Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

GPU Accelerated Limit Order Book Modeling Pipeline

Scientific Students' Association Report

Author:

Viktor Burján

Advisor:

Bálint Gyires-Tóth, Ph.D.

2020

Contents

Kivonat

A pénzügyi modelleket korábban gyakran ökonometriai és matematikai ismeretekre építették, azonban napjaink technológiája lehetővé teszi a nagy mennyiségű adathalmazon dolgozó algoritmusok futtatását is. Ezek közé tartoznak a neurális hálók, melyek futtathatók párhuzamos végrehajtást támogató hardvereken - ilyen pl. a GPU (Graphical Control Unit - videókártya). A párhuzamos futtatás nagyban lecsökkenti a modellek futásidejét, azonban az adatok előfeldolgozása legtöbb esetben továbbra is hagyományos módon, a CPU-n (Central Processing Units - processzor) történik.

A dolgozatban egy alkalmazás kerül bemutatásra, mely nem csak a modellek futtatására képes kihasználni a GPU által nyújtott párhuzamos futtatás lehetőségét, hanem az előfeldolgozás lépéseire is. Az alkalmazás pénzügyi adatokat dolgoz fel – az ajánlati könyvet. A futtatás során adatgyűjtés, az adatok előfeldolgozása, normalizálása, majd tanító mintákba szervezése, valamint a neurális hálók tanítása is megtörténik. A dolgozat tartalmazza a modellek kiértékelését, illetve összehasonlítja a futásidőt a tradícionálisan használt módszerekhez képest.

Abstract

Modeling financial processes is present for a long time by econometric and mathematical models, however, today's technology enables data-driven algorithms to be applied in this field. This includes models such as artificial neural networks, which can be executed on massively parallel hardware, such as Graphical Control Units (GPUs). These technologies allow running models quicker than before, but preprocessing of data is mostly done in a traditional way - e.g on the Central Processing Units (CPUs).

In this work, a GPU accelerated pipeline is proposed, which aims to apply the GPUs parallel execution mode not just on the model execution, but the data preprocessing stages as well. The pipeline handles a specific type of financial data – limit order books – and handles data collection, order book preprocessing, data normalisation, batching into training samples and training of deep neural networks. Measurements of the performance and inference are taken and compared with benchmarking methods.

Chapter 1

Introduction

A limit order book (LOB) contains the actual buy and sell limit orders that exist on a financial exchange at a given time.[1] Every day on major exchanges, like New York Stock Exchange, the volume of clients' orders can be tremendous, and each of them has to be processed and recorded in the book.

Deep learning models are capable of representation learning and modeling jointly [14], thus, other machine learning methods (which requires roboust feature engineering) can be outperformed in case a large amount of data is available. This makes the application of deep learning-based models is particularly reasonable for LOB data.

The parameter and hyperparameter optimization of deep learning models are typically performed on Graphic Processing Units (GPUs). Still, nowadays deep learning solutions preprocess the data mostly on the CPU (Central Processing Unit) and transfer it to GPUs afterwards. Finding the optimal deep learning model involves not only massive hyperparameter tuning but a large number of trials with different data representations as well. Therefore, CPU based preprocessing introduces a critical computational overhead, which can severely slow down the complete modeling process.

In this document, a GPU accelerated pipeline is proposed for LOB data. The pipeline preprocesses the orders and converts the data into a format that can be used in the training and inference phases of deep neural networks. The pipeline can process the individual orders that are executed on the exchange, extract the state of the LOB with a constant time difference in-between, then batch these data samples together, normalise them, feed them into a deep neural network and initiate the training process. When training is finished, the resulted model can be used for

[1] Such an order book does not exist for the so called dark pool, because the orders are typically not published in dark pools [9].

inference and evaluation. In order to define output values for supervised training, further steps are included in the pipeline. A typical step is calculating features of the limit order book (mid-price or volume weighted average price, for example) for future timesteps, and labelling the input data based on these data.

The main goal of the proposed method is to reduce the computational overhead of hyperparameter-tuning for finding the optimal model and make inference faster by moving data operations to GPUs. As data preprocessing is among the first steps in the modeling pipeline, if the representation of the order book for modeling is changed, the whole process must be rerun. Thus, it is critical, that not only the model training and evaluation, but the data loading and preprocessing are as fast as possible.

Although deep learning models trained on the data processed in such a way should be able to learn on the provided data, this is not yet included. However, a few widely-spread machine learning models are run as benchmarks on the processed dataset.

The remaining part of this document is organized as follows. The current section provides a brief overview of the concepts used in this document. Chapter 2 explores previous works done in the related domains. Chapter 3 introduces definitions of the topic and Chapter 4 explores the used learning algorithms. Chapter 5 breaks down our architecture into separate components and gives a brief overview of them, while the proposed methods and functionality is presented in Chapter 6 in details. Chapter 7 shows the implementation details of the previously presented algorithms. Chapter 8 evaluates the performance of both pre-processing and prediction. Finally, the possible applications are discussed and conclusions are drawn.

Chapter 2

Previous works

2.1 Deep learning

Machine learning and deep learning algorithms have revolutionized many research areas, not only financial modelling. Some examples are mentioned of all areas, but only financial modeling is covered in-depth.

The idea of neural networks is not a new concept, they exist since the middle of the 20th century, however, the topic was not researched for a period of time. A brief summary of neural network history can be read in [5].

While deep fully connected neural networks were useful, at the end of the 20th century, other, different models were invented. One of them is the convolutional neural network, which is particularly useful for recognizing images - but it is used in many other areas, such as natural language processing [7]. Convolutional networks can also be applied for sequence-to-sequence learning - e.g when translating from one language to one another [10].

Another tool for deep learning modeling is the Recurrent Neural Network. While the previous models are suitable for recognizing specific patterns, they don't have a sense of sequences or history. On the contrary, this is the strength of the Recurrent Neural Network (RNN) - they remember the past - not just the training samples - and this influences their decisions. They do this by having previous outputs as inputs, and maintaining an inner state.

The most often used recurrent network is the LSTM [11], or Long Short-Term Memory, which defines specific rules to decide what inputs to use, or to forget in the given layer.

The components described above are just the most fundamental, basic building blocks of deep learning models. To accomplish complex modeling exercises, researchers usually extend, modify or combine these models to achieve the desired results.

2.2 Financial modeling

As the volume of the data increases, data driven machine learning algorithms can exploit deeper context in the larger amount of data, but financial modeling hasn't just started with deep learning models, but has quite a long history. There already have been many attempts to model markets with mathematical methods earlier. [3] presents one of the most famous models for option pricing, written in 1973, and is still used today by financial institutions and researchers who study this topic.

In the previous century, many models were built on proven mathematical (mostly statistical and probabilistic) formulas. [31] gives a thorough summary of financial econometric models, from 2008. It mostly mentions likelihood and estimation models. An example for modling with such a statistical approach is shown in [22].

Also, there have been multiple articles published on limit order books specifically. [19] examines traders behaviour with respect to the order book size and shape, while e.g [6] is about modelling order books using statistical approaches. As order book data is many magnitudes larger than simpler financial data-sets, which just contain specific features as time-series (such as VWAP and middle-price), the processing of these were much more difficult earlier.

