (Barahedi et al. 2016) The changes in the market liquidity have changed transaction costs and premiums demanded by traders, and it has been a driver of the development of new trading strategies.

2.3 Liquidity measures

Liquidity is a complex concept with multiple dimensions and there exists multiple ways to measure it. In the literature, there have been distinguished four dimensions of the liquidity, which are width, immediacy, resiliency, and depth. However, the interrelation of these dimensions creates challenges when trying to measure liquidity. (Verlag 2008)

Width is defined as the available bid-ask spread S, that is the difference between the best bid price and the best ask price

$$S = p_1^{ask} - p_1^{bid},$$

where p is the price of the stock. Assuming that the median value between the best ask and bid prices reflect the fair value of the asset, the half spread can be interpreted as a transaction cost. (Hachmeister 2007, pp. 22) This measure is used for example by Rösch & Kaserer (2013) where they use EURIBOR-EONIA-spread in their research as a measure of funding liquidity.

Immediacy is the trading time that can be defined as the speed at which trades will be executed with given cost. Immediacy then reflects the ability to sell or buy quickly. It can be measured in multiple ways such as the waiting time between two subsequent trades, number of trades per time unit or the time until the order is completely executed at the certain price. (Barardehi et al. 2016)

Resilience is the dynamic of how prices react to new information or to new order volumes. The resilience describes the speed at which prices return to former levels after liquidity shock such as a large transaction, assuming that there is not any change in the underlying asset value. Resilience is the markets ability to recover from the shock. It can be measured as price-volume elasticity of a given asset. The resilience can only be calculated over a time period. (Hachmeister 2007, pp. 22)

Depth is the volume of stocks demanded at the bid side (bid depth) and the volume of shares provided at the best ask side (ask depth). The sum of bid depth and ask depth is referred as depth D_t

$$D_t = q_t^{ask} + q_t^{bid},$$

where *q* is the quantity of the limit orders. The depth can be measured on multiple levels of the LOB. The level 1 depth is the level 1 quantity of the LOB and it could be interpreted as order the market can absorb without evoking a price change. Multi-level depth will capture wider view of the liquidity than just the level 1. The measurement concept of market depth is one of the most common approaches to measure liquidity. (Pristas 2007)

In this thesis, the level 1 depth and multi-level depth are used to measure of a stock liquidity. Level 1 depth consists of level 1 bid depth and level 1 ask depth, which are the best bid and ask quantities and referred as level 1 bid and ask quantities. For the level 1 quantity to change, one of these has to happen:

1. A new market order is posted. A new market order will decrease the quantity of the level 1 by the quantity of the new market order. The new level 1 quantity would then be $q_1^{new} = q_1^{old} - q_m$. However, if q_m is large enough and q_1^{new} would become negative, it means that all the q_1^{old} limit orders got executed. Then the new level 1 quantity will be the highest level that did not got fully executed $q_1^{new} = q_n^{new}$.
2. A new limit order is posted with the same price as the level 1 price. A new limit order with the same price will increase the quantity of the level 1 by the quantity of the new limit order. The new level 1 quantity will then be $q_1^{new} = q_1^{old} + q_l$.
3. A new limit order is posted with better price than the current price. A new limit order with better price than the current level 1 price will then become the new level 1 quantity. The new level 1 quantity will be $q_1^{new} = q_l$.

The level 1 quantity captures a wide range of the market movements as well as the liquidity supply changes, which is why it is used as one liquidity measure in this thesis.

However, while level 1 depth and bid-ask spread are often used as a liquidity measure, the literature has also pointed out problems in using only the level 1 depth. Since liquidity is multidimensional concept, it might be problematic to use only one dimension of the liquidity as a liquidity measure. Some information of the LOB depth is ignored while focusing only on level 1. For some investors, it is important to measure LOB liquidity across multiple price levels to capture depth and width when trading in large quantities. It be-

comes more important as new information arrives, since the need for depth and immediacy rises, which makes it relevant to measure multi-level depth of the LOB. (Degryse et al. 2014; Siikanen 2018, pp. 14)

Multi-level depth is used to capture wider view of the LOB, instead of just the level 1, which may offer too narrow view of the liquidity. That is why cumulative depth over multiple levels is also used as a liquidity measurement. In this thesis, in addition to the level 1 depth, cumulative depth of five levels of the LOB is also used as a liquidity measurement. The used multi-level depth is the sum of the five top levels of the LOB quantities of both sides. Level 5 depth is the cumulative sum of the quantities at the five top levels

$$D_5^{ask} = q_1^{ask} + q_2^{ask} + q_3^{ask} + q_4^{ask} + q_5^{ask}$$

$$D_5^{bid} = q_1^{bid} + q_2^{bid} + q_3^{bid} + q_4^{bid} + q_5^{bid}.$$

The multi-level depth used as a liquidity measure is the sum of the level 5 ask depth and level 5 bid depth

$$D_5 = D_5^{ask} + D_5^{bid}.$$

More sophisticated multi-level measurements would be Exchange Liquidity Measure (XLM) and separating quantities to different levels with fixed prices (Siikanen 2018, pp. 14).

2.4 Liquidity determinants

In the literature, there exists numerous determinants of the stock market liquidity. First proposed factor is the trading volume. There is a theoretical explanation that trading activity is positively related to liquidity because the increased activity will allow the market participant to reduce its inventory risk. (Hochmeister 2007, pp. 51) Another proposed explanation is that the trading amount is positive related to the liquidity because investors have a tendency to concentrate their trading during same hours which will allow them to benefit from the increased liquidity supply. However, due adverse selection problem, the rising trading volumes generate disequilibrium in the market which leads to increased trading costs that have been offset by enlargement of the spread, decreasing liquidity. (Ajina et al. 2015) Empirical evidence remains ambiguous.

Price volatility has also been proposed as a determinant for liquidity as it measures the information content, information arrival and information asymmetry in the market. A change in the market prices that is followed by change in investors' expectations will lead to an increased variance of returns. Price volatility affects especially in inventory holding costs and risks, that are associated with widening bid-ask spread. (Ajina et al. 2015) It is

also found out that increased volatility will increase the portion of limit orders compared to market orders in the order flow, but at the same time it decreases the aggressiveness of the limit order prices (Hochmeister 2007, pp. 51). The stock liquidity is also dependent on the firm's position compared to competitors. Kale & Loon (2010) show that stock prices of a firm with strong market position are less sensitive to product order flow, which will result in greater stock liquidity. In, general, the stock liquidity increases with market power because it reduces the price volatility (Kale & Loon 2010).

Share price is commonly used to explain stock liquidity and it has been both, positively and negatively, associated with liquidity. It has been found out that liquidity is impaired when stock price declines (Rösch & Kaserer 2013). Also, it has been shown that higher share price induces a lower relative spread, meaning negative relation to liquidity (Hochmeister 2007, pp. 51).

Other mentioned affecting factors have been firm size, listing country and international financial reporting standards (IFRS). It has been proposed that stocks of smaller companies with weaker capitalization are less liquid because they are more sensitive towards high level of information asymmetry. Also, stock being publicly listed in the U.S. markets has positive correlation to stock liquidity due the standards and regulations increasing confidence and attracting investors. Similarly adopting IFRS has enabled investors to better understand economic reality due more informative reporting. (Ajina et al. 2015)

2.5 Predicting market liquidity

Regardless of the investor type, predicting stock market liquidity is necessary for all the market participants to perform efficiently. The motivation driving the liquidity prediction is simply reducing financial risks and increasing expected returns. The results of liquidity prediction have been source of creating new and improving existing trading strategies.

Banks and other financial institutions have commitments to their shareholders to maximize profits, which leads to a development in the increased volume of investments. At the same time, the banks have liabilities to depositors to refund their deposits, which makes it necessary to retain sufficient liquidity, especially because of the depositors' rather stochastic behavior. The rising clash of interests calls for a balance between profitability of longer-term investments and risks due the shorter-term liabilities towards depositors. The liquidity management is critical for the bank as too much liquidity causes ineffective allocation of capital while too low liquidity can result into a loss of market and credit. (Tavana et al. 2018) To perform liquidity risk management efficiently, it is necessary to be able to predict the market liquidity.

Stocks are generally considered to be rather liquid assets, however a need for liquidity prediction still exists. While smaller equity trades are often executed with price levels close to the mid-price, larger trades often face far inferior prices. (Breen et al. 2002) In portfolio management, better estimations of trading costs improve the ability to manage portfolio successfully. Thus, successful liquidity prediction can be used in creating and improving optimal order execution strategies. With the results of liquidity prediction, the trades can be timed efficiently to reduce the transaction costs and to execute the trades at favorable time. This is especially useful for traders, who are willing to close their positions overnight to avoid risks. (Härdle et al. 2012)

Since the emergency of high-frequency trading, prediction of the relevant metrics in high-frequency financial markets and monitoring the dynamics of LOB has become an efficient way to gain information edge. (Ntakaris et al. 2018) To reduce the numbers of losing positions, traders in high-frequency markets implement a wide range of prediction models that predict short-term direction of the markets as well as shortages of liquidity. These methods allow traders to define the quantities and levels of aggressiveness of their orders based on expectations of surplus or paucity of the market liquidity. (Aldridge 2013, pp. 17)

2.6 Previous research

Previous research offers evidence of predictability of stock market liquidity. Chan et al. (2002) describe positive autocorrelation among trading volumes in the stock market, which can be interpreted as the liquidity is not random, but dependent on the previous states. Similarly, von Wyss (2004) finds high autocorrelation with vector autoregressive model and the results are interpreted as liquidity measures have tendency for mean-reversion. The finding gives evidence that the mean of previous values could be used to predict liquidity. Breen et al. (2002) use price impact as a liquidity measure in their cross-sectional regression model for predicting stock market liquidity with promising results. Härdle et al. (2012) forecast intraday LOB volumes using dynamic semiparametric factor model and they show that the recent liquidity demand had the strongest impact to the current state of the LOB. This thesis contributes to the existing research by generating new models to predict stock market liquidity with the high-frequency LOB.

3. METHODS USED

3.1 Time series prediction

A time series is a discrete sequence of a time-valued data samples over time (Konar & Bhattacharya 2017, pp. 2). Samples in time series are not independent and they need to be addressed in their time dependent context. There exist a lot of instances such as air temperature, person's heart rate and daily closing value of Dow Jones Industrial Average, where the data is measured with regular intervals of time. The unprocessed data within given a finite interval of time describes a time series. (Konar & Bhattacharya 2017, pp. 2). A fundamental feature of the time series is that the adjacent observations are dependent, which makes it possible to predict future values of a time series from currently and previously observed values. (George et al. 2016, pp. 2)

The motivation of the time series data collection is the time series analysis and the time series prediction. Prediction of time series is practical for its extensively implemented applications, especially in the financial sector. Various financial institutes have their investments in equities, forex, derivatives, commodities, and other financial instruments. For these kind of trading activities, a broad speculation is necessary. The speculation is possible with the time series prediction of the financial instruments. Wrong prediction of time series causes losses for institutions as well as correct predictions results yields profits. (Konar & Bhattacharya 2017, pp. 2). Predicting financial markets is also a critical component of financial risk management.

In time series prediction, some hindrances exist. As time series prediction problems include the time component, and while time component gives more information about the problem, it also makes the problem more complex compared to other prediction problems (Konar & Bhattacharya 2017, pp. 2). Thus, there does not exist any straightforward technique to determine the exact nature of the time series, which makes it a challenging prediction task.

In the literature, there exists many different techniques to perform the time series prediction, such as regression, probabilistic techniques, heuristics, neural network based supervised learning and many others (Konar & Bhattacharya 2017, pp. 2). Since the complexity of financial time series, multivariate dynamic models are required to make predictions over time. Lately machine learning has received recognition as being the best practice of doing time series prediction (Schmidhuber 2014).

In addition to performing the time series predictions, it is also necessary to evaluate their accuracy. This way the risks included in the predictions can be obtained, which then again can be included in the decision making when considering actions based on the predictions. (George et al. 2016, pp. 2) The value of a time series prediction is based on the performance of the prediction capabilities for unseen data, which can be described as a prediction power.

3.2 Supervised machine learning and regression

Supervised learning is a concept of utilizing a known dataset to make predictions. The supervised learning is distinguished from unsupervised learning by the requirement of training instances which are the input and output combinations provided by the known dataset. The main purpose of the supervised learning techniques is to learn how to predict a variable Y based on a set of variables X, where Y and X depended on the problem that is been solved. The goal is to find the best predictor $f: X \rightarrow Y$, $X \rightarrow f(X)$ among the set of all functions $F = \{f: X \rightarrow Y\}$. (Grischi et al. 2012) In machine learning, the predictor f is a neural network and X is a task related dataset. The performance of the predictor is evaluated by a loss function L that is dependent on f, X and Y. The predictor f is better than the predictor g if $L(f, X, Y) < L(g, X, Y)$. The loss function is used to estimate the predictor f depending the nature of the Y.

The supervised machine learning techniques are used to perform regression and classification. In classification, the goal is to identify which category an object belongs to. An example of classification application is an e-mail spam detector that will label the coming emails as spam or not spam. In regression, the goal is to predict an attribute associated with an object. An example of regression application is a prediction of a house price based on details of the house. The main difference between regression and classification is that the output variable Y in regression is continuous while in classification it is discrete. In this thesis, the regression problem is to predict the LOB depth, where the output will be a real floating point value.

