AI Trading Engine

TI 3806 - Bachelor End Project

R.v. Gurp
Y.J. Hu
H.V. Kooijman
A. Somai

Exploring the capabilities of AI in digital asset trading

TU Delft

AI Trading Engine

TI 3806 - Bachelor End Project

by

R.v. Gurp
Y.J. Hu
H.V. Kooijman
A. Somai

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday February 1, 2019 at 14:00.

Student number:	4271742,	(rvangurp)
	4356241,	(yjhu)
	4126890,	(hvkooijman)
	4366220,	(asomai)
Project duration:	November 12, 2018 – February 1, 2019	
Thesis committee:	Prof. Dr. H. Wang,	TU Delft, Bachelor Project Coordinator
	Prof. Dr. E. J. Rellermeyer,	TU Delft, Bachelor Project Coach
	M.L. C. Jacobs,	Blockrise
	J. Lazet,	Blockrise

TU Delft

Preface

This project is the final course for the Bachelor's degree programme of Computer Science at the TU Delft University of Technology. The goal of this project was the development of a digital asset trading engine backed by artificial intelligence. It was developed for the company Blockrise which deals in digital asset investment and trading. The project started on November 2nd, 2018 and finished on February 1st, 2019. This document describes the product and its development process during this time, including the requirements, design choices, implementation details, the product results and future discussions.

We would especially like to thank Prof. Dr. Jan S. Rellermeyer for taking the role of mentor for this project and for his guidance and support during the development process. We would furthermore also like to thank Joshuan Lazet and Cornelis Jacobs of Blockrise for having provided us a great learning opportunity and a chance to explore the limitations of technology. Lastly, we would like to thank the Bachelor End Project coordinator, Prof. Dr. H. Wang.

R.v. Gurp
Y.J. Hu
H.V. Kooijman
A. Somai
Rotterdam, January 2019

Summary

Scientific advances in the field of artificial intelligence, and the ever increasing processing power of computers, have opened up opportunities to use artificial intelligence for new industries and applications. Blockrise foresaw opportunities in using artificial intelligence for digital asset management. Blockrise's founders and the development team formalised the problem and drew up a project proposal, which was accepted by the TU Delft Bachelor End Project coordinators and overseen by Prof. Dr. Jan Rellermeyer in the role of TU Coach.

The development team first reserved roughly two weeks to spend on researching the problem and possible solutions. A research proposal was formalised, in which details of the problem were explained and abstract solutions were given. During the next six weeks, the team started the concrete implementation of the solution. The team created a neural network to use as core functionality in the product. Supporting functionality was simultaneously developed to allow retrieval and processing of necessary data. With enough useful data on hand, the neural network could be trained to make predictions based on an asset's closing price, opening price, the highest price and the lowest price for each next minute, hour and day.

Extensions to the product were made in the form live-data processing, and validation and visualisation of predictions. Trading strategies were included to allow fully automated decision on placing market orders. The final stage of the development period was spent tweaking the neural network parameters in order to minimise prediction error.

Contents

1 Introduction

Ever since the first stock exchanges arose, stock traders have been attempting to predict market trends to turn a profit. By analysing stock price charts and identifying trends, they hoped to predict asset rise or fall. Consistent alpha generation, a term to describe edge over the market or ability to make profit, has remained a hot topic with the potential to earn billions.
Then came along the digital age. With the rapid improvement of computing power and the availability of vast amounts of stock market data gathered from all over the world, stock traders saw new opportunities. More accurate ways for predicting the stock market were devised. The reliability of these predictions often remains questionable. Moreover, it still takes many years of experience and extensive knowledge of the stock market to make even reasonable predictions. This difficulty in predicting the stock prices is all the more true for the recent digital asset (e.g. Bitcoin) markets, which have proven extraordinarily volatile.

The challenges stakeholders face when attempting to make predictions involve human emotions (optimism fosters investment whereas depression begets restraint) and a vast amount of influencing events, including unexpected ones such as, for example, the Volkswagen scandal regarding the manipulation of exhaust gas emission. The general rule is that the more information is available, the more accurate the predictions. However, humans can gather and process only limited amounts of data to make a trading decision. There is some aid to be found from internet trading bots that automatically give advice based on market data calculations, such as the Gekko bot [18]. Their primary benefit, however, is trading speed, as computers can react near instantaneously to changes. They do not predict the specific value of a digital asset in the future, if at all.

However, the recent rise of machine learning and artificial intelligence (also known as AI) has brought new ideas and opportunities. With enough processing power, an AI can analyse enormous amounts of data and ”learn” decades of experience in a matter of minutes or hours. It has the potential to outperform humans and autonomously turn massive profits. It is, therefore, no surprise that substantial investment and asset management companies, the primary benefactors of such an AI, have begun investing billions into developing AI-based trading and investment applications.

These developments have also sparked the interests of Blockrise, a newly-formed company dealing in asset management. The benefits that AI offers would aid them both in general digital asset management and price prediction, potentially increasing their profits. AI-based trading engines are a relatively new concept that, to our knowledge, are not widely deployed as of yet. As such, the novelty

of this technology presents an opportunity to be among the first companies investigating AI-based trading of digital assets. The project, as conducted by the development team, serves as an exploration of the possibilities of this technology in combination with Blockrise's core business.
Chapter two describes a detailed analysis of the problem, whereas chapter three provides the problem description. In chapter four the proposed general solution to the problem is described. Chapter five goes into detail about the concrete implementation of the said solution. In chapter six one can find the results of testing the product's performance. Chapter seven outlines the feedback received from the Software Improvement Group (SIG). Chapter eight discusses the obtained results and provides recommendations for future work. Finally, conclusions about the project are drawn in chapter nine.

2 Problem Analysis

Developing an AI Training Engine requires a clear understanding among the stakeholders about the requirements. Every stakeholder may have their ideas about what the engine should be or should not be capable of, and some of these ideas might clash with those of other stakeholders. After each of the stakeholders' requirements has been clarified, a consensus can be reached on the intended capabilities of the trading engine.
This chapter aims to further define the views of the stakeholders on the problem at hand. By defining the desires of each stakeholder, a clear definition of the problem each stakeholder agrees with can be formulated.

2.1. Defining the Stakeholders

The Business Dictionary describes a stakeholder as: “A person, group or organisation that has interest or concern in an organisation ”[11]. In this case, the interest or concern is in a project. This project has multiple stakeholders, all of whom have an interest or concern in the product that is to be developed, which are the following:

- The Company: Blockrise
  - The product owner. This project is executed on their behalf.
- The University: TU Delft
  - This project is part of the bachelor programme.
- The Developers
  - The students take the role of developers and are responsible for the product development.

2.2. Stakeholder Analysis

As mentioned before, each stakeholder has separate ideas about what should and should not be done. This section further investigates the stakeholders and defines their requirements.

2.2.1. The Company

Blockrise is the product owner for this project. It requires a solution and therefore initiated the project. As the product owner, they have a clear vision of what the result should be.
Blockrise desires a digital asset individual portfolio management system, which provides the following functionalities:

- The ability to set up a diverse and individual portfolio of multiple digital assets.
- The portfolio setup should be secure and cost-efficient, multiple asset exchanges should be monitored, and orders should be placed with the 'best execution rule'.
- Portfolios should be kept in 'cold storage'. The number of assets should be monitored.
- Building an AI Trading engine which manages the digital assets portfolio and takes care of rebalancing the assets.
- The execution of orders on exchanges should be handled optimally, such that it prevents unnecessary transfer fees and arranges internal transfers.
- Everything should be available as an API. There should be an interface for both managers and users where current assets and portfolio can be viewed and where changes can be made.

2.2.2. The University

The TU Delft is also a stakeholder because the project is done as part of the bachelor programme provided by the TU Delft. They have set the requirements for the project as a whole:

- The topics covered and skills learned in the bachelor programme should be bundled and used to do this project.
- The students should be able to work in a real-life development team, decide on development processes in consultation with the client and keep track of all the processes.
- The students should be able to assess the quality of the project and make decisions based on that assessment.
- The results of the project must be explained and presented in a good way.