2.3 Financial applications of deep learning

As limit order books are used in exchanges, many research aim to outperform the market by utilizing LOB data. The most common deep learning models for predicting the price movement are the convolutional neural networks (CNNs) and Long Short-Term Memory (LSTM) networks. [27] emphasizes the time series-like properties of the order book states, and uses specific data preprocessing steps, accordingly. The publicated results are very greatly detailed, the authors run the algorithms using many different parameter-setting, and also provide the used network architectures. One quite unique approach is shown in [25], where instead of using time-series like data, the authors have chosen to just use still images, generated from the top 30

Dow Jones stocks - however, they only use the price data, and not the limit order book itself.

A benchmark LOB dataset is published [18], however, it is not widely spread yet. It provides normalised time-series data of 10 days from the NASDAQ exchange, which researchers can use for predicting the future mid-price of the limit order book. The LOBSTER dataset provides data from Nasdaq [2] and is frequently used to train data driven models.

[29] shows a reinforcement learning based algorithm, which is used for cryptocurrency market making. The authors define 2 types of orders: market and limit, and approach the problem in a market maker's point of view. They experiment with multiple reward functions, such as Trade completion or P&L (Profit and Loss). They train deep learning agents using data of the Coinbase cryptocurrency exchange, which is collected through Websocket connection. [8] gives a comprehensive comparison between LSTM, CNN and fully-connected approaches. They have used a wider time-scale, the data are collected from the S&P index between 1950 and 2016.

Recently, limit order book and other information - which is only available from the exchange - is not the only source of data for prediction. With the continuously evolving IT systems, more and more kinds of datasets are available. These include, for example sentiment data of different social media sites, such as Twitter or Reddit. [1] observes Twitter media to track cryptocurrency news, and also explores machine learning methods to predict the number of future mentions. [15] targets to use Twitter and Telegram data to identify 'pump and dump' schemes used by scammers to gain profits. As deep learning algorithms benefit from larger data-sets, any related information could be useful in the future.

One of the most recent works in the topic is published in [23]. The researchers apply deep reinforcement techniques in the domain. The most fundamental element of their model is the event itself - while other works mostly evaluate order book states over time, they instead put an effort into an event-driven approach. This probably enables the algorithm to adapt easier - the number of executed orders can differ when looking at different periods of the market/asset.

One of the most recent articles yields surprising results in predicting Bitcoin price movements - presented in [12]. The authors use temporal convolutaional networks to achieve a 71 % accuracy on predicting the next price movement. They also mention that cryptocurrency exchanges, such as Coinbase are more likely to be easily predicted by such algorithms, as the used technologies are much more up-to-date compared to a regular exchange.

2.4 Processing large data with GPUs

Graphic Processing Units (GPUs) are much different than CPUs. They are known to be better at executing massively parallel algorithms. However, this parallelism requires that the tasks which are executed don't depend too much on each other. For example, [17] shows an example usage on the role of GPU in a big data processing system. These kind of units are not yet fully utilized for tasks like preprocessing or data manipulation - they are mostly used for running machine learning and deep learning models only, although recently this trend is changing, e.g NVidia's Rapids framework enables everyone to swith to GPU preprocessing seamlessly - just as the user would be acting on a regular CPU, using data frames.

2.5 Cryptocurrencies

Cryptocurrencies are a recently invented new type of financial assets, which offer many benefits compared to the regular, existing currencies.

The first paper published in the topic, which launched the whole cryptocurrency development is published in [16]. The personality of the author, Satoshi Nakamoto is still unknown. The document defines the electronic coin as a chain of digital signatures. The transactions are then grouped into blocks, where each block has a hash, which validates the block. As the hash of each ith block can only be created using the previous, $i-1$th block's hash, these blocks together form a chain, called the blockchain. This mechanism also makes it really difficult for attackers to falsify the transactions done in the blockchain. Each block also has a nonce, which is incrementing block-by-block. The main goal of the hashing - which is done by the Bitcoin 'miners', who run computing algorithms for rewards - is to create a hash starting with a specific number of zeros. This would require to redo the computation if an attacker would like to change anything in the block.

The number of coins in circulation is limited for Bitcoin, in fact, only 21 million coins can exist. These coins can only be acquire using two methods, either mining or by participating in a transaction. Mining rewards are halved each 210,000 mined blocks. Each participant using Bitcoin need to have a public and a private cryptographic key, which ensures the secure usage of the network.

The mechanism described in the previous chapter is also known as proof-of-work - the work is the computation power, which is given by miners, who are running the hashing algorithm. Many cryptocurrencies, such as Bitcoin and Ethereum use the proof-of-work mechanism, however, it's hard to maintain so much computational

power with the growing demand. Hashes are also getting harder and harder to compute. To mitigate this computational power, many cryptocurrencies use another method, which is called the proof-of-stake - e.g the next version of the Ethereum blockchain will use proof-of-stake [1] instead. [24] provides more information on Proof of stake algorithms.

Since the presence of Bitcoin and Ethereum, many other crypto coins have appeared. Most of them claims to have different advantages compared to others. Although there are many of these, it's worth to mention the Ripple (XRP) coin [2] - which was in the top 5 currencies (by market cap) in the recent years.

[1] Ethereum Proof-of-stake information, as of 2020. 10. 21: `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`

[2] `https://ripple.com/files/ripple_consensus_whitepaper.pdf` - as of 2020. 10. 22.

Chapter 3

The Limit Order Book

The LOB consists of many limit orders. Limit orders are defined by clients who want to open a long or short position for a given amount of a selected asset (e.g. currency pair), for a minimum or maximum price, respectively.

The book of limit orders is divided into two parts: bids (buy orders) and asks (sell orders). The bid and ask orders contain the price levels and volumes for which people want to buy and sell an asset. The two sides of the LOB are stored sorted by price - the lowest price of asks and the highest price of bids are stored at the first index - this makes calculations on the Limit Order Book more difficult, as they should always consider the order of the entries. This makes accessing the first few elements quicker as well, as these orders are the most relevant ones. If the highest bid price equals or is more than the lowest ask price (so, the two parts of the order book overlap) the exchange immediately matches the bid and ask orders, and after execution, the bid-ask spread is formed.

The bid with the largest price at a given t time: $b(t)$, while the ask with the smallest price at a given t time: $a(t)$.

Using these notations, the bid-ask spread is defined as

$$spread = a(t) - b(t) \tag{3.1}$$

The middle-price (also referenced as mid-price) is:

$$\frac{a(t) + b(t)}{2} \tag{3.2}$$

which value is within the spread.

Feature engineering can produce further order book representations that are used as inputs or outputs of machine learning methods. A commonly used representation is the Volume Weighted Average Price (VWAP). This feature considers not just the price of the orders, but also their side:

$$\frac{\sum price * volume}{\sum volume} \tag{3.3}$$

It is sometimes considered as a better approximation of the real value of an asset - orders with bigger volume have bigger impact on the calculated price.

A 'snapshot' of the order book contains the exchange's orders, ordered by price at a given t time, represented as

$$a_{depth}, a_{depth-1}, ..., a_1, b_1, b_2, ..., b_{depth-1}, b_{depth} \tag{3.4}$$

, where *depth* is the order book's depth. In this work we use same depth on both bid and ask sides. A snapshot is visualised in Figure 3.1. Other features can be extracted from the order book as well, such as momentum, which are not included in this paper. The orders which are closer to the mid-price are more relevant, as these may get executed before other orders which are deeper in the book.