Despite the great success in financial time series predictions, machine learning methods are mainly developed through empirical testing (Ntakaris et al. 2019). Feature extraction is often performed among other machine learning techniques because financial time series data is stochastic (Zhang et al. 2019). Instead of using the raw dataset as X, unique features are identified from the data which will then be used as X. The benefit of the features is that they reveal hidden information from the data that is not available in the

raw data. The feature extraction is done in order to create machine learning models that perform better compared to models trained on raw data. (Ntakaris et al. 2019)

3.3 Artificial neural networks

Artificial neural networks (ANNs) are widely utilized in modelling complex real-world problems and they have a crucial role in learning the dynamic behavior of a financial time series (Konar & Bhattacharya 2017, pp. 6). ANNs is a subset of artificial intelligence that provides the ability to automatically learn, similarly as biological neural networks. They are partly inspired biological neurons and have partly similar architecture as human brain (Livingstone 2008). ANN is defined as structures comprised of densely interconnected adaptive simple processing elements that can perform massively parallel computations for data processing and knowledge representation (Basheer & Hajmeer 2000; Übeyli 2005, pp. 8). The key characteristic of an ANN is its ability to learn (Livingstone 2008). Other benefits of ANNs arise from information processing capabilities such as high parallelism, robustness, ability to handle complex data, failure tolerance, ability to handle imprecise information and ability to generalize (Basheer & Hajmeer 2000). An ANN consists of three components: node character, network topology and learning rules.

3.3.1 Node

A node is the basic simple processing unit of an ANN. A node receives multiple inputs from the other nodes of the ANN, that are connected with different weights, and calculates their weighted sum. If the weighted sum of node inputs is large enough, the node activates and passes the signal through a transfer function and transmits it to the following nodes. A simple network with a few nodes is represented in Figure 3.1.

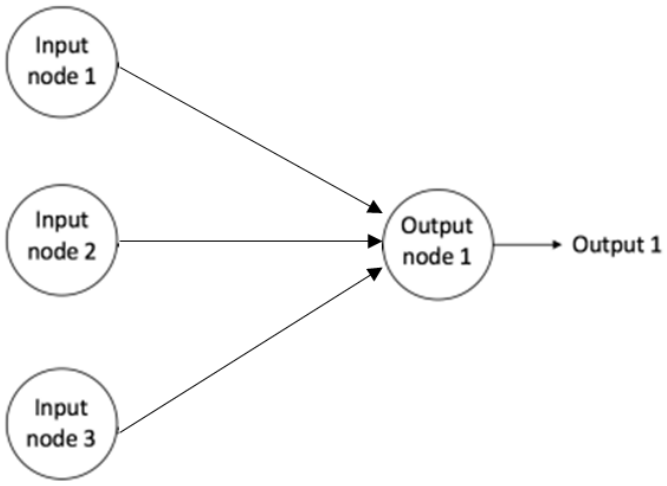

Figure 3.1. *Nodes of the ANN. First nodes are the input nodes and last node is an output node.*

The node receiving and sending signal is modelled with equation

$$y = f\left(\sum_{i=0}^{n} w_i x_i - T\right),$$

where y is the output, f is the activation function, w is the weight, x is the input and T is the threshold. (Livingstone 2008) An activation function is a mathematical equation that determines the output. The activation function is attached to each node in the network. In its simplest form, it is binary, and it determines whether the node should be activated or not, for example value being 0 is interpreted as the node will not be activated and value being 1 is interpreted as the node will be activated. (Keller et al. 2016, pp. 27) The most commonly used activation function in the construction of ANNs is the sigmoid function (Keller et al. 2016, pp. 28):

$$f(x) = \frac{1}{1 + e^{-x}}.$$

In the sigmoid function, the return value is between 0 and 1. Other widely used activation function is the hyperbolic tangent function (Li et al. 2017):

$$\tanh(\mathrm{x}) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

The hyperbolic tangent function can return negative values as well and the return value is between -1 and 1. The hyperbolic tangent function is plotted in Figure 3.2.

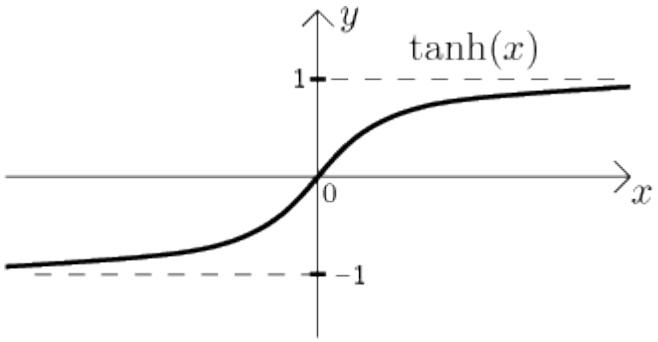

Figure 3.2. *Hyperbolic tangent function.*

The selection of the activation function should be made based on the problem, training data and other parameters. Other functions can be used as activation functions as well which all serve a different purpose. Examples of other activation functions are rectified linear unit function and identity function. (Keller et al. 2016, pp. 27)

3.3.2 Architecture

The architecture of an ANN consists of multiple layers and it can be designed in multiple ways. The nodes of the ANN are organized into layers that are linear arrays. The first

layer of an ANN is an input layer, the last layer is an output layer and between them there can be one or more hidden layers. In general, designing the architecture includes determining the number of layers in the ANN, the number of the nodes in each layer and the connections of the nodes. (Livingstone 2008) There does not exist any general model architecture that works well for all problems. Instead, the architecture needs to be tailored for the specific problem and it usually requires multiple iterations to find the most suitable architecture for the task being solved. General architecture of an ANN is represented in Figure 3.3.

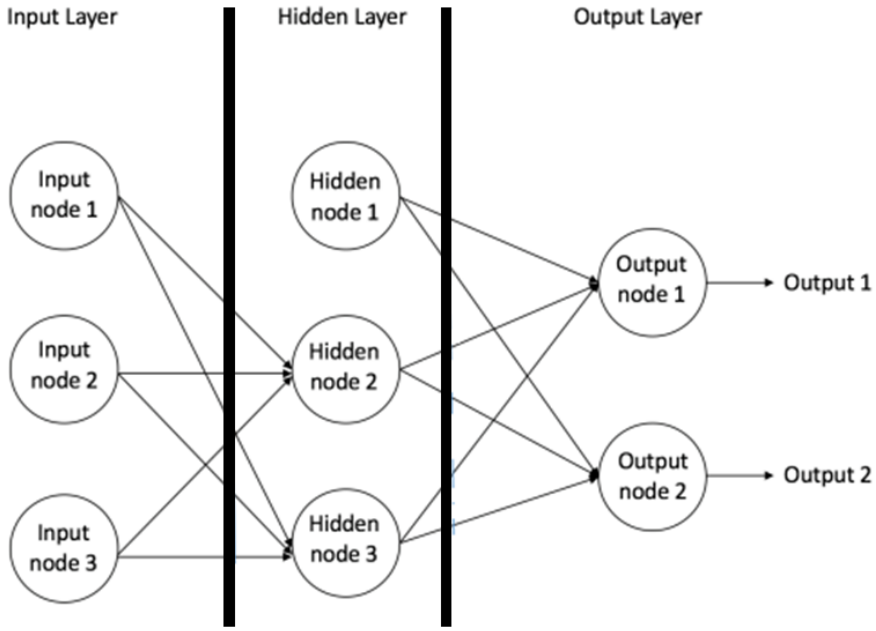

Figure 3.3. *ANN architecture.*

The connections can be one-way connections or loop-back connections. The loop-back connection allows the output node to be the input node of the previous or the same level nodes. Based on this observation, ANNs are classified into feedforward neural networks and feedback networks usually known as recurrent neural networks. (Livingstone 2008)

3.3.3 Learning

The ANNs are trained using learning process during which the weights are adjusted to desired values with optimization methods. The learning can be supervised or unsupervised. In supervised learning the training is done by using a training set and adjusting the weights to minimize the error between the network output and the correct output. On the contrary, unsupervised learning is not using target output values from the training set and the network tries to capture the underlying patterns only from the input. (Basheer & Hajmeer 2000; Livingstone 2008)

The training of an ANN is done using an existing dataset, but the real benefits of using the ANN is only realized when it can be applied to new data samples. ANNs are known for performing well in generalization. However, in order to build the capability to generalize, fitting of the ANN to the dataset needs to be done carefully since underfitting or overfitting will cause poor performance. Overfitting of the training data leads to declining generalization capabilities of the model and untrustworthy performance when applied to new unforeseen data. Overfitting happens due the ANN learning the training dataset too well, capturing the details and noise to the extent that does not represent the actual patterns anymore. It will then affect negatively to the performance of the ANN with the new data. The poor performance is explained by that the noise and the random fluctuations in the training data are captured and learned as concepts by the model even though they do not apply to the new data samples. (Piotrowski et al. 2013) On the contrary, underfitting refers to the learning process where an ANN does not learn the patterns from the training data enough meaning that the ANN will perform poorly even with the training dataset. Thus, the underfitted ANN will certainly perform poorly with new data samples as well. Examples of underfit, good fit and overfit are represented in Figure 3.4.

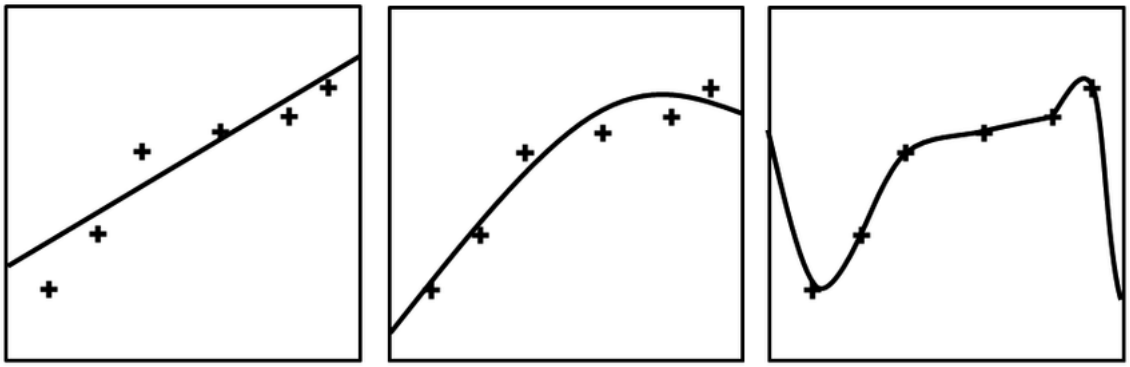

Figure 3.4 *Examples of underfit, good fit and overfit (Rahul et al. 2014).*

On the leftmost side, the line captures only the basic trend of the data points having a large loss. In the middle, the curve captures the pattern better and the generated loss is significantly lower than the underfitted line. On the rightmost side, the curve is perfectly fitted over the data points minimizing the loss while losing the ability to generalize the trend. (Rahul et al. 2014)

An ANN is trained by adjusting the weights v and w, which is done by solving the optimal weights using optimization algorithm. Stochastic gradient descent is considered as the standard optimization algorithm in machine learning and it has been proven to be relatively efficient optimization method (Kingma et al. 2015). Gradient descent algorithm iteratively follows the negative gradient to move in the direction of the descent and eventually locate the desired minimum. First, the gradient ∇L of the loss function with respect

to each weight w_{ij} of the network needs to be calculated. The gradient tells how much a change in that weight will affect the overall loss. The gradient ∇L is calculated

$$\frac{\partial L}{\partial w_{ij}} = -(y_j - x_j)x_i.$$

The gradient descent is then performed by subtracting a small portion μ, called learning rate, of ∇L from the weights. The new weight will be

$$w_{ij} = w_{ij} - \mu \nabla w_{ij}.$$

The gradient descent algorithm will perform gradient descent iteratively until the algorithm converges to $\nabla L = 0$. Important considerations are the learning rate and the initialization of the weights. The learning rate in the gradient descent determines how much the weights are adjusted at each iteration. If the learning rate is too small, the gradient descent algorithm will take very long time until converging. On the contrary, if the learning rate is too large, the algorithm does not converge at all. (Chow et al. 2007, pp. 34) The effect of the learning rate in the scenarios of small and large learning rates are demonstrated in Figure 3.5.

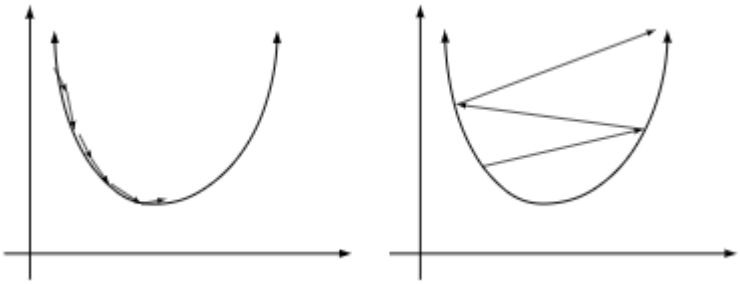

Figure 3.5. *Small (left) and large (right) learning rates (Chow et al. 2007, pp. 34).*

The gradient descent algorithm with a small learning rate slowly converges to the minimum. With too large learning rate, the algorithm diverges. To tackle the issue, instead of choosing a fixed learning rate, the learning rate can be changed during the training process to optimize the learning. The way the learning rate changes over time is called learning rate decay. The simplest way to perform learning rate decay is to decrease the learning rate linearly from a large initial value to a small value. (Goodfellow et al. 2016, pp. 294). The second consideration of the gradient descent algorithm is the initialization of the weights. The initialization of the weights determines the starting point of the algorithm. With some initializations, the algorithm converges to a local minimum instead of the desired global minimum. The gradient descent algorithm will stop when it converges to a minimum. Examples of different minimums are visualized in Figure 3.6.