2.2.3. The Developers

The students are the ones doing this project, which automatically makes them stakeholders. Their goal is to do this project according to the requirements set up by the other stakeholders. The students have important interests in this project as it is the closing course of their bachelor programme. Furthermore, they also have requirements for this project:

- The project should be challenging, and the goal is to apply all the knowledge gained in the previous years.
- There should be a certain degree of freedom regarding design and implementation choices.
- The project should carry some educational value. This project is as close as it gets to a real-world project, so it is important to learn how to cope with everything that comes along.

2.3. Conclusion

The company is the largest stakeholder in this project, as they are the target stakeholder and the product owner. Regardless, the university as stakeholder owns this project as part of their bachelor's programme, so their requirements should be met regardless of those set by other stakeholders. For example, the company desires a complete software application, whereas the university emphasises research. This causes the focus of the project to shift towards building, analysing and testing the AI trading engine rather than building a customer-ready software package. Additionally, the execution time of the project is limited to ten weeks, which includes researching the topic at hand and writing a report. Effectively, this leaves the development team with only six weeks for product implementation. Therefore the scope of the project is narrowed specifically to the trading engine. The students should bear in mind that the system they are building should operate as part of a larger application, and should, therefore, be designed in such a way that it is open for extension.

3
Problem Description

The goal of this project is to build an AI-powered trading engine capable of managing a digital assets portfolio in the most cost-efficient way. The AI trading engine has to deal with many factors that come into play in order to make efficient and profitable choices. This chapter further breaks the problem down into parts and discusses each.

3.1. Definition of Done

A project can be considered done when the set criteria for the project are met. These criteria are agreed upon by the stakeholders and the developers beforehand to weed out any future misunderstandings on what the final product should entail. Such misunderstandings could lead to unnecessary conflicts that damages both parties.
The criteria list was drawn up according to the MoSCoW method[5]. The project criteria are considered met when all the items listed under 'must haves' have been implemented.

3.2. Requirements

The development team and the stakeholders, which as mentioned are the company and the university, together determine the scope of the project and draw up the requirements the trading engine should comply to. Applying the MoSCoW method, the functional requirements are placed into separate categories based on their degree of importance. 'Must haves' is a list of critical features that the product must possess before the project can even be considered a success. 'Should haves' contains features that are expected to be part of the product. Features listed under 'Could haves' might be considered for implementation, but are not necessary. Finally, 'Won't haves' are features that will not be implemented during this project.

3.2.1. Must Have

1. The trading engine must work in a LINUX environment. LINUX offers a range of benefits for developers and companies, such as stability, security, portability, access and support. This gives it an edge over other platforms.

2. The codebase must be written mainly in C++. C++ is a very powerful, efficient and highly portable language with a rich assortment of libraries.

3. The implementation must be modular and scalable for future development. Even though the scope of the project was narrowed down to just the trading

engine, it still remains the company's intention to assemble a complete software application according to their initial requirements.

4. The trading engine must be able to trade between at least one pair of currencies (e.g. BTC and USDT)

3.2.2. Should Have

1. The trading engine should be able to test its performance with historical datasets. This is known as backtesting and it is a crucial part in market risk management[12].
2. The trading engine should be able to make decisions based on at least three financial market indicators.
3. The trading engine should make one decision every hour to buy, hold or sell.
4. There should exist a backup function, in the case that an order is not triggered because of an unexpected drop or rise of the price, to recreate the order with new circumstances.
5. The performance should be visualised in some way.
6. The code itself should be well tested.
7. The engine should be able to trade multiple pairs of currencies.
8. The engine should use multiple strategies to make decisions to buy, hold or sell.

3.2.3. Could Have

1. There could be paper trading possibilities in a live trading environment. With paper trading, the user can trade with pretend money in the real-time market. This familiarises the user with the system and can be used to gain the confidence of the user in the engine.
2. The project could be considered successful if the trading engine can maintain the starting monetary value with a lower limit of -10%.
3. The engine could place orders on real markets.
4. The engine could trade multiple pairs of currencies in parallel.

3.3. Research Question

Based on the requirements named above, a research question can be formulated which complies to the needs of the stakeholders. Research questions help the developers focus on the research itself so that a clear goal is in mind and prevent deviation. The research question formulated for this project is:

> *Can an AI trading engine utilising trading indicators maintain a stable monetary value of a digital asset in a live market environment?*

During this project, the goal is to try to answer the research question. Answering the question may be difficult, but is easier to do when it is broken down into subquestions. Which gives the following:

1. What kind of AI technology is suitable for this problem?
2. What should be the input and output data?
3. How should trading decisions be made?
4. How should performance be measured?

3.4. Conclusion

Different stakeholders have different interests. By discussing with all the stakeholders what the requirements should be, the different interests can be combined and turned into a list of criteria, which the product should satisfy. By defining this list beforehand, a clear understanding exists between all the stakeholders of what can be expected from the final product, and potential dissatisfaction is eliminated.
The criteria list consists of several items, however not every item on it has the same priority or need to be included. By dividing it into several categories, the stakeholders know what they definitely can expect in the final product, and what possibly can be expected.

4 Solution design

As described in the problem analysis, Blockrise wants to have a trading engine implemented, while the university strives for a research project. To be able to reach a concession to their conflicting perspectives, the solution will be a trading engine, which also has the functionality to function in a live market. Subsequently, the research will mainly focus on the predictions of the trading engine, and deciding what the optimal strategy is. Hence, this chapter will give a more elaborate view on our solution for the problem mentioned in the previous chapter. Also, it will justify why the designed solution forms an answer to the research question, and this chapter will also contain the limitations of the design.

4.1. Design specifications

The general structure is to create a system, which uses artificial intelligence. This system trains on historical data, to predict the next candlestick. Following the prediction of the next candle, a strategy utilising a technical analysis trading indicator will be applied and based on that decision, the system will output a ”Buy”, ”Hold”, or ”Sell” signal. Consequently, the system will buy, hold, or sell assets based on the latest signal. In the case, a new candle is formed in the market, the system is trading in, it will update the network with the new value, and repeat the process. Using a visualiser for the results, it is possible to visualise the results in a more comprehensible manner.
In other words, there are five steps, which form an infinite loop and provide a solution to the problem. These steps are illustrated in Figure 4.1.

The different components will be explained independently, and in detail in the following sections.

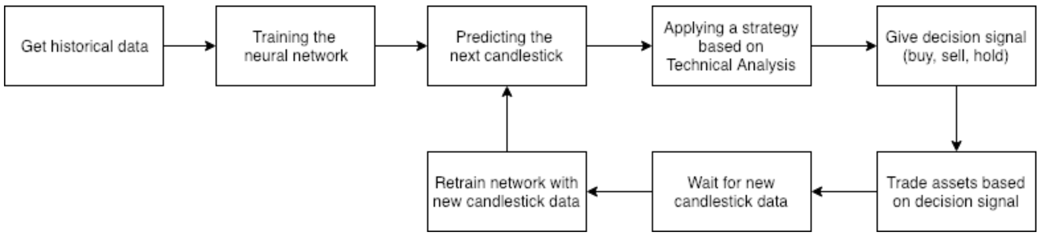

Get historical data
Training the neural network
Predicting the next candlestick
Applying a strategy based on Technical Analysis
Give decision signal (buy, sell, hold)
Trade assets based on decision signal
Wait for new candlestick data
Retrain network with new candlestick data

Figure 4.1: Flow diagram of the solution

4.1.1. Neural network

The neural network is used to predict the data in the future. It is trained on historical data, which consists of candlesticks in an hourly time frame. In other words, the data is a sequence of hourly candlesticks from the beginning until now. Subsequently, the data is a time series, where every point in the future is dependent on points in the history. This type of neural network is especially used on data, which is related to data in the past. It feeds the information back to its hidden layer, and since it can use the previous state to help to make a decision for the next state, it is a proper solution for the type of neural network, that is used in the system.

However, the recurrent neural network could introduce problems during training, such as the exploding and the vanishing gradients problem, as mentioned in the paper by Bengio et al. [9]. Pascanu et al. [23] showed multiple different approaches to solve this problem, which led the choice for the recurrent layer in the neural network to be the Long Short Term Memory (LSTM), which is introduced by Hochreiter and Schmidhuber [19] in their paper. An LSTM unit consists of the input, forget, and output gates, where the forget gate handles the degree of forgetting its memory. Because of this, the gradient does not vanish away and, since the forget gate activation function never becomes greater than zero, the gradient will also not explode.