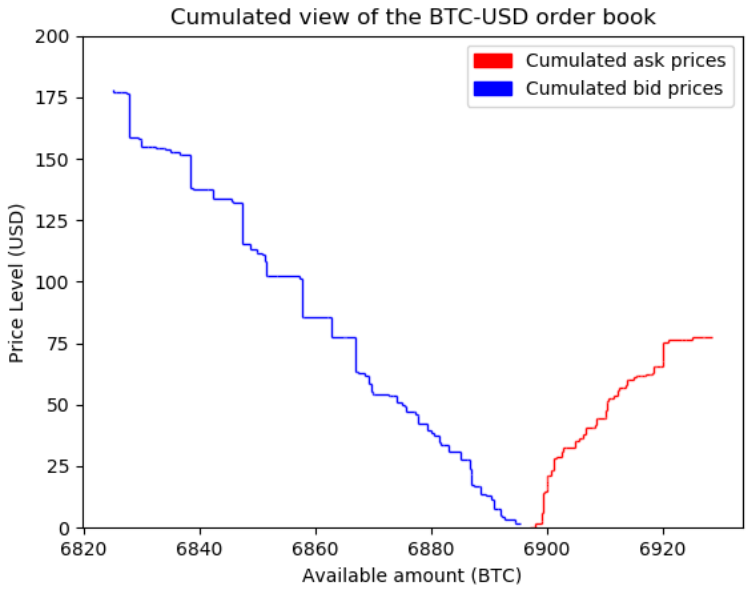


Figure 3.1: Cumulated view of the order book. The plot display the quantity of Bitcoin that can be bought/-sold on a specific price level.

Order book snapshots are often represented on plots using the cumulated quantity of the available asset on the y axis. This can be seen on Figure 3.1. This can be useful for investors, as they can instantly see if the buy or sell pressure is bigger - in the figure, the used LOB depth was 100, yet the quantity of buy orders add up to 175 BTCs, while the first 100 sell orders only make around 75 BTC - but are closer to the middle-price.

It is worth mentioning that Limit Order Book data is not widely available for regular, big exchanges. Large organizations, like hedge funds and other companies pay for this data, as it can be considered a benefit in trading these assets. In fact, the delay of retrieving this data is also important for high-frequency trading. For these reasons, LOB data can be hard to model by individuals with regular assets. However, many online cryptographic exchanges offer free data feeds.

Chapter 4

Machine Learning algorithms

In this chapter, the used algorithms are described in detail. Some of these algorithms were not used for the pipeline, but for benchmarking on the gathered dataset for comparison of the selected methods. Learning methods are usually divided into two groups - machine learning and deep learning algorithms. Machine learning includes any learning algorithm, which is able to change itself when applied on a new data point. Deep learning methods include neural networks and their variants.

4.1 Linear regression

Linear regression is one of the most widely used machine learning algorithms. It is able to grasp linear connection between features of a dataset. It uses a set of features (explanatory variables) to predict the desired feature (dependent variable). Although it has been used for many years, researchers still use it for new applications, as it's results can be interpreted easily, and in many cases, there are linear connections between the features of a dataset. An example of its applications can be read in [21] - the article uses linear regression to predict COVID-19 daily cases, which is one of the most up-to-date use-cases of machine learning methods.

4.2 Random forests

Random forests is a simple ensemble method [4] - which means, it converts weak learners into a common, stronger learner. A general problem with the decision tree algorithm is that it overfits on training data easily - the random forest aims to cancel this effect by aggregating many decision trees.

4.3 Artificial neural networks

Artificial neural networks are the one of the most widely used learning algorithms due to their versatility and learning capabilities. The original idea of the neural network was inspired by the cells of the human brain, which is made up of many small neurons, creating a system which is able to execute complex algorithms. [5] shows the brief history of this method.

Neural networks became usable for practical problems with the invention of the backpropagation algorithm. Backpropagation uses a function which calculates a defined error. This is done by comparing the output of the network to the actual truth. This error is then propagated back on the whole network - changing each weight so that the network keeps evolving and predicts samples with less and less error. In recent years, algorithms which utilize the size of data has gained more and more popularity. Neural networks are one of the algorithms benefiting most of this - the bigger the data gets, the better results the procedure can yield.

4.4 Convolutional neural networks

Convolutional neural networks are mostly used for processing images, as they can model spatial (and temporal) correlation within the data effectively [13] [28]. The first few layers of the convolutional network recognizes the most simple shapes like lines, but as a layer lays deeper in the network, it can recognize more complex images, like faces, cars, and so on.

Convolutional networks also include special layers, such as max pooling layers. These layers are used to reduces the size of the passed activation maps, and also these work as feature extractors.

The output of the convolutional layers is typically fed into a regular neural network, which is responsible to perform the classification or regression from the extracted features. Some methods, like semantic segmentation utilizes convolutional layers only [20].

4.5 Long Short-Term Memory

As stated in the introductory, the previous networks were sufficient to learn on the data they are fed with, but they don't remember these samples. The architecture of a Long Short-Term Memory [11] network addresses this by adding a feedback loop

into the network. This allows the algorithm to learn long-time dependencies from the data.

Chapter 5

System Design

In this section, the overall design of the software components are shown. A high-level overview is shown on Figure 5.1. The components of the pipeline are executed sequentially. When a computation is done, the result is saved to a centralised database, which is accessible by every component. The next step in the pipeline just grabs the needed data from this data store. It's worth to mention that this data store can be changed easily, as the implementation of the storage mechanism is independent of the other components.

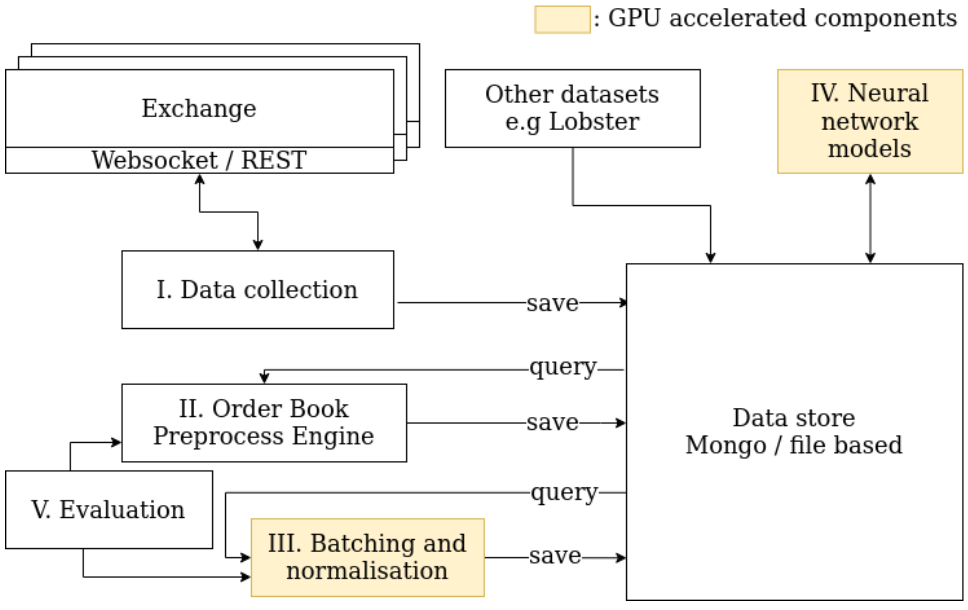


Figure 5.1: Block diagram of the proposed GPU accelerated pipeline

The components shown in Figure 5.1 are implemented as separate Python classes, mostly manipulating Numpy arrays of data. Not every component uses GPU acceleration.