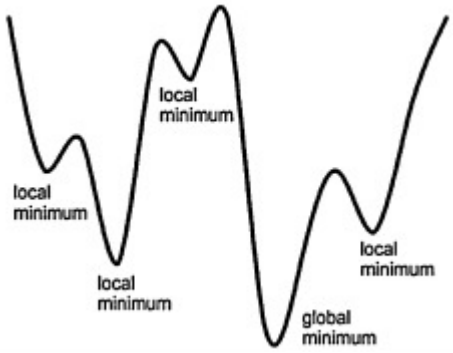

Figure 3.6. *Local minimums and a global minimum (Andreeva & Chaban 2015).*

Only some initializations that are close enough to the global minimum will converge there. One of the simplest initialization methods is the random weight initialization which gives generally quite good results, but as seen in the figure, the weights may converge to different minimums which may not be desired (Thimm & Fiesler 1994).

A stochastic gradient-based optimizer referred as “Adam” proposed by Kingma, Diederik and Ba (2015) is a replacement algorithm for stochastic gradient descent. Adam is described as an algorithm for first-order gradient based optimization of stochastic objective functions, based on adaptive estimates of lower order moments. Thus, it is a variant of stochastic gradient descent that only requires first-order gradients. Adam is suitable for problems with large datasets, since it is computationally efficient and does not require much processing memory. It is also suitable for problems with very noisy and sparse gradients. (Kingma et al. 2015)

As a part of the network optimization, a dropout regularization is used to avoid the overfitting the training data. A dropout is regularization method for fully connected neural network layers. The dropout is performed by learning network output weights that provide a compromise between the original hidden layer outputs and the hidden layer outputs obtained by applying dropout with a probability value p that is also called the dropout rate. (Iosifidis et al. 2014) This means that the dropout works by probabilistically dropping out inputs of a hidden layer making nodes in the network generally more robust to the inputs. The dropout mechanism is illustrated in Figure 3.7. The crossed nodes represent the nodes that are ignored during the training.

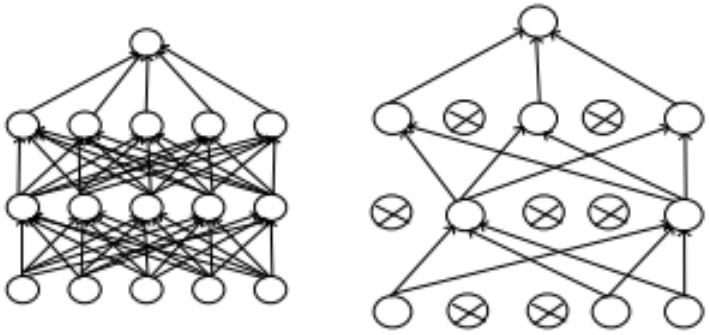

Figure 3.7. *Neural network connections without dropout mechanism (left) and with dropout mechanism (right) (Yang & Yang 2018).*

The random selection of the dropout mechanism ignores some of the hidden nodes in the training process. Due the randomness, each training network might be different. Ignoring some of the hidden nodes weakens the connections between the nodes which makes the network less prone to overfitting. In general, the dropout reduces the cost of calculation and helps in discovering the most essential characteristics of the data. (Yang & Yang 2018)

To further avoid overfitting, early stopping mechanism is used to stop the training before the overfitting happens. Training of an ANN with tens of epochs usually takes a lot of time and after some number of epochs, the training will not improve the performance anymore and instead the loss will start to increase. The early stopping mechanism will monitor the validation loss after each epoch and after the loss will start to increase, the training is stopped. However, often the first sign of the loss increasing is not the optimal time to stop the training, because the performance of the ANN may get slightly worse before improving due the noise in the training data. This is handled by a patience parameter, which is added to the early stopping mechanism. Patience parameter is a counter which counts the epochs in which the loss has increased. After the loss has increased the determined number of times, the early stopping will be triggered. (Rawat et al. 2020) Using the patience parameter will thus delay the early stopping over the noise until the loss increases multiple times and it becomes unlikely that the loss would decrease by continuing the training.

3.4 Feedforward neural networks

A feedforward neural network (FNN) is an ANN where the connections between the nodes do not form a cycle. In this type of network, the information moves only forward from input nodes to possible hidden nodes and finally to the output nodes. There does not exist any feedback connections in which outputs of the networks are fed back into

itself. FNNs are mostly used in the fields of computer vision and text recognition (Kumar et al. 2013; Zhang et al. 2014).

3.4.1 Single-layer perceptron

The simplest instance of a FNN is a single-layer perceptron (SLP). The perceptron has a similar structure as a biological neuron. The single-layer perceptron network has a single layer of output nodes and the inputs are fed directly to the outputs with series of weights. The input sum is the input of the activation function and if the threshold value is exceeded, it activates. (Graupe 2013) The architecture of a simple SLP is represented in Figure 3.8.

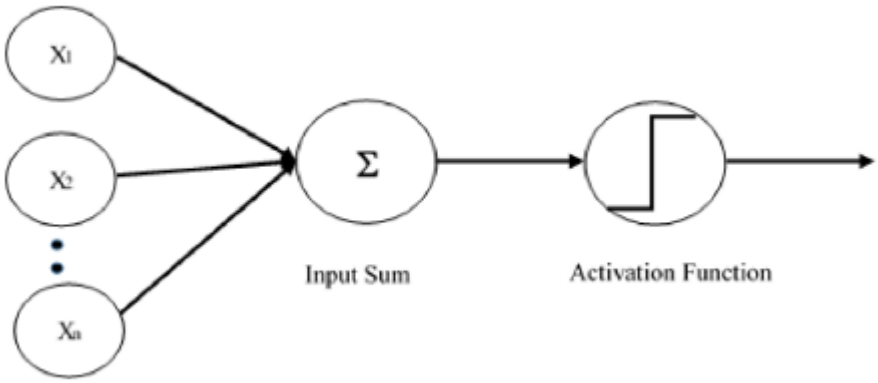

Figure 3.8. *A simple SLP architecture (Zaccone 2016, pp. 98).*

Researchers have used SLPs to various two-class problems. However, SLP is only capable of learning linearly separable patterns. And for non-linear regression, a multi-layer perceptron is required. (Hu 2010)

3.4.2 Multi-layer perceptron

Multi-layer perceptron (MLP) is a FNN where the neurons are interconnected in multiple layers. MLPs are one of the most popular and versatile ANNs and they are useful for regression prediction tasks where a real value is predicted given a set of inputs. (Castellani 2017) Usually MLPs are used with image data, text data and time series data (Rana et al. 2018).

MLP includes one input layer, one output layer and at least one hidden layer between them. Each layer consists of multiple nodes. (Liu et al. 2018) Given a set of features $X = x_1, x_2, \dots, x_n$ and a target y, the MLP is capable of learning a complex function approximator for either classification or regression task. In classification, the output y is the class with the highest probability and in regression the output y is the predicted value. MLP is also suitable for multi-output tasks, where the sample has more than one output value.

(Rana et al. 2018) The architecture of a simple MLP with single output is represented in Figure 3.9.

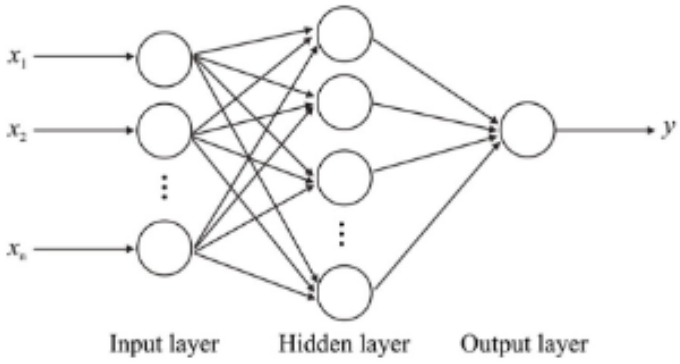

Figure 3.9. *MLP architecture with a single hidden layer (Fath et al. 2020).*

The first layer on the left side is called the input layer and it consists of a set of neurons that represent the input features. In the middle, there are one or more hidden layers, one in this simple architecture. The last layer is the output layer that receives the information from the last hidden layer and transforms it into final output value. The nodes in consecutive layers are connected with weighted connections w and v, which will be modified during the training of the MLP. Each node performs a weighted sum of its inputs followed by an activation function. The MLP is defined as

$$y^p = v_0 + \sum_{j=1}^{J} v_j f(w_{j0} + \sum_{i=1}^{I} w_{ij} x_i),$$

where y^p is the estimated value of the dependent variable y, J is the number of hidden nodes, I is the number of input nodes and f is the activation function. The weights w and v include thresholds w_0 and v_0 and weights connecting nodes in the consecutive layers w_{ij} and v_{ij}. (Haykin 1999, pp. 32)

The advantages of MLP is its capability to learn non-linear patterns from the input data and to learn the patterns in real time with partial fitting. The disadvantages of MLP is that it needs tuning of several parameters such as the number of hidden neurons, the number of the hidden layers and the number of the iterations. Usually the parameter selection is done by testing different values and choosing the most suitable one (Parra et al. 2014). The MLP has also two critical drawbacks. One is that MLP has a non-convex loss function where exists multiple local minimums, which means that different initialization will lead to different validation accuracy. The other one is the slowness in learning speed. MLP is also sensitive to feature scaling, thus data normalization is required. (Byung-Joo 2012)

3.4.3 Convolution neural network

Convolution neural networks (CNN) are variations of FNNs that utilize convolutional layers. A CNN has a multi-layer neural network structure that simulates the operation mechanism of a biological vision system. (Winkler 2016; Cao & Wang 2019). CNNs are most commonly applied in image recognition, but they have also found applications in natural language processing and financial time series prediction (Zhang et al. 2019). Compared to regular FNNs with similar number of layers, CNNs have much fewer connections and parameters due to the local-connectivity and shared-filter architecture in convolutional layers, which makes them less prone to overfitting. Other advantages of the CNNs are ease of the training and the pooling operation that improves the generalization capability. (Wu & Gu 2015)

CNNs consist of consecutive convolutional layers and pooling layers usually followed by fully connected layers. The convolutional layers and the pooling layers are used to capture the fine temporal dynamics of the time series. After the convolution and pooling is completed, the fully connected layers are used to perform the regression to return the final output value. (Tsantekidis et al. 2018) CNNs can be separated by the dimension of the convolution they are performing. In time series analysis, one-dimensional (1-D) convolutions are used since the input data is one-dimensional. A simple 1-D CNN architecture with one convolutional layer is presented in Figure 3.10.

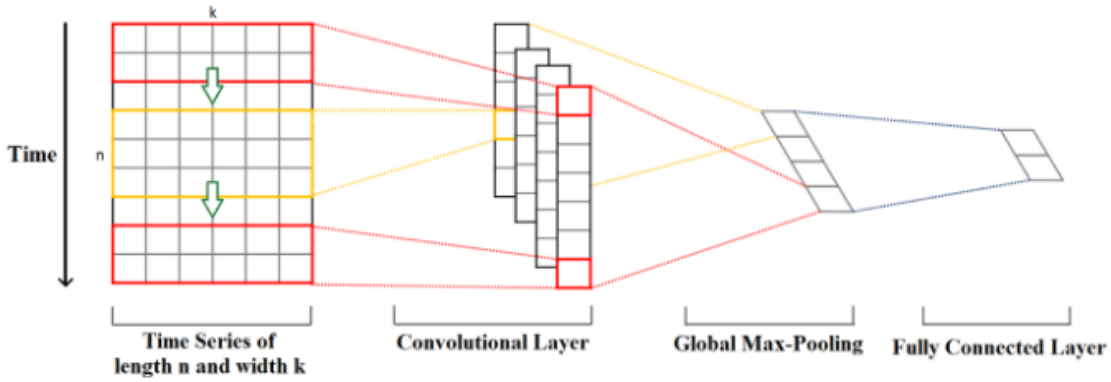

Figure 3-10. *A simple 1-D CNN architecture (Yoon 2014).*

Convolutional layers with convolving kernels are applied to the input data to extract features that represent the dynamics of the input data. For time series data, 1-D convolution is applied, and the convolution kernels have the same width as the time series data. The 1-D convolution means that the time dimension is used to calculate the convolution and the shape of the output is an 1-D array. The 1-D CNN takes an input dataset as an input that has length n and width k, where the length is the number of timesteps, and the width is the number of variables. The kernel performs convolution from the beginning of the

time series to the end. The elements of the kernel are multiplied with the input so that the output is enhanced in a desirable way. Subsequently, the results are added together, and a non-linear activation function is applied generating a filtered time series vector. The number of the filtered time series vectors will be the same as the number of used convolution kernels. (Yoon 2014)

Next, global max pooling is applied to each of the filtered time series vectors in the convolutional layer. Max pooling is a discretization process, where the purpose is to perform down sampling for the input data by reducing its dimensionality. For example, max pooling can transform an 8 x 8 matrix into a 6 x 6 matrix. As a result, the max pooling will reduce the spatial information of the input data. (Hang & Aono 2017) Ideally the pooling technique is expected to maintain the important information while discarding the irrelevant information. Successful max pooling will result in avoidance of overfitting by providing more abstract form of the representation. Also, it will reduce of the required computational time. (Wu & Gu 2015) The max pooling is simply performed by choosing the maximum value from the pool of values. The global max pooling means that the pool is the whole vector. A new vector is formed from the results of the global max pooling, which will be the final feature vector that is used as an input to the fully connected layer. (Wu & Gu 2015) The fully connected layer uses the feature vector as an input to generate the final prediction similarly as a SLP.