Moreover, in our system, the input data does not only exist of historical hourly candlesticks data but also a variety of technical indicators. The system starts with the most used indicators by traders in the financial market. These indicators are added to the input data of the neural network, to make more data for the neural network available, so that it could give a better prediction. The core of this idea comes from experienced and professional traders' setup; they do not only use the candlestick data to predict the future price of the asset but also amongst others technical indicators.

4.1.2. Trading strategies

After a prediction, technical indicators are used to determine a trading strategy. Consequently, the trading strategy is used to determine a buy, hold, or sell signal. These signals are used to determine whether you should, or should not trade the asset.

The system will have multiple trading strategies, where each trading strategy is based on technical analysis. The optimal values for the trading strategies differ per trading setup. Thus, the system implements standard initial parameters, and have to be optimised afterwards. The implemented indicators are chosen from the most widely-used indicators in the technical market. Afterwards, by combining those indicators and using multiple standard initial values of those indicators, the trading strategies are implemented.

Subsequently, 'backtesting' allows us to test the trading strategy and its parameters on historical data. By using the last part of the data as 'futuristic' data, it enables the system to simulate the strategy in a real environment. To illustrate this, the system could use the data up to last week as training purposes, whereas it would behave as it went back a week in time, and use the last week of data to simulate the strategy and get a result. In a live trading environment, this is called 'paper trading', where you simulate trades, to test any strategies.

4.1.3. Trading assets

In the case the system receives a signal, it should execute a trade on an exchange accordingly. These trades should be done according to the value and contents of your portfolio. Limits should be set to manage the portfolio to prevent the system from spending all of the liquidity to buy the assets. Likewise, it also should not sell all of the assets in case the system receives a sell signal. These limits should be adjustable to the risk profile of the person using the system, or in short, the investor.

Like the trading strategies, setting the limits could also be tested using back-testing in a simulated environment to optimise the values for the user's strategy and portfolio. In the same manner, paper trading could be used in a live trading environment to validate the limits.

4.2. Design limitations

The project's duration is ten weeks, of which approximately seven are reserved for coding the project. Therefore, the limitations are primarily the result of a deficiency of time. Also, the solution primarily lays its focus on the functionality, whereas optimising the hyperparameters and parameters of the trading strategies, is seen as a secondary objective. That does not mean, that the intention is not to optimise any parameters. However, in the case all the parameters have to be optimised, the search space grows exponentially for every parameter, that has to be optimised. That is why, instead of finding the optimal values for each parameter, the goal for optimising these parameters, is finding at least sub-optimal values. Subsequently, a value is defined at least sub-optimal, when it outperforms other neighbouring values.

Also, optimising some parameters is more important other parameters [17]. To illustrate this, in the system, optimising the parameters for the neural network, is more important than optimising the value for trading strategies, or assets. The reason for this is, that in case the parameters for the neural network are not optimised correctly, then it will give bad predictions. Consequently, it would not matter if the values for the trading strategy or assets would be optimised since it would take wrong decisions, based on the bad predictions of the next price. Therefore, if there is time for optimising parameters, the emphasis will first lay on optimising the parameters of the neural network.

4.3. Design justification

As already mentioned in the previous section, several limitations of the way the solution was designed, have to be taken into account. The following aspects will show whether the solution succeeds in - at least partly - providing an answer to our research question.

As aforementioned, the designed solution consists of three components: the neural network, trading strategies, and trading assets. Assuming that every component has sub-optimal parameters, and thus give proper results, it would mean that it could predict the price for the next hour without a too high error percentage. This prediction would be converted to a decision signal, which will be converted into a trade. These trades will occur one hour before the next candle is formed, and thus our system will have an advantage over normal traders. Using the trading strategies and portfolio management, it should be able to increase the total value of the portfolio. Even if the price of the asset, which the system is trading against, is suddenly decreasing significantly, the system should still be able to do trades, that at least minimise the losses of the unforeseen price decrease.

4.4. Conclusion

In summary, in this chapter, the different components of the system have been discussed, which are the neural network, the trading strategies, and trading the assets. Subsequently, the limitations of each component and the system, in general, have identified as mainly optimising the parameters, and also which parameters are more important than other parameters. Lastly, the designed solution offers an answer to the research question, because it should be able to do trades, which at least minimise the losses, by getting proper predictions of the neural network.

5 Implementation

This chapter will give a more elaborate view on the implementation of the designed solution, mentioned in the previous chapter. The system is also shown in Appendix B.

5.1. Implementation plan

The implementation assesses whether the AI trading engine can keep a stable monetary value when trading with digital assets. Also, the implementation shows a method of how the parameters are converged to near-optimal ones. The code is located in a repository on Bitbucket, a platform that facilitates source code repository management. There it is maintained and updated using Git version control. During the process, different code evolution techniques are applied, such as the agile and scrum methods, along with the SMART criteria and the method of rapid prototyping. The development team held standard daily meetings for software development and general cooperation, along with longer weekly meetings to discuss the state of the product and find solutions to complex problems. These methods prevented the development team from being stuck on a single problem for too long and helped them stay up-to-date with each others' code in a timely manner, ultimately leading to the results discussed in the following subsections.

5.1.1. Environment

As mentioned in the 'must have' requirements 3.2.1, the trading engine must work in a LINUX environment and it must be written in C++. Hence the development team chose C++ as the main programming language and a programming environment which is well suited for this language, namely CLion. This choice of language meant that libraries would have to be found for extended C++ functionality as well. As the system is developed to run in a LINUX environment, there are no guarantees that it operates in other environments such as Windows. One possible reason for this drawback is that certain libraries either do not work properly on various operating systems or are simply not available for other operating systems.

5.1.2. Data-visualizer

The data-visualizer is a separate web interface that is written in HTML, CSS and Javascript. It is used to visualise the data that is consistently fetched online by the data fetching tool. The data-visualizer continuously receives market data at time t, the prediction for that data at time $t + 1$ and the simulated trade made based on that prediction. This information is then represented in graphs of the actual data, the predicted data and the difference between them. Hovering over the graph with the cursor reveals the value of the portfolio and the change in value compared to the previous timestep. Figure C.1 shows some examples of the data-visualizer's output.

5.1.3. Libraries

As mentioned before, the only libraries that could be used in the trading engine are C++ libraries. When there was a need for a certain functionality, the development team had to either implement that functionality using a library or to re-invent the wheel. These decisions have lead to the implementation of the following libraries:

Armadillo

Armadillo is a fast and open source library for linear algebra computations, written by Sanderson [24]. It provides a high-level application interface, similar to MATLAB. Miscellaneous matrix factorisations are implemented through other Basic Linear Algebra Subprograms (BLAS) libraries. These libraries do the actual computation, and various BLAS libraries can be chosen to work with Armadillo. This way, it also enables the trading engine to use the Graphics Processing Unit (GPU) to do the linear algebra calculations, to enhance the performance to an even further extent. An example of this GPU acceleration is when the program uses the NVBlas [21] library as the BLAS library of the Armadillo library. The reason for using this library, is that it was required by the used machine learning library, also it provided a fast and simple solution to do linear algebra computations.

MLPack

MLPack is a fast and flexible machine learning library, written by Curtin et al. [13]. This library allowed the development team to use the neural network implementation and the LSTM layer. Also, the library allowed for different configurations, such as the use of a Fast LSTM layer, which is described in detail in the next subsection. The implementations of the linear algebra calculations in this library are done using Armadillo. As Armadillo is heavily optimised, especially for matrix calculations, the MLPack code is capable of rapid calculations as well. MLPack was chosen as the framework for the AI implementation, because it provides an efficient open-source solution for machine learning functionality that is completely written in C++.

TA-Lib

TA-Lib is a library which provides the methods for calculating the corresponding values of various technical indicators [15]. The library is written in C and does not use any other underlying libraries. It allows for easy implementation and testing of a wide range of indicators and spared the development team the burden of having to implement all the indicators from scratch. Furthermore, it provides functionality for testing various different indicators simultaneously.