The defined components:

- Data collection: Responsible for collecting data. Our implementation does this using Websockets to connect to the Coinbase Pro exchange.
- Order Book Preprocess Engine: This tool is created to generate snapshots from the collected data, with any time difference between them.
- Batching and normalisation: This component heavily utilises the GPU, and is responsible for putting the processed snapshots into batches. Afterwards, these batches are normalised.
- Neural network models: This component is created to prediction of the next price movement
- Evaluation: A separate component, which is used to take measurements of the previous ones, to evaluate and compare their performance.

5.1 Technology considerations

While coming up with the architectural design of the pipeline, one of the most important goals was to make it as standardised and re-usable as possible. This is done by various decisions:

- Using additional abstraction layers for the re-usable parts. For example, an abstract class is defined for data saving in the crawler, so that it can be changed easily - e.g saving to MongoDb or to any other data store, which has it's interface implemented in the codebase.
- Using standardised data structure for processing. The normalisation only uses numpy arrays, which are of data type float, and are fully compatible with the GPU dataformat.

5.2 Software frameworks

Most of the libraries used in the implementation are considered as industry standard for deep learning. The Python language was chosen due to it's easy use and it's wide-spread usage in the industry.

Nvidia CUDA interaction - which is a huge part of the pipeline - is enabled by a Python library called Numba[1]. It enables writing massively parallel algorithms without using any other language than Python. The library is really easy to use, however, it puts limitation on the written code, so that it can be compiled to run on CUDA easily - Numba only supports array manipulation and methods of the Numpy library, but not other frameworks, such as Pandas. The limitations added by using Numba are not too restrictive when using the just-in-time compilation mode, but higher restrictions are present when the Python code is compiled into CUDA kernels.

Although Numba gives quite an easy method of using the CUDA kernels (a method can be compiled for CUDA with a single annotation), developers have to be careful using these annotations as-is. If not configured correctly - e.g the grid not configured properly - no performance benefit is going to be achieved.

Other libraries have been utilized in the pipeline as well, such as the SortedContainers[2] Python library for the Preprocess Engine. The dates all over the application were parsed by the ciso8601 library, which was remarkably faster when compared with Python's built-in date parsing mechanisms[3]. During development, significant speed-up was achieved using the orjson[4] library instead of the built-in parsers, as JSON saving and loading is extensively used across the code.

The neural network models were implemented using the Keras framework. Benchmarking models were created with various Python libraries, such as sklearn.

[1] `https://numba.pydata.org/`, as of 2020. 10. 27.
[2] `https://pypi.org/project/sortedcontainers/` as of 2020. 10. 27.
[3] Benchmark of ciso8601: `https://github.com/closeio/ciso8601` as of 2020. 10. 27.
[4] `https://github.com/ijl/orjson` as of 2020. 10. 27.

Chapter 6

Proposed methods

In this section, we introduce the proposed GPU accelerated pipeline for LOB modeling and explain the components shown in the previous chapter in details.

6.1 Data collection

The data format should be universal - no exchange or asset specific features should be included in the process. However, to collect a large enough dataset for training and performance evaluation, a data collection tool is a part of the pipeline, that can save every update that's happening on an exchange.

The data collection tool saves the following information from an exchange:

- Snapshot: The tool downloads the current state of the exchange periodically - the length of this period can be supplied when starting the process. The saved data is organised so the snapshots can be queried rapidly by dates.
- Updates: updates that are executed on the exchange and have effect on the order book.

The framework supports multiple assets when (down)loading data, due to the supposed universal features - training with multiple assets may increase the model accuracy [26, 30]. As the number of LOB updates in an exchange can be extremely high, compression is used to store the data.

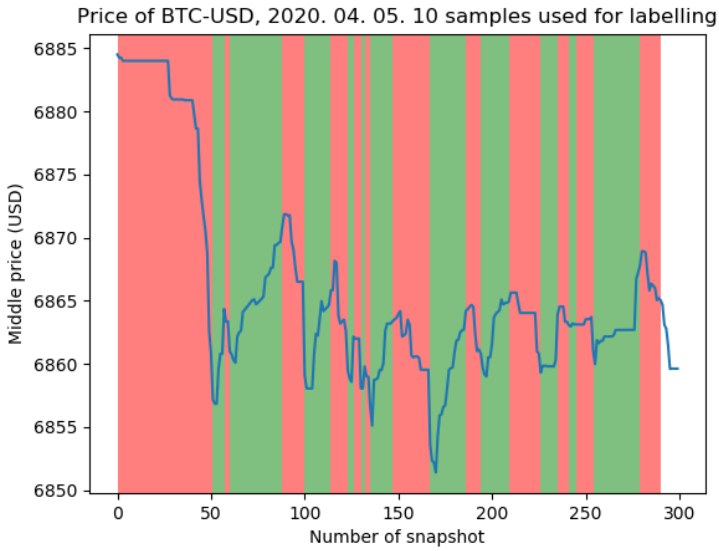


Figure 6.1: Middle-price sequence of a few snapshots from the collected dataset.

Figure 6.1 shows an example of the collected data - for each snapshot the middle-price is calculated, and the background shows the according label - positive or negative. 10 future snapshots were used for the classification of each snapshot.

6.2 Data preparation

The data preparation consists of two steps; the first one is the Order Book Preprocess Engine, which executes specific orders on a locally maintained order book, while the second component is responsible for normalization and batching.

6.2.1 Order Book Preprocess Engine

This step applies the LOB updates onto the snapshots, as it would be applied on the exchange itself, thus, creating a time series of LOB snapshots. As this part of the pre-processing purely relies on the previous state of the order book, and needs to be done sequentially, it cannot be efficiently parallelised and transferred to GPUs.

In this document, only limit orders are considered – market orders are not in scope, as these are executed instantly. For the Order Book Preprocess Engine, the following order typesare defined that can change the state of the LOB:

- **Open**: A limit order has been opened by a client.

- **Done**: A limit order has either been fulfilled or cancelled by the exchange; it must be removed from the book.
- **Match**: Two limit orders have been matched, either completely or partially. If the match is partial, the volume of the corresponding orders will be changed. If the match is complete, both orders will be removed.
- **Change**: The client has chosen to change the properties of the order - the price and/or the volume.

There could be additional types of orders on different exchanges, indeed.

The goal of this first phase is to create snapshots of the exchange, by a fixed time gap between them. Each order in the stored book has the following properties:

- **Price**: The ask or bid price of the order.
- **Volume**: The volume of the asset which the client would like to sell/buy.
- **Id**: A unique identifier of the order.

While applying updates onto the stored orders of the exchange, the procedure also has to take the fact into account, that at any point in time it should be able to produce a snapshot of the Limit Order Book - in which all of the orders should be ordered by their price. This means, looking up each element by an identifier should be relatively quick (which makes applying orders computationally efficient), but sorting the existing orders should also be achievable. The performance drawback of each snapshot generation cannot be completely eliminated, so as smaller time-frames are chosen between each generated snapshot, the computation will take more and more time.