3.5 Recurrent neural networks

Recurrent neural networks (RNN) are ANNs where connections between nodes form directed cycles, allowing information to be stored inside the network. With the stored information, RNNs have an internal state, which is convenient for time series prediction. (Ian et al. 2017) RNNs are thus distinguished from FNNs by a feedback loop that allows previous outputs to be used as inputs while having hidden states. Unlike FNNs, RNNs can process arbitrary sequences of inputs using their internal memory state. (Potznyak et al. 2019) RNNs have found broad success in multiple applications such as natural language understanding, speech recognition, time series prediction and video processing (Sak et al. 2001; Dixon 2018). Previously RNNs haven been applied to LOB to predict the next event price-flip by Dixon (2018).

A simple RNN is established by applying a function F_h repeatedly to the input data $X = X_1, X_2, \dots, X_n$. For each time step t the function generates a hidden state h_t from the current input X_t and from the previous output h_{t-1}

$$h_t = \sigma(W_h X_t + U_h h_{t-1} + b_h),$$

where $\sigma(x)$ is a non-linear activation function, W is a weight matrix, U is a hidden weight matrix and b_h is a bias term (Dixon 2018). The information is stored to the hidden state h_t. Since the feedback loop occurs at every time step, each hidden state contains information also from the states that preceded h_{t-1}. The final output Y of the output of the final hidden state

$$Y = W_y h_t + b_y.$$

A simple RNN is a single hidden layer neural network unfolded over all timesteps. The architecture of a simple RNN with one hidden layer that is unfolded over a sequence of six timesteps is presented in Figure 3.11.

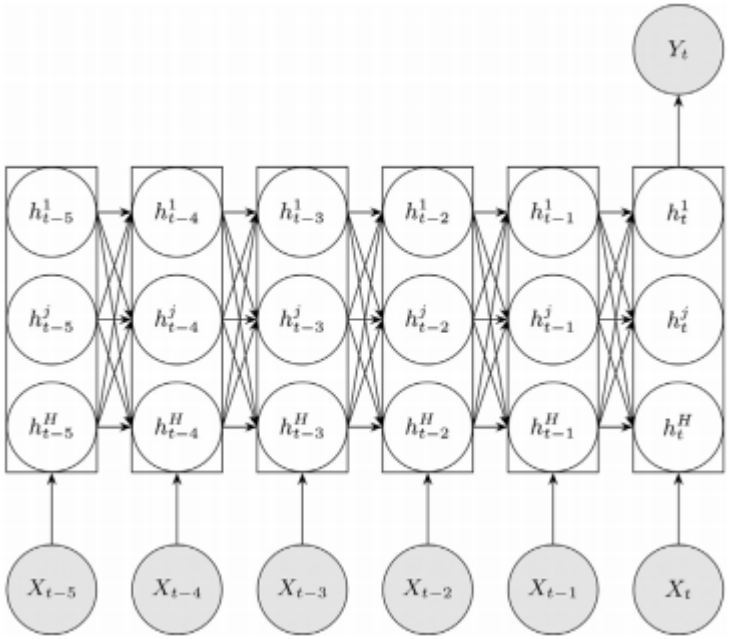

Figure 3.11. *Architecture of a simple RNN with one hidden layer that is unfolded over a sequence of six timesteps (Dixon 2018).*

The designing of the RNN architecture requires decisions about the number of the times the network in unfolded and the number of the nodes in the hidden layer (Dixon 2018).

Long Short-Term Memory (LSTM) is a specific RNN that consists of one input layer, one output layer and a series of recurrently connected hidden layers known as memory blocks. Each memory block has one or more self-recurrent memory cells and three multiplicative units that are an input gate, an output gate, and a forget gate. (Li et al. 2017) The memory cell stores the temporal state of the network and controls the information flow in the network. Due to the memory cell, LSTMs perform well at finding and exploiting long range dependencies in the data which is often crucial in time series prediction. (Graves 2014) A memory block with single memory cell is illustrated in Figure 3.12.

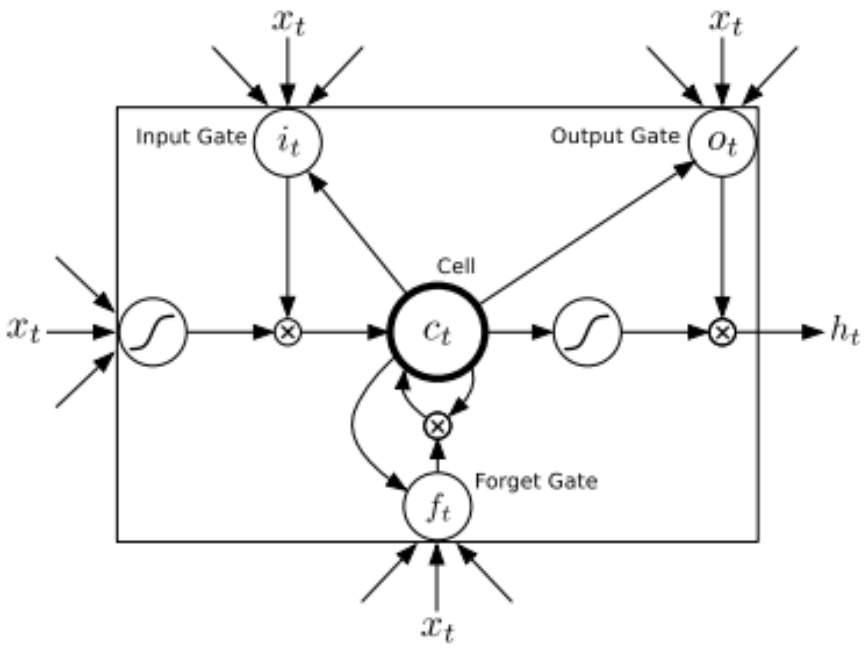

Figure 3.12. *Long short-term memory block (Graves 2014).*

The self-recurrent cell state C_t allows information from previous intervals to be stored. The memory cell is modified by the forget gate f_t below the cell state. The forget gates purpose is to discard long-term dependencies and it tells the memory cell to which information to forget. The input gate i_t controls which information will enter and be stored in the cell state. The output gate o_t determines which information will be moved to the next hidden state h_t.

Activation functions are applied in all three gates and they determine the amount of information to be passed through the gate. The activation functions are usually sigmoidal. (Soutner & Müller 2013) The output of the sigmoid function is between 0 and 1, where 1 could be interpreted as all the information will be passed through the gate and 0 could be interpreted as none of the information will be passed through the gate. The outputs of the activation functions have the following effect to the network. When the output value of the input gate is close to zero, it flattens the value from the net input, effectively blocking that value from entering to the cell state. When the output value of the forget gate is close to zero, the memory block will effectively delete the previous values stored in the cell state. When the output value of the output gate is close to zero, effectively none of the information will be passed to the next hidden state. (Soutner & Müller 2013) The full architecture of the LSTM is illustrated in Figure 3.13.

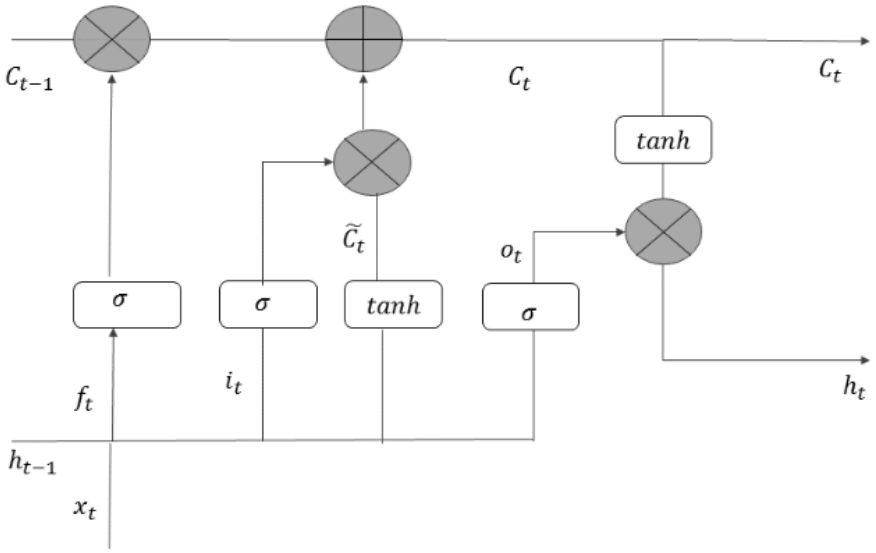

Figure 3.13. *LSTM architecture (Li et al. 2017; Pattanayak 2020).*

The first step in LSTM is to determine the information to be removed from the cell state, which is done by the forget gate f_t that uses the sigmoid activation function

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big).$$

The second step is to determine, what new information will be stored to the cell state. The input gate i_t determines the values to be updated and a new vector $\bar{C}_t$ is created with potential values to be stored into the cell state, which will then be combined to update the cell state

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\bar{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$

Next, the new cell state can be formed. The previous cell state C_{t-1} is updated to the new cell state C_t by multiplying it to the forget gate f_t and the new information to be added is received by multiplying the input gate i_t with the new value vector $\bar{C}_t$

$$C_t = f_t C_{t-1} + i_t \bar{C}_t.$$

After updating the cell state there is still a need to determine the output that is based on the cell state. The hyperbolic tangent activation function will be applied to the cell state that will be multiplied with the output gate o_t

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \tanh(C_t).$$

(Hochreiter & Schmidhuber 1997; Li et al. 2017) Since the recurrence, the same actions will be performed to all the following cells states.

4. DATA ANALYSIS

The research was conducted using NASDAQ's "TotalView-ITCH" data from 2014. The dataset contains time series LOB data of five stocks from the Nasdaq stock market in April. The dataset includes approximately 27 million samples in total extracted over time of five consecutive trading days. The dataset contains the first ten levels of the LOB.

4.1 Data understanding

The dataset includes five full days of ultra-high frequency intra-day data that was received from NASDAQ ITCH feed. The dataset consists of five stocks that are Apple (AAPL), Facebook (FB), Intel (INTC), Google (GOOG) and Microsoft (MSFT), which are high technology stocks traded in the Nasdaq stock market. The stocks have lot of trading activity ensuring continuous flow of trades and changes in the LOB quantities. The details of the stocks of the dataset are represented in Table 4.1.

Table 4.1. *Stocks of the dataset.*

ID	ISIN	Company	Samples
AAPL	US0378331005	Apple Inc.	2673470
FB	US30303M1027	Facebook Inc.	9680014
INTC	US4581401001	Intel Corp.	3353984
GOOG	US02079K1079	Alphabet Inc.	4518370
MSFT	US5949181045	Microsoft Corp.	7135202

Initially Nasdaq stands for National Association of Securities Dealers Automated Quotations. Today, Nasdaq is described as a hybrid market and it offers market participants to choose between human floor brokers and fully automated electronic market. Almost all the trades are executed purely electronic, but some large institutional investors rely on floor brokers. In Nasdaq stock market, the trading hours are from 9.30am to 4.00pm eastern time. The data does not include records outside the trading hours, pre-opening period or post-opening period. The auction period is left out to make the LOB unbiased with less exceptions, making the order book dynamics comparable. The raw data includes step, time, type and 10 levels of LOB. An example of the LOB data is represented in Table 4.2.

Table 4.2. *Example of the LOB data.*

Step	43042
Time	34200559
Type	ORDER
Ask Price Level 1	5252000
Ask Quantity Level 1	100
Bid Price Level 1	5250100
Bid Quantity Level 1	8
…	…
…	…
Ask Price Level 10	5254700
Ask Quantity Level 10	5
Bid Price Level 10	5250000
Bid Quantity Level 10	431

4.2 Descriptive analysis

The selected stocks have continuous inflow of orders and the order quantities change almost constantly during the trading hours. This constantly changing nature of the LOB quantity is expected to be suitable for regression prediction task. The first prediction task focuses on the dynamics of the level 1 quantities. The level 1 quantities change when new orders arrive with the same or better price than the previous level 1 order, or the current level 1 orders are cancelled. The level 1 quantities are usually relatively low, but sometimes spikes with very large quantities appear. In general, the LOB seems to be balanced: the level 1 bid and ask quantities are close to each other throughout the day, which means that same models can be applied for prediction of both sides of the LOB. Spikes also happen for both sides, and it is unclear whether they are more frequent on other side. The intraday curves of the level 1 bid and ask quantities are represented in Figure 4.1, where rolling averages of level 1 quantities of INTC, MSFT and FB are represented. INTC is represented by a blue curve, MSFT by a green curve and FB by a red curve. The solid lines are ask curves and dotted lines are bid curves.