Binacpp

Binacpp is the application programming interface (API) wrapper for the Binance exchange, written in C/C++ [10]. Using this API wrapper, it allows the system to fetch the data of a candlestick every hour and act accordingly, by predicting the price and trading with the generated decision signal. Furthermore, this API offers the ability to connect with the exchange directly, so it is also possible to connect it with the trading engine in a later stage so that it can fetch the live portfolio data, and do trades on the exchange. The Binance exchange is the only exchange, that provides a stable API, that functions without problems. Additionally it has been documented fairly well in comparison to other similar libraries, which provide either very few or outdated documentation (or both) or simply none whatsoever.

JsonCPP

JsonCPP is a library that allows parsing from a JavaScript Object Notation (JSON) string to a C++ object [22]. It also can be used to convert a C++ object to a JSON string. Since the live data of the Binacpp API comes in JSON string format, it has to be converted to a C++ object before it can be used in the trading engine. Other than converting the live-fetched hourly candlestick data to a C++ object, it is also used to convert a C++ object to a JSON string, which is sent by the web socket server to the data visualiser. Using this library provides a solution to connect the web layers to C++ code and the other way around.

Websocketpp

Websocketpp is a C++ web socket client and server library. It enables the system to open a web socket server to send a message with the predicted data to the data visualiser, so that it can visualise the data, that it receives. It differs from the web socket library, used by Binacpp, in the programming language it uses, and also in the number of options, the libraries provide. Whereas the web socket library of Binacpp is written in C, has limited options and community support, Websocketpp is written in C++, has more options, and also has more support of the community. Websocketpp is used in the code, because it offers the option to implement a simple web socket server, and is documented well.

5.1.4. Packages

All the source code besides the main function has been split up into several packages. The main function which resides in the root folder makes calls to those packages. The splitting of source code into packages satisfies the third condition of the 'must have' requirements mentioned in 3.2.1. Elaborate package names and generally keeping package size small increases maintainability and scalability, as grouping related content allows for files to be located more easily.

Data

The data package contains the code to construct the input dataset of the neural network (illustrated in Figure B.6). In other words, it contains all the functions that allow for transforming data from the parsed input file to the standardised neural network input dataset, as well as the other way around. The parsed data from the input file is inserted into an Armadillo matrix object. Then the technical indicators are inserted, the data is transformed to a stationary dataset and finally the data is normalised. Hereafter, each step will be elaborated upon in a more detailed fashion.

Several technical indicators implemented, these are the following:

- **SMA - Simple Moving Average**
  The average price computed over a specific number of periods.
  $SMA(x,N)$ for closing price x_i and timeperiod N:

$$SMA(x_i,N) = \begin{cases} \frac{\sum_{j=0}^{j=N-1} x_{i-j}}{N}, & \text{if } i-j>0 \\ 0, & \text{otherwise} \end{cases}$$

- **EMA - Exponential Moving Average**
  The average price computed over a specific number of periods, with more weight applied to recent prices.
  $EMA(x,N)$ for closing price x_i and timeperiod N:

$$EMA(x_i,N) = (x_i - EMA(x_{i-1})) * Multiplier + EMA(x_{i-1})$$
$$EMA(x_0,N) = SMA(x_0,N)$$

$$Multiplier = \frac{2}{N+1}$$

- **MACD - Moving Average Convergence / Divergence**
  First the difference between two EMA's, a lower EMA called the slow moving average and a higher EMA called the fast moving average, is calculated, then an EMA of the difference itself is calculated, called the signal. The difference and the signal line are then compared.
  $MACD(x, slow, fast, signal)$ for closing price x_i and timeperiods $slow, fast$ and $signal$:

$$MACD(x, slow, fast, signal) = (EMA(x_i, slow) - EMA(x_i, fast))$$
$$-EMA(EMA(x_i, slow) - EMA(x_i, fast), signal)$$

- **RSI - Relative Strength Index**
  A momentum indicator that checks the impact of recent price changes.
  $RSI(x, N)$ for closing price x_i and timeperiod N:

$$RSI(x_i, N) = 100 - \frac{100}{1 + RS(x_i, N)}$$
$$AverageGain(x_0, N) = \text{Sum of Gains past N periods}$$
$$AverageLoss(x_0, N) = \text{Sum of Losses past N periods}$$
$$AverageGain(x_i, N) = \frac{(AverageGain(x_{i-1} * N - 1) + Gain(x_i)}{N}$$
$$AverageLoss(x_i, N) = \frac{(AverageLoss(x_{i-1} * N - 1) + Loss(x_i)}{N}$$
$$RS(x_i, N) = \frac{AverageGain(x_i, N)}{AverageLoss(x_i, N)}$$

- **StochRSI - Stochastic Relative Strength Index**
  An oscillator that measures RSI values over a given time period and checks whether the RSI value is overbought or oversold.
  $StochRSI(x, N)$ for closing price x_i and timeperiod N:

$$StochRSI(x_i, N) = (RSI(x_i, N) - \frac{MIN(RSI(X, N))}{MAX(RSI(X, N)) - MIN(RSI(X, N))}$$

  where X contains all the closing prices up to x_i

- **ROC - Rate of Change**
  An indicator that measures the change in price between the current price and the price a specific number of periods ago.
  $ROC(x, N)$ for closing price x_i and timeperiod N:

$$ROC(x_i, N) = \frac{x_i - x_{i-N}}{x_{i-N}} * 100$$

- **CCI - Commodity Channel Index**
  Measures the current price level relative to an average price level over a specific period of time.
  $CCI(x, N, c)$ for average price x_i, timeperiod N and a constant c:

$$CCI(x, N, X) = \frac{x_i - SMA(x_i, N)}{(c * MeanDeviation)}$$
$$AveragePrice(i) = \frac{high(i) + low(i) + close(i)}{3}$$

Furthermore, the dataset is converted to a stationary dataset, by taking the difference with the next data point. The mathematical construction of the new stationary matrix, is given as follows: "Given a matrix x, with dimensions $m \times n$, for the new matrix y holds:

$$y_{i,j} = x_{i,j} - x_{i,j+1}, 1 \leq i \leq m, \& 1 \leq j \leq n - 1". \tag{5.1}$$

Thus, the dimensions of the new matrix y, expressed in the dimensions of the old matrix x, are $m \times n - 1$.

Moreover, row-based normalisation is applied to the stationary dataset, to convert the values to values between -1 and 1. The new row-normalised matrix is mathematically defined as follows: "Given a matrix x, with dimensions $m \times n$, for the new matrix y holds:

$$\begin{aligned} max_i \in y_i \rightarrow max_i \geq y_{i,j}, 1 \leq i \leq m, \& 1 \leq j \leq n, \\ min_i \in y_i \rightarrow min_i \leq y_{i,j}, 1 \leq i \leq m, \& 1 \leq j \leq n, \\ y_{i,j} = 2 * (\frac{x_{i,j} - min_i}{max_i - min_i}) - 1, 1 \leq i \leq m, \& 1 \leq j \leq n". \end{aligned} \tag{5.2}$$

The dimensions of the new matrix have not changed, and therefore, are the same as the dimensions of the old matrix.

Lastly, since MLPack requires the input dataset to be a rank three tensor object of the Armadillo library, the matrix is transformed from a $m \times n$ matrix, where m is the total amount of rows, and n is the total amount of columns, to a $m \times 1 \times z, z = n$ cube, where z is the number of slices. An extra column could be added, to add another sample to the data, such as the data of a similar digital asset.

Network

Whereas the previous package contains the code to construct the input dataset for the neural network, this package contains the code related to the neural network and is illustrated in Figure B.3. It contains the wrapper around the library of MLPack, the integration with the data package, the Time Series Cross Validation (TSCV), and the Time Series K-fold Cross Validation (TSKCV). Hereafter, these different components will be described independently.

The wrapper around the library of MLPack, contains the implementation of the actual neural network. The layers of the neural network are arranged as:

1. *Identity layer*
   The identity layer is a dummy layer, required by MLPack. It maps the input to itself, so nothing changes to the dataset.