6.2.2 Batching and normalisation

The input of this component is n pieces of snapshots provided by the previous steps. The component can be tore down to smaller pieces - it creates fixed-size or logarithmic intervals, scales the prices of the orders in a snapshot, maps these into the intervals and calculates cumulated sum of the volumes on each side of the LOB. The process is described in details below.

1. Splitting the input data into batches: n consecutive snapshots are concatenated (batched) with a rolling window. The number of snapshots in each

batch, and the index of the considered snapshots (every 1st, 2nd, 3rd, etc) are parameters. These batches can also be interpreted as matrices, which look like the following:

$$\mathrm{X}_1 = \begin{bmatrix} a(t-k)_d & a(t-k)_{d-1} & \dots & a(t-k)_1 & b(t-k)_1 & b(t-k)_2 & \dots & b(t-k)_d \\ a(t-k+1)_d & a(t-k+1)_{d-1} & \dots & a(t-k+1)_1 & b(t-k+1)_1 & b(t-k+1)_2 & \dots & b(t-k+1)_d \\ & & & \dots & & & & \\ a(t)_d & a(t)_{d-1} & \dots & a(t)_1 & b(t)_1 & b(t)_2 & \dots & b(t)_d \end{bmatrix} \tag{6.1}$$

$$\mathrm{X}_2 = \begin{bmatrix} a(t-k+m)_d & \dots & a(t-k+m)_1 & b(t-k+m)_1 & \dots & b(t-k+m)_d \\ a(t-k+m+1)_d & \dots & a(t-k+m+1)_1 & b(t-k+m+1)_1 & \dots & b(t-k+m+1)_d \\ & & & \dots & & \\ a(t+m)_d & \dots & a(t+m)_1 & b(t+m)_1 & b(t+m)_2 & \dots & b(t+m)_d \end{bmatrix} \tag{6.2}$$

$$\mathrm{X}_{l-1} = \begin{bmatrix} a(t-k+l*m)_d & \dots & a(t-k+l*m)_1 & b(t-k+l*m)_1 & \dots & b(t-k+l*m)_d \\ a(t-k+l*m+1)_d & \dots & a(t-k+l*m+1)_1 & b(t-k+l*m+1)_1 & \dots & b(t-k+l*m+1)_d \\ & & & \dots & & \\ a(t+l*m)_d & \dots & a(t+l*m)_1 & b(t+l*m)_1 & \dots & b(t+l*m)_d \end{bmatrix} \tag{6.3}$$

In these matrices, t is the timestamp on the last snapshot of the first batch. d is the order book depth, m shows how many snapshots we skip when generating the next batch, and k is the length of a batch. X_i refers to the actual batch,

2. Normalising each batch independently: the middle price of the last snapshot in the batch is extracted for each, and all the prices in a batch are divided by this value. This also means, for the last snapshot of the batch, the bid prices will always have a value between 0 and 1.0, while the normalised asks will always be larger than 1. However, this limitation does not apply to the previous snapshots of the batch, as the price moves move up or down. This scaling is necessary, because we intend to use the pipeline for multiple assets. However, one asset could be traded in a price range with magnitudes of difference compared to other assets. Applying the proposed normalisation causes prices of any asset to be scaled close to 1. This will make the input for the neural network model quasi standardised.

3. Determining price levels for the 0-th order interpolation. In this step, the discrete values are calculated, which will be used for representing an interval of prices levels in the order book. These will contain the sum of volumes available in each price range at the end of processing.

   The pipeline offers two methods to create these: linear (the intervals has the same size) and logarithmic (the intervals closer to the mid-price are smaller). The logarithmic intervals have a benefit over the linear ones: choosing linear intervals that are nor too big or too small needs optimization, as there could

be larger price jumps - considering volatile assets and periods. These jumps could cause the values to be outside the intervals. With logarithmic intervals, this problem can be decreased, as the intervals on both sides can collect a much greater range of price values. Furthermore, prices around the spread are the most important ones, as discussed above – which are represented more detailed in the logarmithmic case.

One interval can be notated as:

$$[r_{lower}; r_{upper}) \tag{6.4}$$

where *lower* and *upper* are the limits. In this step, multiple of these intervals are created, and stored in a list:

$$[r_{1;lower}; r_{1;upper}), [r_{2;lower}; r_{2;upper}), ...[r_{n;lower}, r_{n;upper}) \tag{6.5}$$

The lower bound of the $k+1$th interval always equals to the upper bound of the kth interval.

The selected intervals are re-used through the application, in the next steps.

4. 0-th order interpolation and summation. For each order in a snapshot the corresponding interval is defined (the price of the order is between the limits of the interval). This is done by iterating over the intervals, and verifying the price of the orders:

$$r_{i;lower} \leq price < r_{i,upper} \tag{6.6}$$

where i is the index of the actual interval. The volumes of orders which should be put into the same interval are summed. At the end of this step, the output contains the sum of volumes for each interval.

5. Iterating through the intervals which contain the volumes and calculating the accumulated volumes, separately for bid and ask side. At the beginning of this step, the summed volume for each interval is known. After the step is done, the value in each interval tells what is the volume that can be bought/sold below/above the lower/upper price level of the interval. When this part of the processing is done, the output for each snapshot could be plotted as in Figure 3.1. This step can also be parallelized for each batch. An example to a completely processed batch can be seen on Figure 6.4.

6. Generating labels for the batches. This step is required for supervised learning algorithms to define a target variable for every batch. In the scope of this

document, the mid-price and VWAP properties of the order book have been chosen to be used as labels, however, it can be arbitrary. Labels are generated according to the snapshots after the batch and scaled by the mid-price (or VWAP) of the last snapshot in the batch. If a regression model is used, then the value itself is the target variable. If the goal is to have a classification model, the target value can be clustered e.g., into 3 labels:

- Upward, downward and sliding movement. When calculating these, an α parameter is considered, which is a percentage value. If the average of the next snapshots is greater than $(1 + \alpha) * midprice$ it is considered upward. If it is less than $(1 + \alpha) * midprice$ it is downward, else the batch is labelled as a sideway movement.

The steps above result in a set of batches, where each batch contains n snapshots and corresponding output label(s). When displaying a batch, the relative price movement over time can be discovered.

6.2.3 Data inconsistency

While checking the data feeds from the exchange, a noticeable number of updates were missing - this could be due to many factors, for example issues with the connection, or the reliability of the websocket feed. To mitigate this, an additional check has been added into the Order Book Preprocess engine - if the application does not receive updates for a longer period of time, the processing is stopped. The algorithm checks the amount of missed update messages, and the processing is restarted from the next point where the data is continous and can be used.

A single missing update can cause the order book to be in a completely inconsistent state - the taken snapshot will not be usable for training networks properly. For example, if the missing update message was a removal, an order gets stuck in the snapshots. This can cause prominent errors in the quality of the data, e.g the price of the largest bid can be higher than the price of the first ask in a later step of the processing. Although the quantity of the data will be less (the processing restarts from the next crawled snapshot, if possible), the quality and usability is going to be much better, so this step is always completed when the pipeline is run.

6.3 Labeling methods

Multiple labelling methods were used while executing learning methods.