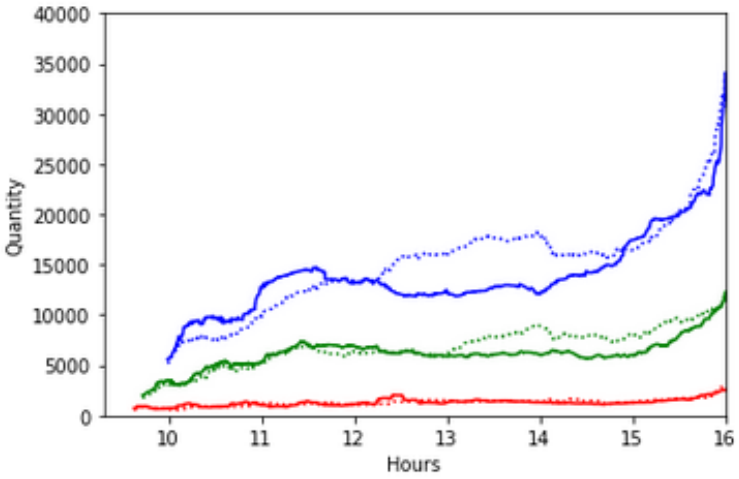

Figure 4.1. *Rolling averages of the level 1 bid and ask quantities of Intel, Microsoft, and Facebook on April 8, 2014. Blue: INTC, Green: MSFT, Red: FB. The solid lines are ask curves and dotted lines are bid curves.*

The quantities are rolling averages, so the level 1 quantity spikes are not visible in the figure, but instead the longer-term patterns. Firstly, it can be observed that different stocks are traded with different level 1 quantities. Secondly, the level 1 bid and ask quantities are quite close to each other throughout the day. During some hours, the there exists a gap between the level 1 bid and ask curves, but in general the level 1 quantities seem to be quite even. Third observation is that the level 1 quantities seem to rise significantly during the last trading hour.

The second prediction task focuses on the dynamics of the multi-level quantities. The multi-level quantity is calculated by summarizing the top five level quantities from both sides of the LOB. The multi-level quantity captures wider view of the quantities and the value is higher and changes more often than in the case of level 1 quantity. Also, the differences of the stocks become clearly observable as other stocks have much larger trading quantities. The intraday curves of the multi-level quantities are represented in Figure 4.2., where rolling averages of multi-level quantities of INTC, MSFT and FB are represented. INTC is represented by a blue curve, MSFT by a green curve and FB by a red curve.

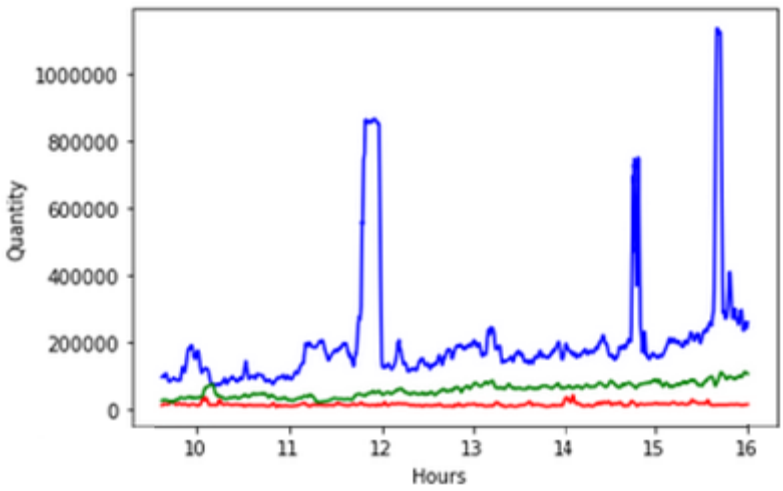

Figure 4.2. *Multi-level bid and ask quantities of Intel, Microsoft, and Facebook on April 8, 2014. Blue: INTC, Green: MSFT, Red: FB.*

The magnitude of the multi-level quantities differs from the level 1 quantities, but the order remains the same. Again, INTC has the largest quantity and FB has the smallest quantity among these three stocks. Also, similarly as in the case of level 1 quantity, the quantities seem to get larger towards the end of the trading day. The INTC has more volatile quantity compared to the other stocks, which may cause challenges to the predictions.

In addition to the intraday changes, longer-term patterns can be observed on the full 5-days period. In Figure 4.3, the level 1 quantities of AAPL are represented over 5 days. The level 1 ask quantity is represented by red curve and the level 1 bid quantity is represented by a blue curve.

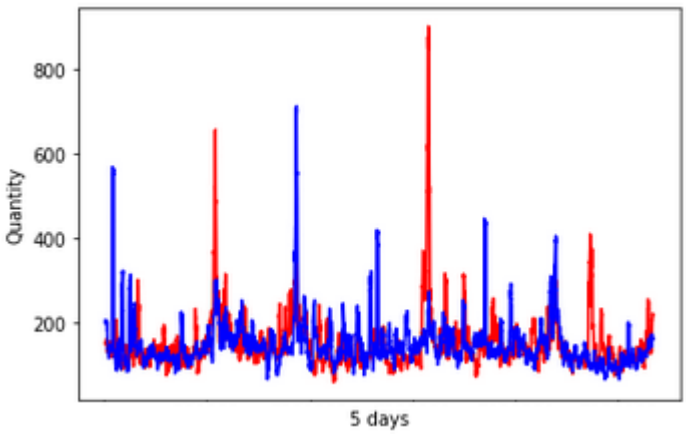

Figure 4.3. *Level 1 bid and ask quantities of AAPL during full five trading days. Red: Ask curve, Blue: Bid curve.*

Level 1 ask and bid quantities continue to remain close to each other for this five-day period, thus it not just an intraday phenomenon. Also, the level 1 quantities do not change

by much between days and the curves seem to have similar patterns each day. Most of the time, the quantities remain at small levels, usually under 200, but some larger quantities still occur. Some spikes may be happening, but they are short-lived, and block trades do not occur, where the level 1 quantity would remain huge for a longer time-period.

4.3 Data pre-processing

The data pre-processing is done to transform the input data for the use of machine learning algorithms. It will increase the efficiency of the model and the processing will be faster with less biases. The data pre-processing is done by firstly deleting the attributes that are not used in the modelling. Step, time, and type are metadata that is not required in the modelling and they are deleted from the dataset. Next, the dataset is normalized using z-score normalization to change the numeric values to a common scale, without distorting differences in the ranges of values as recommended in Ntakaris et al. (2018). Without normalization the features with big values would be outweighed in the model lowering the performance. The normalization is done using the z-score normalization

$$z_i = \frac{x_i - \bar{x}}{s},$$

where x_i is the input data and $\bar{x}$ is the mean and s is the standard deviation. After the data modelling, the z-score values are inverted back to the original unit to report the findings in the same unit as the original data. Normalized LOB data is used as input data X of the models.

4.4 Model validation

Model validation is a method of measuring the predictive performance of the model and it is used to avoid overfitting the model. In time series prediction, it is relevant to evaluate the model performance with out of the sample data. To evaluate the model with different data, it is required to split the dataset to the data to be used in the training and to hold back some of the data to make predictions with. However, the mostly used validation methods in machine learning such as k-fold cross-validation and train-test splits are not suitable for time series problem because they assume the observations are independent, thus ignoring the temporal time component, which is fundamental in time series prediction. Instead of the common validation methods, the data split must be done with a respect to the order of the samples in the dataset. (Brownlee 2017)

A day-based prediction framework is developed following a three-fold split, where the sequence of the samples is kept the same. The dataset includes 5 full trading days, and the split is done by using first three days as a training set, fourth day as a testing set and the fifth day as a validation set. The split mimics a common 60-20-20 train-test-validation split, but the actual number of samples vary due to different number of samples during a trading day. The number of samples of each stock in each set are represented in Table 4.3.

Table 4.3. *Dataset split and the number of samples in each set.*

Set	Days	AAPL	FB	GOOG	MSFT	INTC
Train	1-3	1575420	5373365	2983051	4401184	1936311
Test	4	473920	1986294	872955	1527272	765289
Validation	5	618920	2320355	662364	1206746	652384

5. RESULTS

5.1 Evaluation metrics

To quantify the model performance, model evaluation metrics are used. The effectiveness of the machine learning algorithms can be evaluated using suitable performance measurements called evaluation metrics. The metrics estimate the performance of the model for unseen data, how well the model can predict the future. For regression problems, most used metrics mean squared error (MSE) and mean absolute error (MAE) (Handelman et al. 2019).

With regression problems, the relationships between variables is demonstrated by an equation that calculates the distance between the fitted curve and the actual data point. A measure of a degree that the regression curve fits the data and makes predictions reliably is represented by MSE. It represents the variance between the predicted values and realized values. It is applied to observe how much the predicted variables differ from the actual variables. (Handelman et al. 2019) MSE is the mean deviation from the actual values

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2,$$

where Y is the actual value and $\hat{Y}$ is the predicted value. Since the error is squared, MSE treats deviations to either directions in the same way. As MSE squares the differences, it penalizes even a small error which may lead to over-estimation of how bad the model is. In this thesis, MSE is the preferred evaluation metric, because it is differentiable and hence can be optimized better. MSE is a good metric when large errors are undesirable, since it is sensitive towards outliers.

MSE values are always positive and the smaller the score is, the better the model is performing. As the error is squared, MSE shows clearly when the model is not performing well as the MSE score gets very large when the predictions are inaccurate. MSE as an evaluation metric does not tell whether the predicted values are too large or too small. To have a better understanding of the directions of the errors, the predictions need to be plotted against the actual values. After measuring the absolute performance of the model, the scores can be compared to a benchmark model.

5.2 Models

Models were built with Python 3.6.1 using Keras, the most used application programming interface (API) for building neural networks in Python. Keras is a free open source Python library for developing machine learning models. Keras is built on top of TensorFlow 2.0 and it allows user to define and train neural networks. Keras supports sequential neural networks, RNNs and CNNs. Next, the process of developing a neural network model with Keras is explained and then the details of the developed models are described.

The development begins with creating a sequential model with Keras and then the layers are added to the model one by one. The input layer is ensured to have the correct input dimension according to batch size, timesteps and number of features. The number of hidden layers and their types are determined by the process of trial and error experimentation to find out a network that is large enough to capture the dynamics of the prediction task. For fully connected layers, the number of the nodes is defined by the desired output. Activation functions are specified based on the nature of the problem and the type of the layer. For single-output regression problems, the fully connected layer needs to have a linear activation function. For other layers, multiple choices are available, and the function is chosen by trial and experimentation.

After defining the network architecture, the network is compiled and fitted with Keras using TensorFlow backend. Before compiling the model, some additional properties need to be defined. A loss function is defined to evaluate a set of weights and an optimizer is defined to search the desired weights of the network. Also, metrics argument is defined to evaluate the accuracy of the model. After defining the model and compiling it, the model is trained by calling fit function, which is a built-in function provided by Keras. The training process runs multiple of iterations, called epochs, that is given as a parameter to the model. Another approach is to monitor the training process and apply early stopping when the training results stop improving their accuracy. The last defined parameter for training process is batch size. The batch size means the number of training samples considered before the model updates the weights. Finally, after training the network, it can be used for making predictions on new unforeseen data.

Three models were first developed: MLP, 1-D CNN and LSTM. After early experiments, the LSTM model presented the most promising results, and the model was then further developed. The developed LSTM model is a stateful LSTM, where the states of the nodes are saved for further training session instead of resetting the states. The stateful approach was chosen over stateless since stateful LSTM can learn dependencies over the sequences, which was expected to be useful in time series prediction. The developed

LSTM model consists of one input layer, one hidden layer and one output layer. The architecture of developed LSTM model is represented in Figure 5.1.