2. *Linear layer*
   The linear layer connects the amount of nodes of the input layer to the amount of nodes of the hidden layer. It is a fully connected layer, of which every edge has an individual weight.

3. *Fast LSTM layer*
   The FastLSTM layer is a special faster version of the LSTM layer, which combines the calculation of the input, forget, output gates and hidden state in a single step. Figure 5.1 illustrates a single LSTM memory cell. The composite functions of the standard LSTM used, are based on the paper

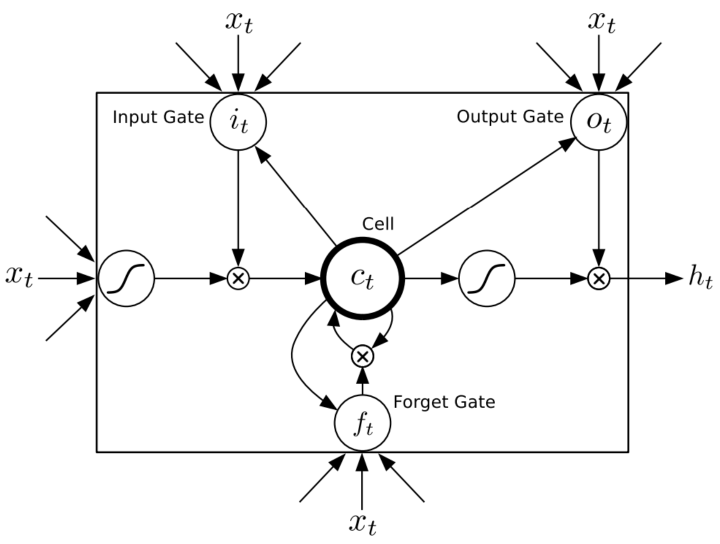


Figure 5.1: Figure of LSTM cell, from [16]

from Graves et al. [16], and are defined as follows:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \\ h_t &= o_t \tanh(c_t). \end{aligned} \tag{5.3}$$

However, since the network does not contain a conventional LSTM layer, but a FastLSTM layer implementation, the composite functions change to [14]:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ c_t &= f_t c_{t-1} + \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ h_t &= \tanh(c_t). \end{aligned} \tag{5.4}$$

4. *Output layer with Mean Squared Error (MSE) performance function*
   The output layer evaluates the neural network, in case it is used for training purposes. If the neural network is used to predict a values, the value that is returned reflects the output of the specified output layer. In this case, the MSE function is used to evaluate the network. It does that as follows:

$$MSE = \frac{1}{m*n}\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij} - \hat{x_{ij}})^2 \tag{5.5}$$

   where x_{ij} is the actual value, $\hat{x_{ij}}$ is the predicted value in row i and column j, m is the total amount of rows, and n is the total amount of columns.

The network also uses the Adam optimiser, introduced in the paper of Kingma and Ba [20], to speed up the training process. The Adam optimiser is computationally efficient, requires little memory, is invariant to diagonal rescaling of the gradients, and is well suitable for large quantities of data, or problems with many parameters. The resulting formula for updating parameters is:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ a_t &= \frac{a * \sqrt{1-\beta_2^t}}{1-\beta_1^t}, \\ \theta_{t+1} = \theta_t &- \frac{a_t * m_t}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \end{aligned} \tag{5.6}$$

where α is the stepsize, m_t is the mean at time t, and v_t is the uncentered variance of the gradients at time t.

The second component is the integration with the data package. This component combines the neural network by preparing the data and post-processing the prediction to get the real value again. As aforementioned, when the data is prepared, it is also transformed to a stationary dataset and normalised, to fit into the range of -1 and 1. To undo this transformation, the data is transformed by taking the inverse from the functions mentioned above and applied to the data.

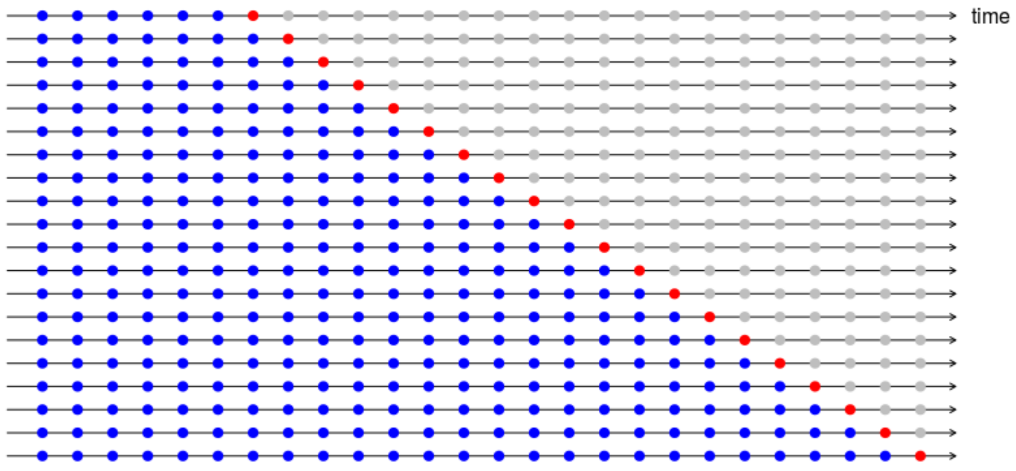


Figure 5.2: An illustration of time series cross-validation.

The third component is the Time Series Cross-Validation (TSCV), and this is also illustrated in Figure 5.2. This component allows the system to run a test, where it takes an x per cent of the dataset to train and takes the rest of the dataset to validate the results. The x is adjustable The difference with normal cross-validation is that when the network is trained until a time t, and the prediction is given for $t+1$, you cannot give the prediction for $t+2$ afterwards. The network first has to be retrained until $t+1$, before it can give a prediction for $t+2$. The error is given using two different measurements. The first measurement is the error percentage, where it is the percentage difference between the predicted data points and the actual data points. The second measurement is the Root Mean Squared Error (RMSE), this is calculated as follow:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij} - \hat{x_{ij}})^2}{m * n}} \tag{5.7}$$

where x_{ij} is the actual value, $\hat{x_{ij}}$ is the predicted value in row i and column j, m is the total amount of rows, and n is the total amount of columns.

The last component is the Time Series K-fold Cross-Validation (TSKCV). In contrary to the TSCV, the TSKCV is based on k-fold cross-validation and made suitable for time series data, or dependent data. The data set is divided into k folds. Afterwards, like in the TSCV, the data is split up into a predefined ratio for training and validation purposes. Consequently, every point is predicted again and validated against the actual value. The error is calculated using the RMSE. By dividing the dataset into k folds, it strives to decrease the possibility of a bias influencing the result of the validation because it prevents the possibility of a bias existing in a specific part of the dataset, which is then used for validation.

Trading

The trading package contains all the code that concern trading and is illustrated in Figure B.4. In other words, it handles the process from the prediction to the decision signal. To go into more detail, it contains the portfolio model, the trading strategies, the engine to combine them both into a trading engine, and the backtesting functionality to test the trading strategy.

The portfolio model contains the value of the portfolio in BTC and USD value, which represents the portfolio that is used on the exchange.

Furthermore, as aforementioned, the trading strategies are based on TA indicators.

- **Weighted Rate of Change**
  The Rate of Change strategy considers the change of the predicted value relative to the last actual value. To the four outputs, close, low, open and high, weights are assigned which add up to one. For each of the output, the percentage based difference with the previous value is calculated. Then every difference is weighted and added up. When this sum is below, in between, or above a lower and upper threshold, a decision is made whether to sell, buy, or hold.
  The weights and thresholds are modifiable, so the user can check themselves which parameter suits them best. Besides, the Rate of Change checks the difference with the previous value, but the possibility exists to compare it to n days before.
  The weights for open and close values are assigned to 0.3, and 0.2 for low and high. The thresholds are set to -0.01 and 0.01, so an average rise or decline of 1% triggers a signal.

- **Relative Strength Index** The RSI strategy makes use of the RSI indicator described in 5.1.4. It checks for upward and downward trends. When an upward or downward trend is persistent for a long period of time, an asset will be marked as overbought or oversold. When the RSI indicates that an asset is overbought, a decision signal is sent to sell, and when an asset is oversold, a decision signal is sent to buy, otherwise, a hold signal will be emitted. The RSI will output a number between 0 and 100, and the thresholds are set to 30 and 70, which indicates when to mark an asset as oversold or overbought.