6.3.1 Fixed horizon labeling method

This is the simplest used solution for labeling. The algorithm only takes the next n snapshots, calculates the mean value of their mid-price and compares it to the current snapshot's mid price. If it's larger than $1 + alpha * m$, then the current label is positive, if it's less than $1 - alpha * m$, then it's considered negative. If the value lies within $1 - alpha * m$ and $1 + alpha * m$, then it is considered a sliding sample.

This method is simple to implement, however, it has its drawbacks. The algorithm doesn't take the volatility of the data into consideration. Volatilities of the examined assets change in each timeframe, however, the *alpha* parameter can be only chosen once in this method. Using this method for a single day will result in un-balanced number of labels - e.g, for one day, most of the labels would be negative, while for other days, most samples would be sliding, due to the decreased volatility. The method could be improved by adaptively choosing the alpha parameter from the volatility of the current dataset. The next method takes this factor into consideration - using the variance of the dataset.

6.3.2 Triple barrier method

The triple barrier method is slightly more complicated than the fixed horizon labeling. It introduces 3 barriers, which will classify each sample:

- Stop-loss limit: In financial terms, this limit is set for automated trading, so that the holder of a financial asset will not lose too much money, when a bigger dip happens in the price movement of the asset. There are many methods to choose this limit - at first, a constant *alpha* parameter is used. Automated trading algorithms instantly sell the asset when the stop-loss limit is set - even if the asset is sold at a lower price compared to when purchased, the bigger loss will be eliminated.

- Profit-taking exit price: this limit works similar to the stop loss, but in the opposite direction - if the price of the purchased asset reaches the profit-taking exit price, then the algorithm sells the asset, as the desired profit has been reached.

- Sliding time limit: Contrary to the others, this limit is not defined as price, but as time. When the time reaches this limit (for example, 10 seconds after the sample data), and none of the previous mentioned barriers have been reached, the sample is considered as sliding.

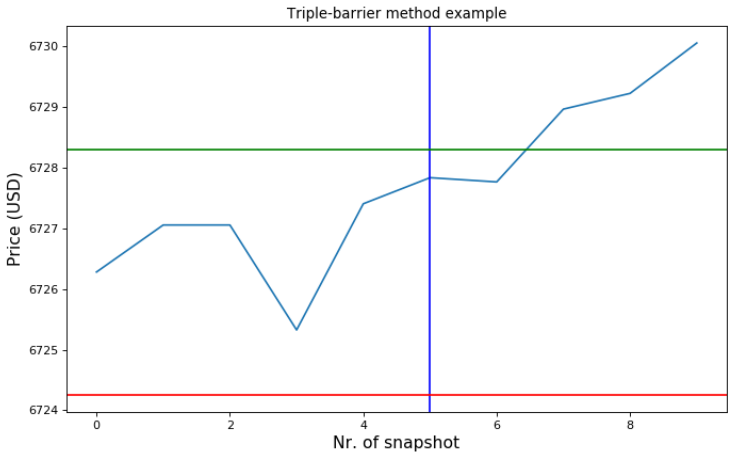
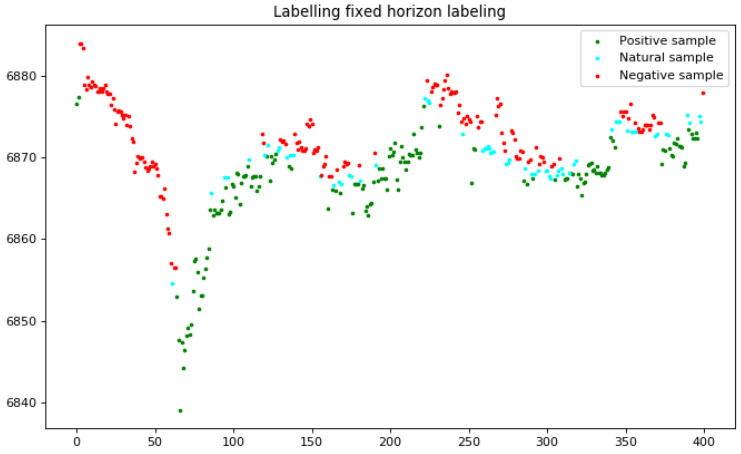
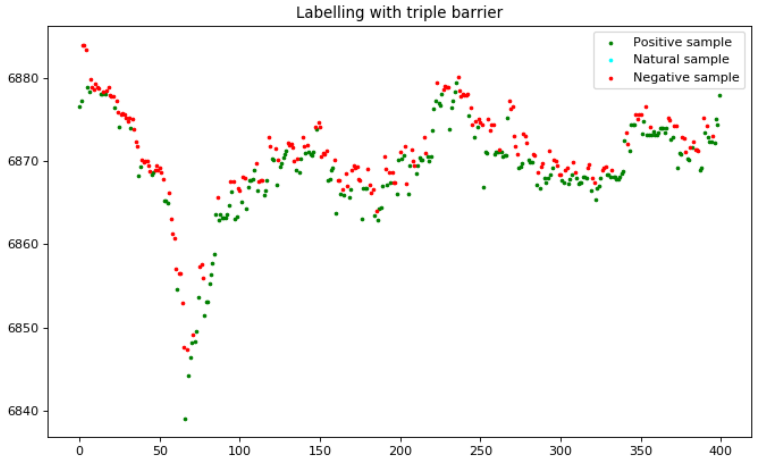


Figure 6.2: An example of the barriers projected on the middle-price of BTC-USD. $alpha = 0.0003$. The green line is the profit-taking limir, the red is the stop-loss limit, while the blue is the sliding time limit.

This method does not average the future middle-prices, but looks at the possibility of losing/winning money, which is closer to real-world applications.



(a)



(b)

Figure 6.3: Comparing the fixed-horizon labeling and triple-barrier method - there are almost no noticable differences for a small dataset.

While the examples in the above figures were using middle-price as the metric of the labeling, VWAP and moving averages can be used as well.

6.4 Modeling

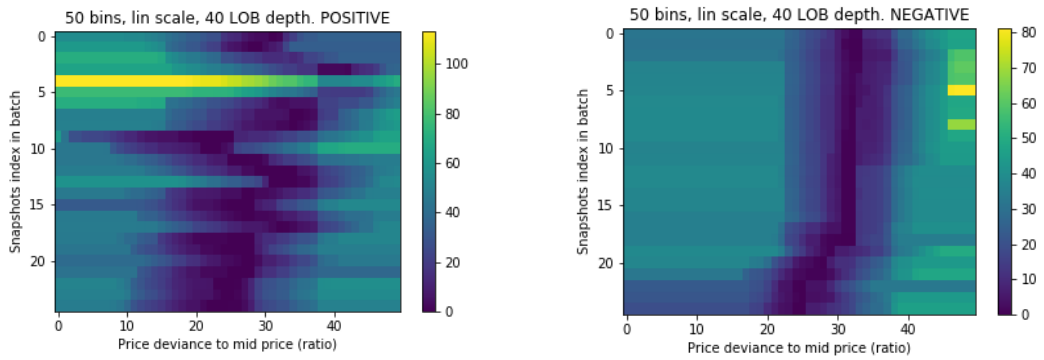
The used models and their architecture, details are presented in this chapter.

6.5 Overfitting on a single batch

As a starting point, to check if the network can 'memorize' small data, a simple validation and sanity check has been done. The point of this was to testify that a simple model can overfit on a single batch.