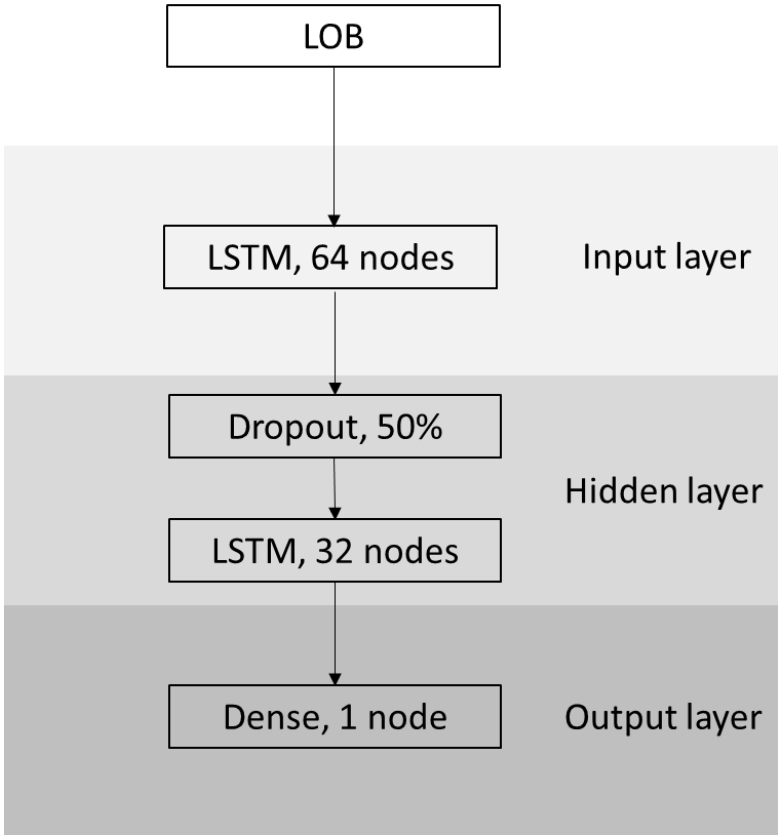

Figure 5.1. *The model architecture of the developed LSTM.*

The LSTM model is constructed as follows. The input data was three dimensional including dimensions of batch size of 10, 75 timesteps and the LOB data. The input layer is a LSTM layer with 64 nodes, hyperbolic tangent activation function and without initializations. After the first LSTM layer, dropout is applied with dropout rate of 50% and the output is then passed to the second LSTM layer. The second LSTM layer has 32 nodes, hyperbolic tangent activation function and does not have any initializations. Bias term was added to the LSTM layers by default. The output layer is a fully connected dense layer with one node and linear activation function.

Initial parameter choices were done by analyzing the structure of the problem and the nature of LSTM, but eventually multiple iterations of manually adjustment of the parameters were required to develop the model. Next, the chosen parameters are further explained. The batch size parameter was defined to be 10 since a stateful LSTM requires a small batch size. The timesteps parameter was defined to be 75 because the LSTM input gate will control the amount of timesteps that will be added to the cell state, which means that the number of timesteps could be larger as well. It was observed that the model was performing better as the number of timesteps were increased. However, the

required computational cost, random access memory (RAM) and the training time, increased significantly while increasing the timesteps parameter, which is why there was not tested larger number than 75. In the LSTM layers, kernel initialization or bias initialization were not needed, since the LTSM gates will prevent the vanishing gradient problem. The LSTM cells already include internal hyperbolic tangent and sigmoid activations functions to capture the non-linearities, so the activation function between the layers was also defined to be the hyperbolic tangent function. Sigmoid activation function was also tried, but the hyperbolic tangent function was observed to give better results. For the output layer, there had to be only one node since the required output was one value. Also, the activation function of the output layer had to be linear for the output value to be desired numeric value.

The compiling and training had also few different options. MSE was used as a loss function since it is sensitive to outliers. Adam was used as an optimizer because of its efficiency. Nesterov was tried as an optimizer, but it was slower, and it affected the performance negatively. For training, early stopping was applied with patience of 3 epochs. It was observed that once the training results start getting worse, the results would keep getting worse, so the small patience was observed to be enough.

After making the predictions with the LSTM model, the performance was then compared to a benchmark model. Comparing to the benchmark model is necessary to demonstrate the prediction capabilities of the LSTM model compared to another prediction model. Benchmarks are used to check whether the prediction model has adequately utilized the available information and to measure the accuracy of the prediction. The naïve prediction model is one of the most used benchmark models for time series prediction. It is simple and effective and often used as a benchmark against the most of sophisticated models. For naïve predictions, all predictions are set to be the value of the last observation

$$\hat{y}_{T+h} = y_T,$$

where $\hat{y}$ is the predicted value, y is the actual value, T is the time and h is the prediction horizon. The naïve prediction model has proven to work well for economic and financial time series data, where the values follow a random walk and does not include seasonality. (Bibhuti et al. 2019)

The LSTM model was developed only with AAPL. After the initial choices of parameters based on the theory, all the experiments and further adjustments were iteratively made by testing the model with AAPL. After optimizing the model with AAPL, it was tested with the other stocks the test the model's ability to generalize for other stocks.

5.3 Level 1 depth prediction

The level 1 depth predictions were completed by predicting separately the bid side and the ask side. The prediction results were surprising, the LSTM model did not present any prediction power for level 1 depth. None of the prediction results were significant and none of them beat the naïve prediction model. The poor performance was experienced with both sides of the LOB. Next, the poor performance of the LSTM model in level 1 depth prediction is analyzed.

The biggest reason for the poor performance of the LSTM model is that the prediction problem was not suitable for regression model due the nature of level 1 order quantity. Most of the time level 1 quantity remained unchanged, which is the main factor for the naïve model to outperform the LSTM prediction model. The LSTM prediction model is constantly adjusting predictions slightly based on the new sample. However, the slight adjustments are insufficient in this kind of prediction task since the actual values are not changing slightly. Instead of a regression prediction model, a conditional prediction model would be more suitable for this prediction task. Microstructure of the level 1 ask quantity predictions compared to the actual values during 100 samples is represented in Figure 5.2. The actual values are represented by a magenta curve and the predicted values by a blue curve.

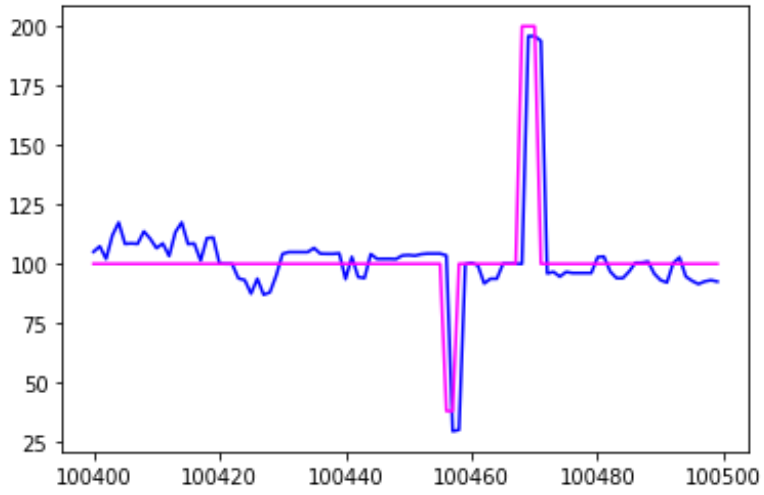

Figure 5.2. *Predictions and actual values of the level 1 ask quantity of AAPL over 100 samples. Magenta: actual quantities, Blue: predicted quantities.*

The predictions are slightly adjusted on short intervals, but the LSTM model is not capable of predicting the non-changing quantity of 100. Similar market movements were observed with the bid side of LOB and the LSTM model faced the same problems while making predictions.

The first reason why the LSTM model is performing poorly compared to the naïve prediction model is that for most of the samples, the level 1 ask quantity does not change compared to the previous sample. Thus, the naïve prediction is exactly correct and does not generate any prediction error. With AAPL validation set, 572733 samples out of the total 618920 samples, the level 1 ask quantity remained unchanged compared to the previous sample. This means that approximately for 92.5% of the samples, the naïve prediction model predicted the exactly correct value. The big proportion of samples with unchanging quantities is explained by the behavior of the market participants posting limit orders instead of only market orders, another explanation being the nature of the level 1 ask quantity. The level 1 ask quantity will remain the same when limit orders arrive to the bid side or to the level 2 or below of the ask side. As the LSTM model fails to learn to adjust to the unchanging quantities, it will perform poorly.

The second reason why the LSTM model performs poorly is the inability to learn the changes in the quantities. The LSTM model usually makes slight changes to the predictions, but the quantity change usually happens with larger magnitude. This is again explained by the behavior of market participants. Because of the transaction costs, it does not make sense for the market participants to post orders with very small quantities. On the contrary, bigger one-time orders do happen. The distribution of the quantities of posted orders is quite discontinuous, which makes the prediction task less suitable for the regression model. The most common change in the order quantity is 100, occurring 11054 times in the AAPL validation set. This is approximately 1.8% of the samples. The quantity of 100 could be explained by some trading strategy where the stock is sold with chunks of 100. As the LSTM model fails to predict the larger jumps in quantities as well as commonly occurring quantities such as 100, the model will perform poorly.

Due the two issues, this prediction task is not suitable for the regression model. Since the discrete quantity changes such as 0 and 100, the nature of the problem seems more like a classification problem. Thus, classification model could yield better prediction results. For example, Dixon (2018) used 3-class classification model to predict the next price change. Similar approach could be applied to the level 1 depth prediction problem, predicting the classes whether the next quantity change will be neutral, downwards, or upwards.

5.4 Multi-level depth prediction

The multi-level depth predictions were completed by taking the top 5 levels of both sides of the LOB and predicting their summarized quantity. The developed LSTM model demonstrated prediction capabilities with AAPL, but it was not able to generalize for the

other stocks, performing poorly with GOOG, INTC, FB, and MSFT. The model was optimized to make predictions with AAPL, and it demonstrated prediction power outperforming the naïve prediction model with AAPL. With other stocks, the naïve prediction model gave better prediction results. MSE of the predictions are represented in Table 5.1., best score being underlined.

Table 5.1. *Results of the multi-level depth prediction. MSEs of the LSTM model and the naïve model with each stock. The LSTM model outperformed the naïve model only with AAPL.*

Model/Stock	AAPL	FB	GOOG	INTC	MSFT
LSTM	<u>37087</u>	5063032	7778	18013338	26216036
Naïve	45276	<u>327946</u>	<u>2875</u>	<u>1671452</u>	<u>270499</u>

With AAPL, the LSTM model outperformed the naïve prediction model having significantly lower MSE. But with the other stocks, the LSTM model could not outperform the naïve prediction model. In general, the performance of both models varied a lot between the other stocks. The stocks that have higher MSE with the naïve model, the quantity is more volatile. The LSTM model performs especially poorly with FB, INTC and MSFT, which have the most volatile quantities. With AAPL and GOOG, the LSTM model performs better than with the other stocks. However, the LSTM model still loses the naïve prediction model with GOOG by some margin, even though GOOG has less volatile quantities than AAPL. Since the LSTM model performed poorly with the other stocks than AAPL, the model would require changes to improve its performance. Figures of the performances of the LSTM model with each stock are represented in Appendix A. Next, the prediction results are analyzed in detail.

The LSTM model performed well with AAPL. To investigate the details of the predictions, the microstructure of the predictions is represented in Figure 5.3. The actual values are represented by the magenta curve and the predicted values by the blue curve.

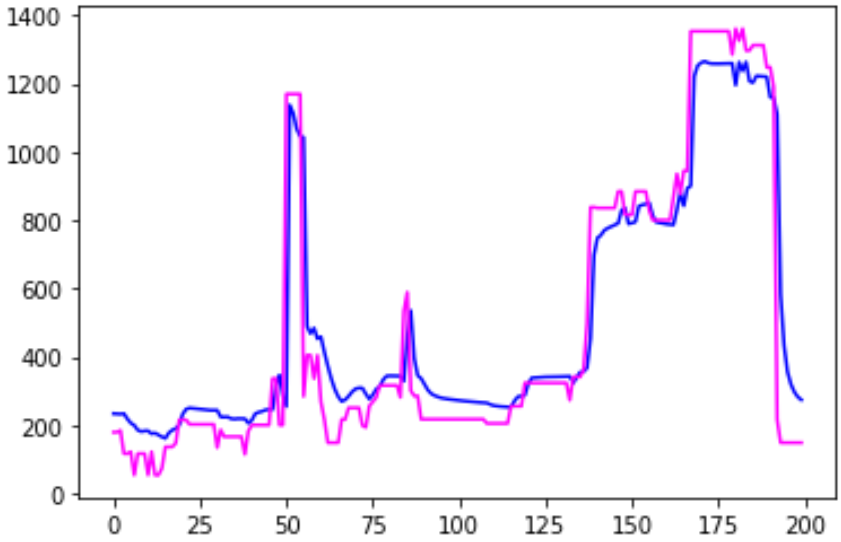

Figure 5.3. *Predictions and actual values with AAPL during 200 samples. Magenta: actual quantities, Blue: predicted quantities.*

Unlike with other stocks, the direction of the prediction error is not constant, sometimes the predicted values are too big and sometimes too small. This finding means that there does not exist any constant bias and the LSTM model is performing well to adapt and predict the multi-level depth.

The LSTM model performed poorly with FB. In the prediction results, the MSE is significantly larger for the LSTM model than it is for the naïve prediction model. To further investigate the performance of the model, the microstructure is of the predictions with FB are represented in Figure 5.4. The actual values are represented by the magenta curve and the predicted values by the blue curve.