- **Double Exponential Moving Averages** The DEMA strategy utilises two exponential moving averages, described in 5.1.4, to determine which decision signal should be sent. It uses a slow EMA, set to 12, which moves closer to the real market, but includes noise, and a fast EMA, set to 26, which lags behind the market, but is more resistant to noise. When the slow EMA falls below the fast EMA, a decision to sell is sent, and a decision to buy is sent for the other way around. Otherwise, a hold signal is sent.

- **Moving Average Convergence/Divergence** The MACD strategy is based on the MACD indicator described in 5.1.4. The MACD line, calculated by taking the difference between two moving averages, EMA-12 and EMA-26, of different lengths, as compared to the moving average, EMA-9, of the MACD line itself, and is called the signal line. Just like the DEMA, a decision is made whenever these two lines cross each other.

Also, the package contains the trading engine, which implements the trading strategies, and decides how much should be traded. It takes two predefined values into account to decide if and how much should be traded. The first value represents the minimum amount that should be traded, and the second value represents the maximum percentage of the total portfolio value the system can trade per trade. According to these values, the trades are calculated, executed, and the portfolio values are updated accordingly.

Lastly, the backtesting component is used to test the trading strategies. A predefined value is used to split up the dataset into a training data set and validation dataset. Each value in the validation dataset is used to predict the next value, execute a trading strategy, and using the decision signal to output the trade. Doing this until the end of the validation dataset is reached, results in the final value for the portfolio. The final value represents how the backtesting went, for example, if the portfolio value is higher using a specific strategy, than other strategies, that means that that strategy is performing the best. Also, if the portfolio value is higher than the initial value of the portfolio, then it made a profit. Likewise, if the portfolio value ends up being lower than the initial value, then you made a loss.

Utility
The utility package contains the utility functions, which are the general utility functions and is illustrated in Figure B.5. It contains components, such as the utility functions for the data, the logger, and the parser. The utility functions for the data, contains, amongst others, to convert JSON objects to Armadillo vectors, and the other way around. In other words, it provides functions to connect the web service with the neural network, and the other way around as well.

Subsequently, the logger is used to write results to data, so that it does not solely exist in the intermediate memory. This class is not directly implemented in the code, but it is mainly used to write the results of a testing process to a file so that the results are persisted. It could be used to test different strategies or to optimise parameters, by running the code several times and writing the results of each run to a separate file to compare it. This logger is also used to obtain the results in the next chapter.

Web - service
The web-service package contains the code to receive and send data to other clients and is illustrated in Figure B.2. Currently, the data is solely fetched from the API of Binance, but it can be extended to support data fetching from (multiple) other exchanges their API. The reason that solely the API of Binance is implemented is that the API of Binance is the only API, that is documented decently. Other APIs do not exist, do not function without unexpected errors, or have outdated or no documentation. This API also allows to send a trade to their platform and fetch the latest portfolio of the user.

Furthermore, the web-service package also contains the implementation to host a web socket server, to which it sends the actual values of time $t-1$, predicted values of time t, and the result of a simulated trade, based on the prediction of time t. This data can be visualised using the data-visualizer. The data-visualizer shows the graph of the actual and predicted values, it also shows the difference of the two values, and the values of the portfolio on that time, if it was sent together with it.

5.2. Issues encountered

As Murphy's law dictates: "Anything that can go wrong will go wrong", during the project, there were also a significant amount of issues. The origin of these issues differed considerably.

The first encountered category of issues were the issues of installing and using the dependencies. These dependencies were sometimes created for specific versions of operating systems and were not directly supported for other operating systems. Also, dependencies were sometimes pre-built for specific operating systems, and the pre-built libraries were used in the code. This issue resulted in changing the source code of the library, to make it work for other operating systems.

Furthermore, the DAS-5 cluster node, that was provided for testing purposes was not able to install some dependencies, because of the way some dependencies were written. Some of the dependencies did not support the C++ version and compiler of the supercomputer, which were older, than the supported versions of the libraries. This problem resulted in not being able to use the supercomputer, without installing another compiler, to solely install those specific dependencies.

Also, some libraries were not (well) documented or were not widely used, so that there could not be found much about them. It resulted in finding out what specific functions in the libraries did, by inspecting the source code of the library, or by trial and error. An example of this was that due to this vagueness, the implementation of the RNN was incorrect, and this resulted in weird results and errors.

Finally, the different operating systems and different C++ versions resulted in errors in some operating systems, but not on other. For example, the shared pointers, introduced in the C++ 11 version, worked on the macOS operating system, but on the LINUX operating system, it worked occasionally. A reason for this, could be that the macOS operating system uses the LLVM compiler by default, while the LINUX operating system utilises the GCC compiler by default. Other times, it returned a segmentation fault. Consequently, it resulted in the removal of much new C++ functionality.

5.3. Conclusion

In summary, this chapter described the implementation of the system. In detail, the system is built for the LINUX environment and mainly written in C++. Furthermore, it consists of a recurrent neural network with a FastLSTM layer. Also, it adds several indicators to the input data, to be able to predict more accurately. With the predictions of the neural network, it decides to buy, sell, or hold the digital asset. Based on that decision, it gives a corresponding decision signal.

Consequently, the decision signal is used together with predefined limits for a trade, to decide if and how much should be traded. When the data of a new hourly candle is fetched, it retrains the network and makes a new prediction. The data of the simulation can be sent using a web socket server to the data-visualizer, which visualises the results.

6
Results

This chapter outlines the results obtained by the research. The performance of the network is measured using cross-validation and the Root Mean Squared Error function. Several configurations of network parameters are tested to determine where optimal values may be found. Subsequently, the implemented trading strategies, utilising the RNN to predict future asset value, are validated on the historical dataset.

6.1. Cross-Validation

Cross-validation is a method to assess the quality and performance of the model used to predict the values. In this particular case, because each value relies on the previous value(s), there is a need for a time-series cross-validation, which uses the results prior to the latest value. The model is trained on a part of the dataset, and validated on the remaining part, as illustrated in Figure 5.2. On each iteration the model is trained with a new element from the validation set, and the next value is predicted. The error is calculated and stored, and the model is retrained with the next value. When all the errors are calculated, an error function can be used to measure the performance.

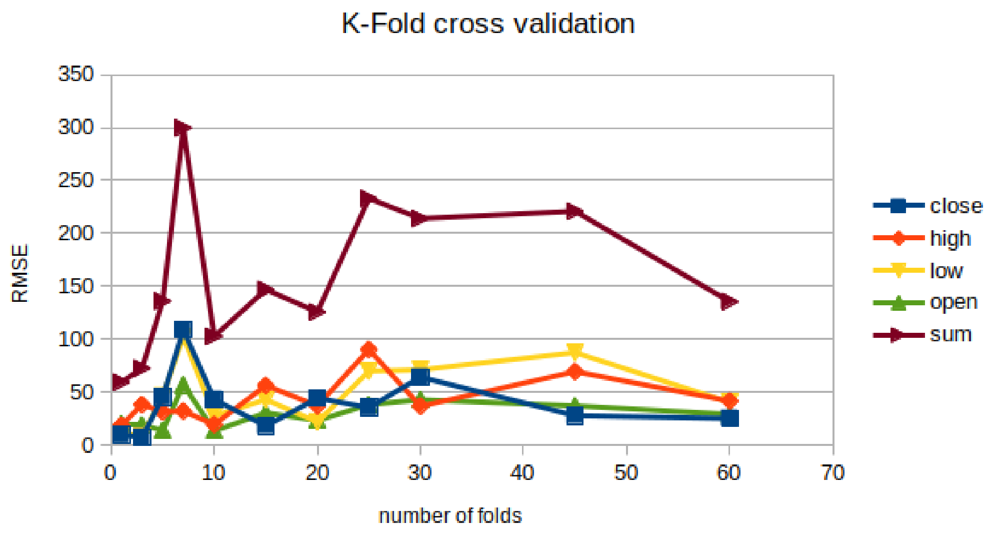


Figure 6.1: Performance of the network, measured using k-fold cross-validation.