For the first experiment, 4 samples were randomly chosen, and to make the experiment even more simpler, only 2 labels were used: positive and negative price movement.

2 examples of samples in the batch can be seen on figure 6.4a and 6.4b.



(a) **(b)**

Figure 6.4: Example batches used for single-batch overfitting. While the left image can not be obviously classified by the human eye, the one on the right is easier to comprehend.

The most simple, experimental architecture which was used for overfitting on a single batch had 2 2D convolutional, 2 Max Pooling and 4 fully-connected layers. Although this architecture seems small compared to the task that it should execute, it already has more than 1.2 million parameters to learn while training.

The labels were one-hot encoded, so that the neural network has 2 outputs. For the sake of simplicity, this experiment used the SGD optimizer, setup with a learning rate of 0.0005, and a decay parameter of 0.0001 was chosen.

6.5.1 Increasing the model complexity

To reach a model which is able to predict the price with usable results, an iterative approach is going to be followed. This approach enabled us not to use too much

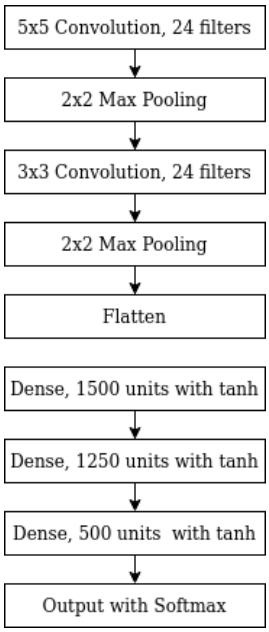


Figure 6.5: The initial, experimental neural network for overfitting on a single batch.

GPU-time for training the models, which were not accurate enough. This was the following:

- Use small LSTM networks that can overfit on single batch or can predict a small dataset. - Increase the size of the dataset and the size of the used data. - When the LSTM is no longer able to predict the data, experiment with other networks, such as CNNs, and try different LSTM and CNN networks combined.

However, the work has been started on developing more complex learning models, they are not included in this document.

Chapter 7

Evaluation

In order to measure the speed enhancements of the proposed methods, experiments with different data sizes and hardware accelerators have been carried out. The evaluation was done with data taken with the data collection tool. The data was collected with Coinbase Pro API. Coinbase exchange provides REST support for polling snapshots. For evaluation, a snapshot has been saved each 5 minutes, while the updates of the book are provided via Websocket streams.

7.1 Hardware and software architecture

The GPU accelerated pipeline extensively uses the Numpy and Numba packages - the CUDA accelerated code was written using Numba's CUDA support. To compare the performance the pipeline was executed on different GPUs (single GPU in each run) for the same amount of raw data. The deep learning models were created with Keras. The data collection, preparation and trainings were run in a containerized environment, using Docker.

The hardware units used in the tests:

- GPU 1: NVidia Titan V, 5120 CUDA cores, 12 GB memory.
- GPU 2: NVidia Titan Xp, 3840 CUDA cores, 12 GB memory.
- GPU 3: NVidia Titan X, 3072 CUDA cores, 12GB DDR5 memory.
- CPU: Intel Core i7-6850K CPU, 3.60GHz

7.1.1 Execution time

In the first phase of the pipeline – the Order Book Preprocess Engine – the updates are sequential, which makes its execution in parallel a difficult problem. Because of that, the measurements have been broken into 2 parts: the snapshot generation and the parallellised normalisation with other steps.

Table 7.1 shows the measurements taken on the Preproces Engine component, while Table 7.2 measures the components that apply the normalisation on the output of the Preprocess Engine. For comparison, the CPU column runs the same algorithms, but using plain Python with libraries such as Numpy. In the 'CPU + LLVM' measurements, we utilise the optimization techniques of the Numba library. This project uses a just-in-time compiler to compile certain steps of the process - the same steps which are accelerated by CUDA when running on a GPU.

Table 7.1: Performance of the Order Book Preprocess Engine component. Data taken from the Coinbase Exchange. The time delay between the snapshots is 1 second.

Pipeline performance		
Batches generated	Time taken [seconds)]	Nr. of processed updates
2088	8.4 seconds	100000
9941	18.4 seconds	1000000
90086	169.58 seconds	10000000
504880	792.01 seconds	57.35 million

Table 7.2: Performance measurements of preparing the data after the Order Book Preprocess Engine. (LOB depth: 100, batch size: 10).

	Pipeline performance [seconds]				
Data used	CPU	CPU + LLVM	GPU 1	GPU 2	GPU 3
2070 batches	129.93	2.08	2	1.7	1.71
9941 batches	641.84	3.9	2.59	2.6	3.2
504880 batches	19404.38	137.237	58.76	61.77	101,86

7.1.2 Inference

To be able to integrate the solution into real-world trading environment, the pipeline should be able to process the stream of order book data and execute the steps even for

a single batch as fast as possible. To simulate the inference phase time measurements were carried out with only a few snapshots. In these experiments 500 updates were used, from which 29 snapshots were created. The first phase - applying these updates and creating the next snapshots - took 7 seconds to generate 29 snapshots (0.15 seconds / snapshot). Normalising with CPU took 2.06 seconds (0.071 seconds / snapshot), while it took around 1.70, 1.95, 1.59 on the Titan V/Xp/X GPUs. The processing took 0.058, 0.067, 0.054 seconds / snapshot, respectively. This means there is is 18% improvement when running on Titan V, and 22% on the Titan X).

7.2 Benchmark algorithms

A widely spread technique for verifying the performance of new prediction algorithms is to compare them with existing ones, using the same data for input. With these benchmark performance metrics, anyone could verify if the model is performing better or worse, or if it outperforms earlier models in certain situations. For this reason, the benchmark checks should be run on as much data as possible, with multiple conditions. This means, for financial models, both low and high volatility market periods should be checked.

7.2.1 Benchmark details

To make the results of the benchmark models comparable, the same data was used for all of them.

For testing purposes, a smaller dataset of 5 days was chosen from the BTC-USD limit order book data of 2020, May. The data processing for the benchmarking methods did not use all components of the pipeline - only the snapshot generation and the labeling was utilized.

7.2.2 Predicting future price with mean value

As the most simple algorithm, the previous n middle-prices were taken into consideration. The mean of these numbers were used to predict the future price. If the mean of the previous n samples was less of the current sample, it was labelled as negative (using the same alpha threshold as in the naive labeling algorithm). If the mean of the previous samples were larger, the sample is considered negative.

This algorithm requires minimal computational power, and it's very naive - it only compares two numbers to come up with the predictions. The expectations for this predictions were also not too high, and there was no need to separate the data to training/validation/test sets.

The tests run did not predict the sliding label at all, and struggled to predict anything - which was our expectation - sophisticated methods like neural networks wouldn't be needed if the next price could be calculated with such a simple mathematical calculation as using the previous mean values.

7.2.3 Linear regression

The second benchmark model which was used is the linear regression. Itt can only grasp linear connections of variables. Limit Order Book modeling probably includes non-linear movement - if looking at any price chart, just a peak can justify this, the prices often move in an exponential manner.