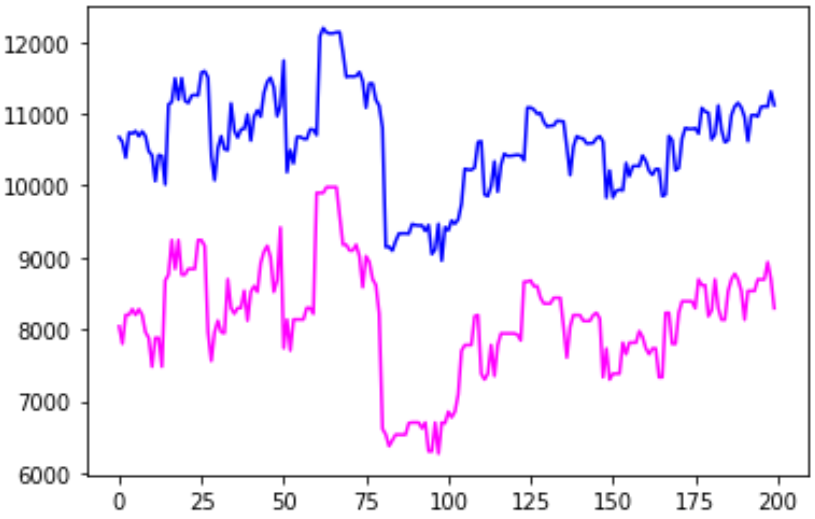

Figure 5.4. *Predictions and actual values with FB during 200 samples. Magenta: actual quantities, Blue: predicted quantities.*

MSE as a metric has a tendency to over-estimate the poor performance of the model and it can be observed that even though the prediction error is very large, the model is able to adjust to the market movements quite well. It seems like that the model can adjust to the multi-level depth changes but does not learn to adapt to the absolute level. This kind of performance continues for the full trading day, the LSTM model is constantly predicting the multi-level depth to be approximately two thousand too large. The LSTM model can only capture the pattern and not the trendline.

Similar performance was observed with INTC and MSFT, the prediction errors were large, and the predicted value was constantly larger than the actual value. Unlike with FB, the performance got better towards the end of the validation set, prediction error being significantly larger at the beginning of the validation set than at the end of the validation set. The performance of the LSTM model with MSFT is represented in Figure 5.5. The actual values are represented by the magenta curve and the predicted values by the blue curve.

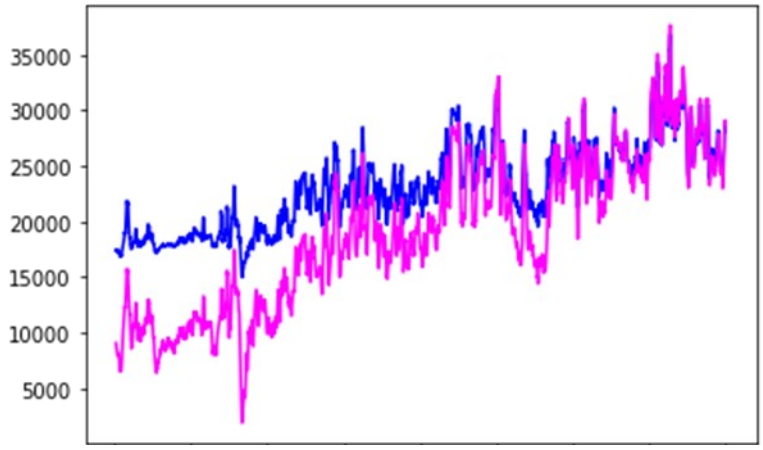

Figure 5.5. *Predictions and actual values with MSFT during full trading day. Magenta: actual quantities, Blue: predicted quantities.*

The LSTM model is not capturing the linear trendline. In the LSTM model, the bias term in final layer b_y is responsible for changing the scale of the series without having an impact to the pattern recognition. In this case, the bias term was not able to capture the linear trendline. The reason for this could be that the model did not learn to adjust the bias term and more training could lead to correct scale predictions. The model did run fewer epochs with FB, INTC and MSFT than with AAPL and GOOG, which could explain their larger errors. This could be explained by the early stopping having been triggered too early and increasing the patience term may improve the performance of the LSTM model.

The main driver of the prediction errors is the inability of the model to capture the linear trendline. For further improvement, a linear autoregressive component could be added to the prediction model. The LSTM would then try capture the patterns and the linear autoregressive component would try to capture the trendline. To otherwise improve the prediction results, better models could be found by continuing empirical research. The LSTM model could be optimized for each individual stock separately to make better predictions or the whole model architecture and learning methods could be changed to seek generalization capabilities. Without a doubt, better models could be found by continuing empirical research. Also, to improve the prediction results in general, feature engineering could be applied to the LOB data as Ntakaris et al. (2019) suggests. Suitable features for this prediction task could be for example bid-ask spread, mean quantity and quantity derivation.

6. CONCLUSION

This thesis studied the use of neural networks and their prediction capabilities for stock market liquidity with LOB data. A new LSTM model was developed to predict stock market liquidity and it was tested against the naïve prediction model to demonstrate the prediction power. The prediction task was defined as a time series regression problem of the next step of level 1 depth and multi-level depth of the LOB, which were used as a measure of the stock liquidity. The neural networks were trained, tested and validated with LOB data of five different stocks during five full trading days.

The first research question was to assess whether the neural networks can predict the stock market liquidity. None of the experimented models showed any prediction capability for the level 1 depth prediction task. The poor performance of the neural network prediction models is explained by the unsuitability of the prediction task. Due the discreteness of the data changes in level 1 order quantity, the problem was found out to be more like a classification task than a regression task. For further model development, reproduction with classification model is suggested.

For multi-level depth prediction task, the developed LSTM model showed prediction power with AAPL. The model was optimized with AAPL, which explains its success compared to other stocks. The finding is narrow, providing proof of prediction power for one model and for one stock only. However, the finding of the prediction evidence encourages for further research to develop more robust and accurate prediction models for stock market liquidity using neural networks.

The second research question was to assess the model's performance against the naïve prediction model. For multi-level depth prediction task, the developed LSTM model outperformed the naïve prediction model with AAPL having significantly lower MSE. With GOOG, the LSTM model lost to the naïve prediction model by some margin. With FB, INTC and MSFT, the LSTM model performed poorly and had large MSE compared to the naïve prediction model. The poor performance is mostly explained by the inability of the LSTM model to capture the linear trendline, capturing only the patterns. For further model development, a linear autoregressive component is suggested to capture the trendline as well.

As machine learning methods are developed mainly through empirical testing, better performing models could be simply found by continuing empirical testing. The developed LSTM model outperformed the naïve prediction model with the one stock that was used

to optimize it. The developed model could outperform the naïve prediction model with other stocks as well if it would be separately optimized for each stock. For further research, exploratory model development, individual model optimization, a linear autoregressive component and feature engineering is suggested.

BIBLIOGRAPHY

Ajina, A., Sougne, D. & Lakhal, F. (2015). Corporate disclosures, information asymmetry and stock-market liquidity in France, Journal of applied business research, Vol. 31(4), pp. 1223–1232.

Aldridge, I. (2013). High-frequency trading a practical guide to algorithmic strategies and trading systems, 2nd ed. Wiley, Hoboken, N.J.

An, Y. & Chan, N.H. (2017). Short-Term Stock Price Prediction Based on Limit Order Book Dynamics, Journal of Forecasting, Vol. 36(5), pp. 541–556.

Andreeva, N.A. & Chaban, V.V. (2015). Global minimum search via annealing: Nanoscale gold clusters, Chemical Physics Letters, Vol. 622 pp. 75–79.

Avellaneda, M., Reed, J. & Stoikov, S. (2011). Forecasting prices from level-I quotes in the presence of hidden liquidity, Algorithmic Finance, Vol. 1(1), pp. 35–43.

Barardehi, Y.H., Bernhardt, D. & Davies, R.J. (2019). Trade-time measures of liquidity, Review of Financial Studies, Vol. 32(1), pp. 129–179.

Basheer, I.A. & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application, Journal of microbiological methods, Vol. 43(1), pp. 3–31.

Biais, B., Declerck, F. & Moinas, S. (2017). Who supplies liquidity, how and when? IDEAS Working Paper Series from RePEc.

Bibhuti, S., Ramakar, J., Ansuman, S. & Deepak, K. (2019). Long short-term memory (LSTM) recurrent neural network for low-fow hydrological time series forecasting, Acta geophysica, 2019, Vol.67(5), p.1471–1481.

Box, G.E.P. (2016). Time series analysis: forecasting and control, 5th ed. Wiley, Hoboken, New Jersey.

Breen, W.J., Hodric, L.S. & Korajczyk, R.A. (2002). Predicting Equity Liquidity, Management Science, Vol. 48(4), pp. 470–483.

Byung-Joo, K. (2012). Improved multi layer perceptron with a prior knowledge applied system identification, Convergence and Hybrid Information Technology, Vol. 7425, pp. 165–172.

Cao, J. & Wang, J. (2019). Stock price forecasting model based on modified convolution neural network and financial time series analysis, International Journal of Communication Systems, Vol. 32(12).

Cartea, Á & Jaimungal, S. (2013). Modelling Asset Prices for Algorithmic and High-Frequency Trading, Applied Mathematical Finance, Vol. 20(6), pp. 512–547.

Castellani, M. (2018). Competitive co-evolution of multi-layer perceptron classifiers, Soft Computing, Vol. 22(10), pp. 3417–3432.

Chow, T.W.S. & Cho, S. (2007). Neural networks and computing learning algorithms and applications, Imperial College Press, London.

Crisci, C., Ghattas, B. & Perera, G. (2012). A review of supervised machine learning algorithms and their applications to ecological data, Ecological Modelling, Vol. 240 pp. 113–122.

Dixon, M. (2018). Sequence classification of the limit order book using recurrent neural networks, Journal of Computational Science, Vol. 24, pp. 277–286.

Elezović, S. (2009). Functional modelling of volatility in the Swedish limit order book, Computational Statistics and Data Analysis, Vol. 53(6), pp. 2107–2118.

Frey, S. & Grammig, J. (2006). Liquidity supply and adverse selection in a pure limit order book market, Empirical Economics, Vol. 30(4), pp. 1007–1033.

Foucalt, T.A., Röell, A. & Pagano, M. (2013). Market liquidity: theory, evidence, and policy, New York, Oxford University Press.

Galeshchuk, S. (2016). Neural networks performance in exchange rate prediction, Neurocomputing, Vol. 172 pp. 446–452.

Graupe, D. (2013). Principles of artificial neural networks, 3rd ed. World Scientific, Singapore.

Graves, A. (2012). Supervised Sequence Labelling with Recurrent Neural Networks, Studies in computational intelligence, Vol. 385, pp. 1–141.

Graves, A. (2014). Generating Sequences With Recurrent Neural Networks, pp. 1–43.

Geithner, T. (2007). Liquidity and Financial Markets. Keynote address at the 8th Annual Risk Convention and Exhibition, Global Association of Risk Professionals, New York City.

Hang, S.T. & Aono, M. (2017). Bi-linearly weighted fractional max pooling: An extension to conventional max pooling for deep convolutional neural network, Multimedia tools and applications, Vol. 76(21), pp. 22095–22117.

Hashemi F., A., Madanifar, F. & Abbasi, M. (2018). Implementation of multilayer perceptron (MLP) and radial basis function (RBF) neural networks to predict solution gas-oil ratio of crude oil systems, Petroleum, Vol. 6(1), pp. 80–91.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory, Neural computation, Vol. 9(8), pp. 1735–1780.

Hu, Y. (2010). A single-layer perceptron with PROMETHEE methods using novel preference indices, Neurocomputing, Vol. 73(16), pp. 2920–2927.

Härdle, W.K., Hautsch, N. & Mihoci, A. (2012). Modelling and forecasting liquidity supply using semiparametric factor dynamics, Journal of Empirical Finance, Vol. 19(4), pp. 610–625.

Kale, J.R. & Loon, C.Y. (2010). Product market power and stock market liquidity, Journal of Financial Markets, Vol. 14(2), pp. 376–410.

Kang, H. & Kim, S. (2018). Order Imbalance and Return Predictability: Evidence from Korean Index Futures, Information, Vol. 21(4), pp. 1283–1291.

Keller, J.M., Liu, D. & Fogel, D.B. (2016). Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation, IEEE Press/Wiley, Hoboken, New Jersey.

Kercheval, A.N. & Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines, Quantitative Finance: Special Issue on High Frequency Data Modeling in Finance, Vol. 15(8), pp. 1315–1329.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice-Hall, New York.

Kingma, D. & Ba, J. (2017). Adam: A Method for Stochastic Optimization.

Kumar, S., Singh, M.P., Goel, R. & Lavania, R. (2013). Hybrid evolutionary techniques in feed forward neural network with distributed error for classification of handwritten Hindi 'SWARS', Connection Science, Vol. 25(4), pp. 197–215.

Li, X., Peng, L., Yao, X., Cui, S., Hu, Y., You, C. & Chi, T. (2017). Long short-term memory neural network for air pollutant concentration predictions: Method development and evaluation, Environmental Pollution, Vol. 231, pp. 997–1004.

Liu, Y., Liu, S., Wang, Y., Lombardi, F. & Han, J. (2018). A Stochastic Computational Multi-Layer Perceptron with Backward Propagation, IEEE Transactions on Computers, Vol. 67(9), pp. 1273–1286.

Livingstone, D.J. (2009). Artificial Neural Networks: Methods and Applications.