6.1.1. Time Series k-fold Cross-Validation

Performance of the network is measured using k-fold cross-validation and the Root Mean Squared error function. With an increasing number of folds the network is consistently trained on smaller portions of the dataset before making predictions, but may generalise better as not only a recent partition of time series is used for validation. Presented in Figure 6.1 are the averaged RMSEs across all folds in a given run.

6.2. Network Parameter Optimisation

Performance is highly dependent on several parameters that fine-tune different aspects of the network. These parameters relate to the topology of the network itself, as well as mathematical functions used to increase the speed of learning and overcome local optima. The results presented in this section are obtained by modifying a single parameter across multiple executions of the program, to examine its impact in isolation. It is likely that, given a different set of default parameters, different results are obtained.

6.2.1. Initialisation

The initial values assigned to the network's weights may significantly impact the speed at which it converges and affects the accuracy of obtained predictions [25].

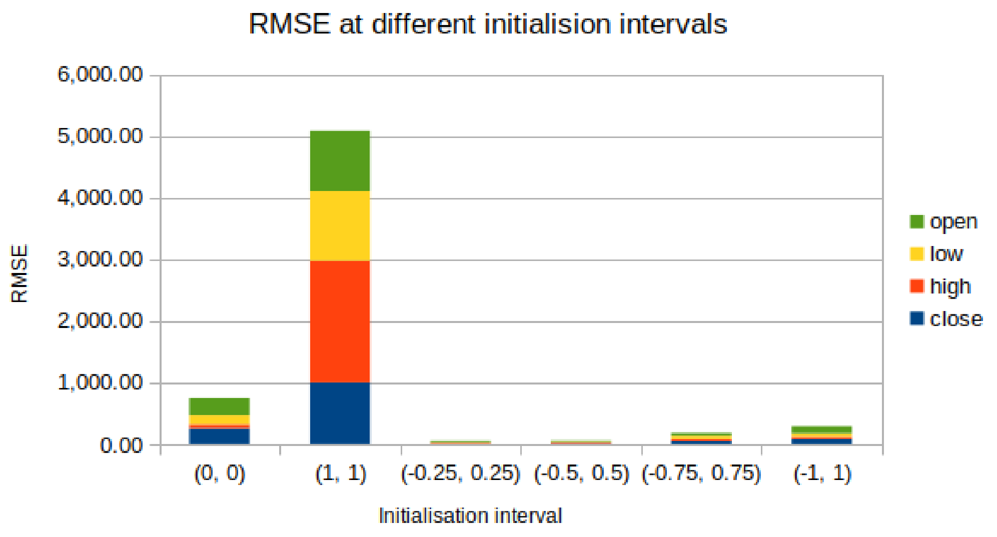


Figure 6.2: Root Mean Squared Error at different initialisation intervals.

Uniform initialisation leads to bad results at either end of the spectrum. Although much faster than random initialisation, convergence happens seemingly instantly and leads to wrong predictions.

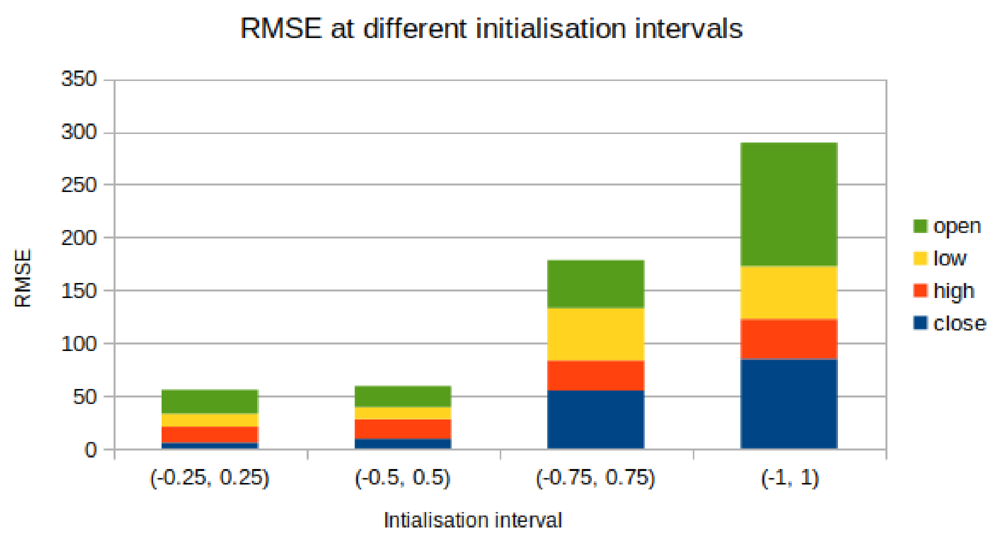


Figure 6.3: Root Mean Squared Error at different initialisation intervals.

The optimal values for random initialisation appear to be near an interval minimum of -0.25 and an interval maximum of 0.25, where prediction accuracy is highest, and the network also converges quickest at 48 seconds compared to 86 seconds at the (-0.5, 0.5) interval.

6.2.2. Optimisation Algorithm

In addition to the initialisation method, the optimisation algorithm used to adjust the weights on each iteration impacts both the speed of convergence, as well as the accuracy of the prediction [25]. Stochastic Gradient Descent (SGD) methods are perhaps the most commonly used optimisation algorithms. The Adam optimiser is used by default in the product. The accuracy of the predictions at different step sizes is benchmarked against the 'default' Stochastic Gradient Descent optimiser used by MLPack.

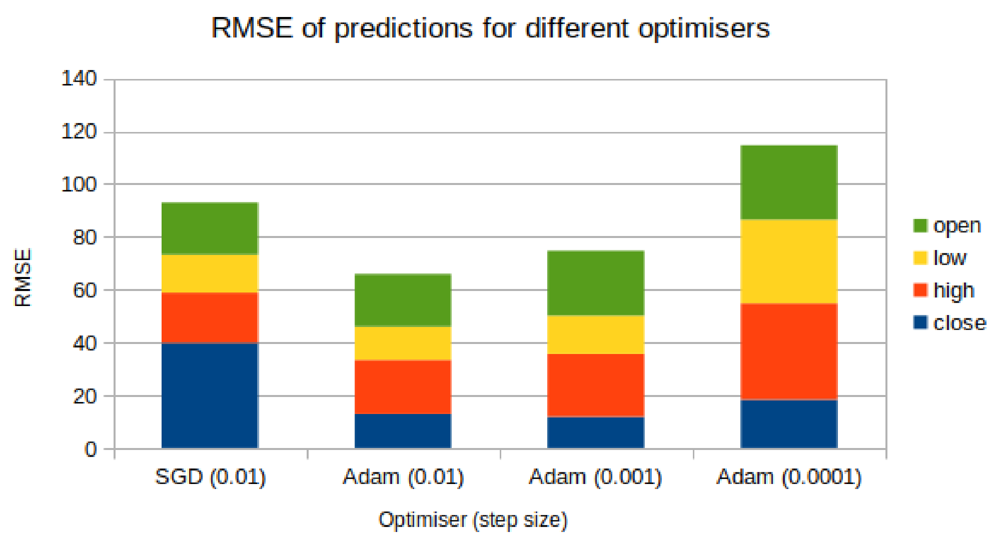


Figure 6.4: Root Mean Squared Error benchmark of different optimiser functions.

Each bar represents the root mean squared error, or RMSE, of one of the prediction outputs for a given optimiser function. The total height of the bar represents the summed error of all outputs. The time costs in seconds for predicting 50 values using single-fold cross-validation are as follows, from SGD(0.01) to Adam(0.0001): 153.59, 80.48, 215.82, 876.13.

6.2.3. Look-back - Rho

The rho value sets the length of the backpropagation through time (BPTT) used to train the network. Effectively; it determines how many recent values are used to predict the output at the next step in the time series.