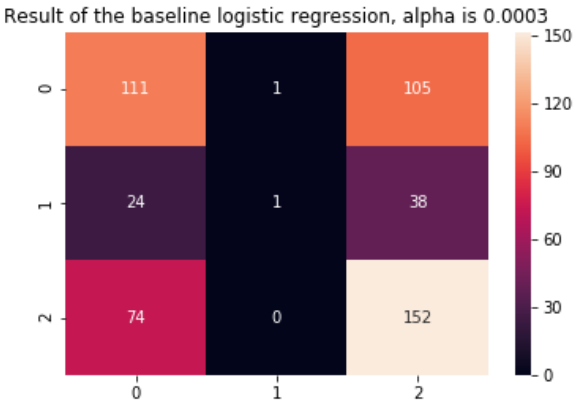


Figure 7.1: Heatmap of the results with linear regression on the benchmark dataset.

The method performed better than using the previous mean values, which was the expectation, however it struggled to predict any sliding labels.

7.2.4 Random forest benchmark

The random forest algorithm is fairly complex compared to the previous benchmark methods, it requires more computational power and it runs for a longer time.

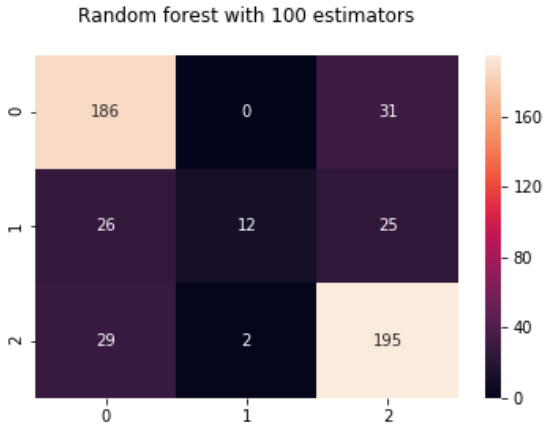


Figure 7.2: Heatmap of the results with random forests on the benchmark dataset.

The random forest algorithm performs surprisingly well on the dataset, however, as the previous model, it struggles to classify sliding samples as well. A simplified version of the task would be to consider only 2 labels - positive and negative price movements - in which the algorithm performs well. The accuracy is also decent, but that does not mean that it would perform well in a real-life situation. It could be due to the small size of the dataset. The data also includes a smaller time period (not years of data) so it may perform poorly in situations when volatilities change compared to the training dataset. The total accuracy of the model was 0.7.

Chapter 8

Summary

Tackling financial problems with learning algorithms can be a hard task, due to the amount and quality of the data, and the behaviour of the markets as well. However, recent technologies - such as massively parallel GPUs - and trends - such as disk space getting cheaper - make executing these algorithms easier.

Usually, preprocessing steps take a considerable ratio of time taken with model training and data processing. The proposed pipeline aims to remove some of this burden for the limit order book, and produce training samples that can be used for anyone developing a learning algorithm. Normalizing order book data with the proposed procedure provides more than 100 times faster execution when compared to using only the regular, CPU algorithms. The performance improvement is also noticeable when running the algorithm outside the GPU - the just-in-time compiler of Numba takes just twice as much time as using CUDA on a decent graphic card.

Future work includes development of algorithms which are capable of predicting the next price movement with higher accuracy than the presented benchmark models. These algorithms will run on the GPU as well. Some parts of the pipeline don't transfer data straight on the GPU - these can also be improved in the future, to deliver faster preprocessing.

Bibliography

[1] Johannes Beck, Roberta Huang, David Lindner, Tian Guo, Zhang Ce, Dirk Helbing, and Nino Antulov-Fantulin. Sensing social media signals for cryptocurrency news. *Companion Proceedings of The 2019 World Wide Web Conference on - WWW '19*, 2019. DOI: 10.1145/3308560.3316706. URL http://dx.doi.org/10.1145/3308560.3316706.

[2] Markus Bibinger, Christopher Neely, and Lars Winkelmann. Estimation of the discontinuous leverage effect: Evidence from the nasdaq order book. *Federal Reserve Bank of St. Louis, Working Papers*, 2017, 04 2017. DOI: 10.20955/wp.2017.012.

[3] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. DOI: 10.1086/260062. URL https://doi.org/10.1086/260062.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. DOI: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[5] Krzysztof Cios. *Deep Neural Networks—A Brief History*, pages 183–200. 01 2018. ISBN 978-3-319-67945-7. DOI: 10.1007/978-3-319-67946-4_7.

[6] Rama Cont, Sasha Stoikov, and Rishi Talreja. A stochastic model for order book dynamics. *Operations Research*, 58:549–563, 06 2010. DOI: 10.2139/ssrn.1273160.

[7] Yann Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. 12 2016.

[8] Luca Di Persio and O. Honchar. Artificial neural networks architectures for stock price prediction: Comparisons and applications. 10:403–413, 01 2016.

[9] Kuzman Ganchev, Michael Kearns, Yuriy Nevmyvaka, and Jennifer Wortman Vaughan. Censored exploration and the dark pool problem, 2012.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. Convolutional sequence to sequence learning. 05 2017.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[12] Rakshit Jha, Mattijs De Paepe, Samuel Holt, James West, and Shaun Ng. Deep learning for digital asset limit order books, 2020.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. DOI: 10.1145/3065386.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[15] Mehrnoosh Mirtaheri, Sami Abu-El-Haija, Fred Morstatter, Greg Ver Steeg, and Aram Galstyan. Identifying and analyzing cryptocurrency manipulations in social media, 2019.

[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[17] Sabuzima Nayak, Ripon Patgiri, and Thoudam Singh. Big computing: Where are we heading? pages 1–10, 04 2020. DOI: 10.4108/eai.13-7-2018.163972.

[18] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37 (8):852–866, Aug 2018. ISSN 0277-6693. DOI: 10.1002/for.2543. URL http://dx.doi.org/10.1002/for.2543.

[19] Angelo Ranaldo. Order aggressiveness in limit order book markets. *Journal of Financial Markets*, 7:53–74, 01 2004. DOI: 10.1016/S1386-4181(02)00069-1.

[20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

[21] Bahman Rostami-Tabar and Juan F. Rendon-Sanchez. Forecasting covid-19 daily cases using phone call data, 2020.

[22] J. Rotyis and G. Vattay. Statistical Analysis of the Stock Index of the Budapest Stock Exchange. Papers cond-mat/9711008, arXiv.org, November 1997. URL https://ideas.repec.org/p/arx/papers/cond-mat-9711008.html.

[23] Jonathan Sadighian. Deep reinforcement learning in cryptocurrency market making. 11 2019.

[24] Fahad Abdullah Saleh. Blockchain without waste: Proof-of-stake. 2020.

[25] Omer Sezer and Murat Ozbayoglu. Financial trading model with stock bar chart image time series with deep convolutional neural networks. 03 2019.

[26] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning, 2018.

[27] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features, 2018.

[28] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

[29] Haoran Wei, Yuanbo Wang, Lidia Mangu, and Keith Decker. Model-based reinforcement learning for predictions and control for limit order books, 2019.

[30] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, Jun 2019. ISSN 1941-0476. DOI: 10.1109/tsp.2019.2907260. URL http://dx.doi.org/10.1109/TSP.2019.2907260.

[31] Zhibiao Zhao. Parametric and nonparametric models and methods in financial econometrics. *arXiv.org, Quantitative Finance Papers*, 1, 01 2008. DOI: 10.1214/08-SS034.