Nousi, P., Tsantekidis, A., Passalis, N., Ntakaris, A., Kanniainen, J., Tefas, A., Gabbouj, M. & Iosifidis, A. (2019). Machine Learning for Forecasting Mid-Price Movements Using Limit Order Book Data, IEEE Access, Vol. 7(99), pp. 64722–64736.

Ntakaris, A., Magris, M., Kanniainen, J., Gabbouj, M. & Iosifidis, A. (2018). Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods, Journal of Forecasting, Vol. 37(8), pp. 852–866.

Ntakaris, A., Mirone, G., Kanniainen, J., Gabbouj, M. & Iosifidis, A. (2019). Feature Engineering for Mid-Price Prediction with Deep Learning, IEEE Access, Vol. 7(99), pp. 82390–82412.

Palguna, D. & Pollak, I. (2016). Mid-Price Prediction in a Limit Order Book, IEEE Journal of Selected Topics in Signal Processing, Vol. 10(6), pp. 1083–1092.

Parra, O., Garcia, G. & Daza, B. (2014). LAN traffic forecasting using a multi layer perceptron model, Internet of Things, Smart Spaces, and Next Generation Networks and Systems, Vol. 8638, pp. 399–407.

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. & Iosifidis, A. (2018). Temporal Bag-of-Features Learning for Predicting Mid Price Movements Using High Frequency Limit Order Book Data, IEEE Transactions on Emerging Topics in Computational Intelligence, Vol. 99, pp. 1–12.

Piotrowski, A.P. & Napiorkowski, J.J. (2013). A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling, Journal of Hydrology, Vol. 476, pp. 97–111.

Poznyak, T., Chairez, O.I. & Poznyak, A.S. (2018) Ozonation and biodegradation in environmental engineering, Elsevier.

Rawat, A. & Solanki, A. (2020). Sequence Imputation using Machine Learning with Early Stopping Mechanism, International Conference on Computational Performance Evaluation, IEEE, pp. 859–863.

Rana, A., Singh, R, Bijalwan, A. & Bahuguna, H. (2018). Application of Multi Layer (Perceptron) Artificial Neural Network in the Diagnosis System: A Systematic Review, International Conference on Research in Intelligent and Computing in Engineering, pp. 1–6.

Rösch, C. & Kaserer, C. (2013). Market liquidity in the financial crisis: The role of liquidity commonality and flight-to-quality, Journal of Banking and Finance, Vol. 37(7), pp. 2284–2302.

Sak, H., Senior, A. & Beaufays, F. (2014). Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition, pp. 1–5.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview, Neural Networks, Vol. 61 pp. 85–117.

Schwartz, R.A. & Weber, B.W. (1997). Next-Generation Securities Market Systems: An Experimental Investigation of Quote-Driven and Order-Driven Trading, Journal of Management Information Systems, Vol. 14(2), pp. 57–79.

Sharma, A. (2018). Guided Stochastic Gradient Descent Algorithm for inconsistent datasets, Applied Soft Computing Journal, Vol. 73 pp. 1068–1080.

Sharma, R., Nori, A. & Aiken, A. (2014). Bias-variance tradeoffs in program analysis, Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on principles of programming languages, Vol. 49(1), pp. 127–137.

Siikanen, M., Kannianen, J. & Luoma, A. (2017). What Drives the Sensitivity of Limit Order Books to Company Announcement Arrivals? Economic Letters, Vol. 159, pp. 65–68.

Siikanen, M. (2018). Investors, Information Arrivals, and Market Liquidity: Empirical Evidence from Financial Markets, Doctoral dissertation, Tampere University of Technology, pp. 1–68.

Soutner, D. & Müller, R. (2013). Application of LSTM neural networks in language modelling, Text, Speech, and Dialogue, Vol.8082, p.105–112.

Speck-Plancke, A. & Cordeiro, M.N. (2015). Artificial Neural Networks, Methods in Molecular Biology, Vol. 1260, pp. 45–64.

Tavana, M., Abtahi, A., Di Caprio, D. & Poortarigh, M. (2018). An Artificial Neural Network and Bayesian Network model for liquidity risk assessment in banking, Neurocomputing, Vol. 275, pp. 2525–2554.

Thimm, G. & Fiesler, E. (1997). High-order and multilayer perceptron initialization, IEEE Transactions on Neural Networks, Vol. 8(2), pp. 349–359.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. & Iosifidis, A. (2018). Using Deep Learning for price prediction by exploiting stationary limit order book features, Applied Soft Computing Journal, Vol. 93, pp. 1–10.

Übeyli, E.F. (2005). A Mixture of Experts Network Structure for Breast Cancer Diagnosis, Journal of Medical Systems, Vol. 29, pp. 569–579.

Verlag, C. (2008). Limit Order Book Dynamics and Asset Liquidity, Doctoral dissertation, University of Zurich. pp. 1–149.

Wang, Y., Mangu, L. & Decker, K. (2019). Model-based Reinforcement Learning for Predictions and Control for Limit Order Books, Association for the Advancement of Artificial Intelligence.

Winkler, J. & Vogelsang, A. (2016) Automatic classification of requirements based on convolutional neural networks, IEEE 24th International Requirements Engineering Conference Workshops, pp. 39–45.

Witten, I.H., Frank, E. & Hall, M.A. (2011). Data mining practical machine learning tools and techniques, 3rd ed. Elsevier/Morgan Kaufmann.

Wu, H. & Gu, X. (2015). Max-pooling dropout for regularization of convolutional neural networks, Neural Information Processing, Vol.9489, pp. 46–54.

Xu, K., Gould, M.D. & Howison, S.D. (2019). Multi-Level Order-Flow Imbalance in a Limit Order Book, Market microstructure and liquidity, Vol.4, pp. 1–42.

Yoon, K. (2014). Convolutional Neural Networks for Sentence Classification, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1746–1751.

Zaccone, G. (2016). Getting started with TensorFlow: get up and running with the latest numerical computing library by Google and dive deeper into your data, Packt Publishing.

Zhang, Y., Wang, S., Ji, G. & Phillips, P. (2014). Fruit classification using computer vision and feedforward neural network, Journal of Food Engineering, Vol. 143 pp. 167–177.

Zhang, Z., Zohren, S. & Roberts, S. (2018). BDLOB: Bayesian Deep Convolutional Neural Networks for Limit Order Books, Third workshop on Bayesian Deep Learning, pp. 1–6.

Zhang, Z., Zohren, S. & Roberts, S. (2019). DeepLOB: Deep Convolutional Neural Networks for Limit Order Books, IEEE Transactions on Signal Processing, Vol. 67(11), pp. 3001–3012.

Zheng, B., Moulines, E. & Abergel, F. (2013). Price jump prediction in a limit order book, IDEAS Working Paper Series from RePEc, pp. 1–16.

APPENDIX A. PREDICTION RESULTS

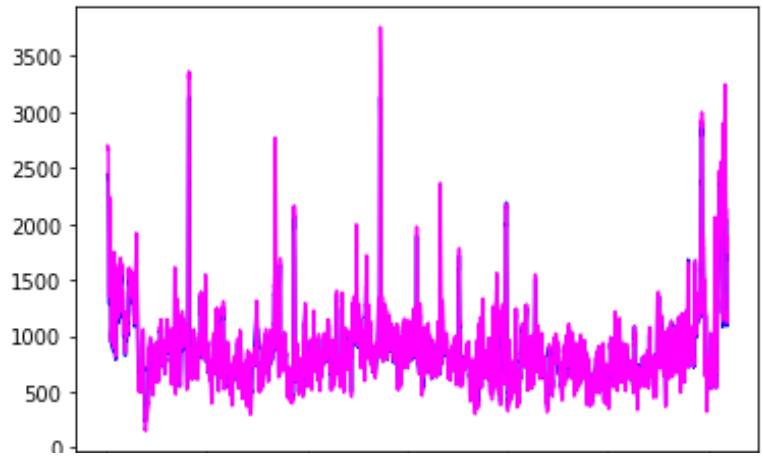

Figure 1. *Predictions and actual values with* ***Apple*** *stock during full trading day. Magenta: actual quantities, Blue: predicted quantities.*

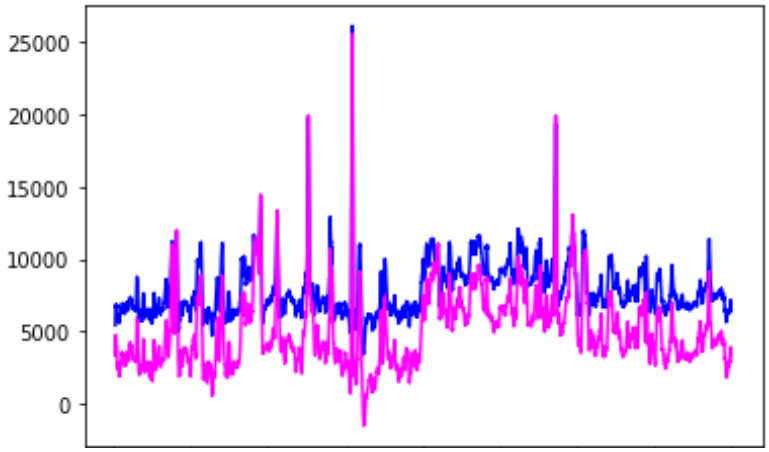

Figure 2. *Predictions and actual values with* ***Facebook*** *stock during full trading day. Magenta: actual quantities, Blue: predicted quantities.*

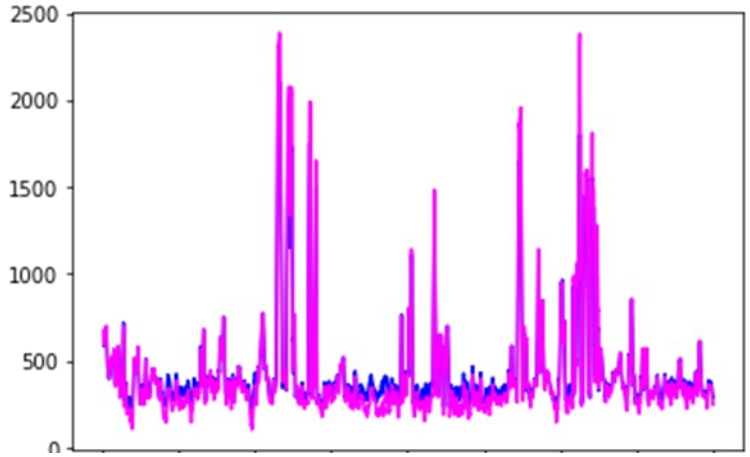

Figure 3. *Predictions and actual values with* ***Google*** *stock during full trading day. Magenta: actual quantities, Blue: predicted quantities.*

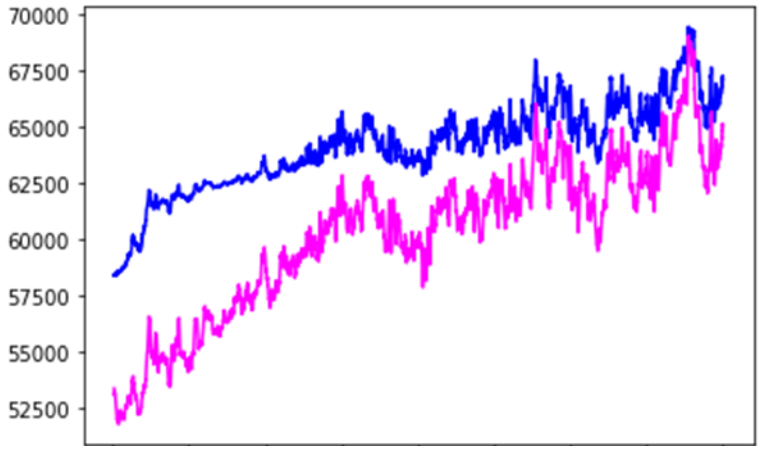

Figure 4. *Predictions and actual values with* ***Intel*** *stock during full trading day. Magenta: actual quantities, Blue: predicted quantities.*

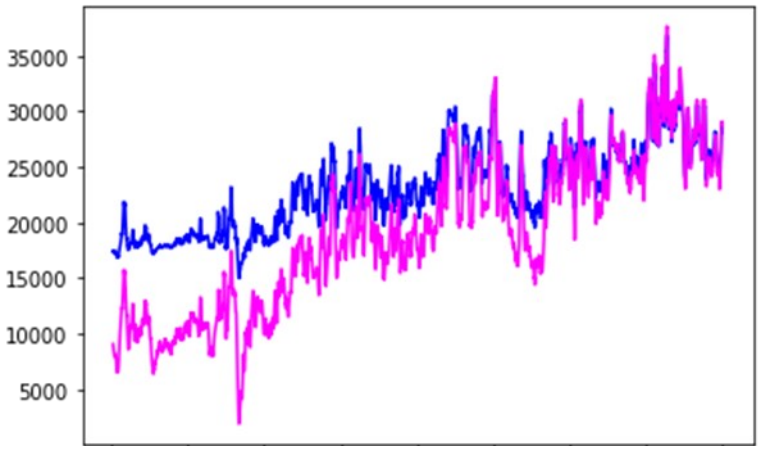

Figure 5. *Predictions and actual values with* ***Microsoft*** *stock during full trading day. Magenta: actual quantities, Blue: predicted quantities.*