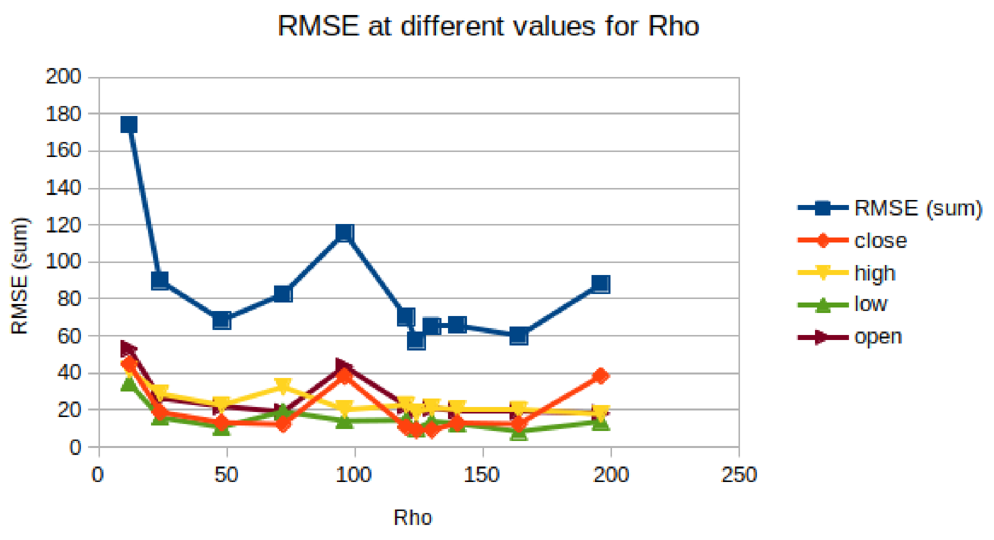


Figure 6.5: Root Mean Squared Error benchmark at different rho values.

Similar to the optimiser functions, the different values for rho are found by predicting 50 values using single-fold cross-validation. Each line represents the root mean squared error of one of the four outputs, followed by the sum of these RMSEs at a given value for rho.

6.2.4. Hidden Layer Complexity

The complexity of the hidden layer refers to the number of hidden nodes used in the FastLSTM layer. Increasing the number of nodes brings the potential to learn more complex rules, but requires more data to train and runs the risk of losing the ability to generalise (and thus may result in inaccurate results if the dataset is not sufficient). As in the previous two figures, the RMSE of each output channel is visualised alongside the sum of all channels, at different hidden layer sizes.

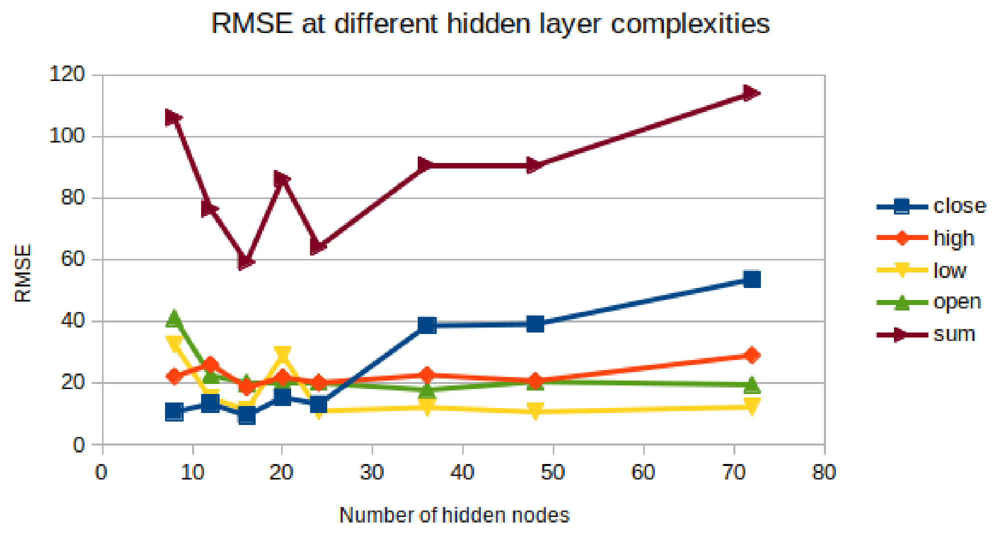


Figure 6.6: Root Mean Squared Error benchmark at different hidden layer complexities.

The sum of RMSEs is minimised when approximately 8 hidden neurons are used, with a close second at approximately 24 hidden neurons. Predictions of the closing price, in particular, appear to lose much accuracy when using 30 or more hidden neurons.

6.3. Trading Strategies

This section discusses the results obtained by validating the trading engine on historical data; a technique popularly known as backtesting. The set of historical data points acts as a simulation of live input and updates the portfolio by executing ’dummy’ trades. The performance of each trading strategy in minimising losses to the portfolio is dependent on the quality of the prediction used to determine a course of action, as well as the quality of the strategy itself. Several strategies are validated below. Two figures are presented for each strategy; the first displays the results of over the entire length of the historical data, while the second considers a rerun over a recent section of the dataset.

To aid in interpreting the results correctly the following is noted: the figures presented consider one asset (e.g. the Bitcoin) to be the reference currency. At each point in time the total value of the portfolio is converted into its equivalent reference currency value as follows:

$$value_{portfolio} = volume_{referencecurrency} + \frac{volume_{pairedcurrency}}{value_{referencecurrency}}$$

A successful trader should, therefore, end up with more (or approximately equal volume) of the reference currency or asset that is being considered. A decrease in reference currency volume indicates a loss of value, even if the monetary value of the asset rises.

6.3.1. Default Parameters

The experiments presented in the following sections make use of the default configuration of the network as delivered with the project. Additional restrictions are imposed: a maximum portfolio value of 25% can be bought or sold during a single transaction, together with a minimum volume of 0.01 of the reference currency, and a transaction fee of 0.1%. Each experiment concerns trading the BTC/USDT pair with initial portfolio volumes of 0 BTC and 10,000 USDT.

6.3.2. Random Walk

A controversial yet widely regarded investment theory known as the efficient market hypothesis[8] states that all available information is immediately reflected in the value of an asset, and therefore no consistently profitable trading strategies should exist. Consequently, if we consider this hypothesis valid, a series of random actions should be able to approximate the results of a series of calculated trades. As a secondary baseline to loss minimisation, the averaged results of several such random walks are presented.

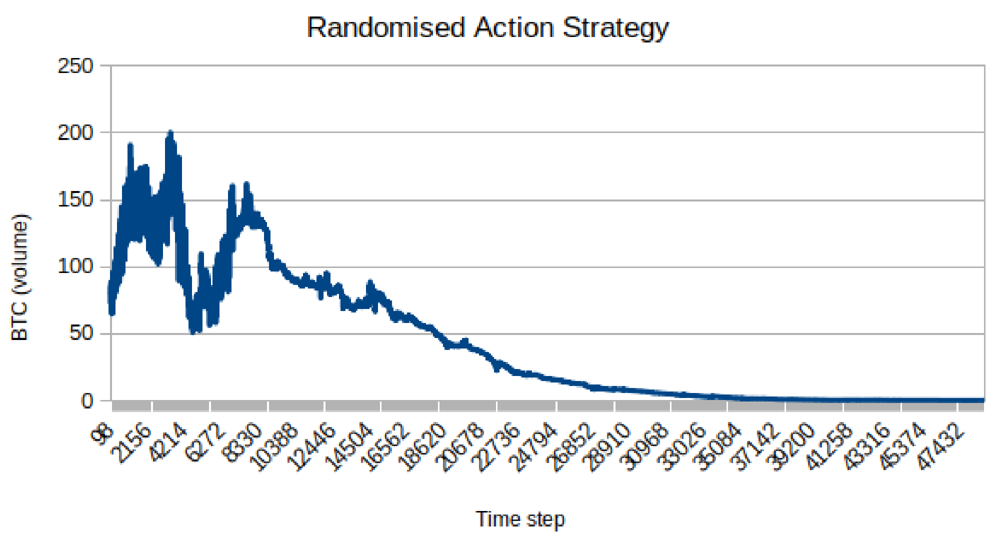


Figure 6.7: Random Walk - Average of 10 runs over the entire dataset.

A random pattern of actions results in a sharp decline of asset volumes after some initial profits. Virtually nothing remains of the initial volume after approximately 30,000 time steps.

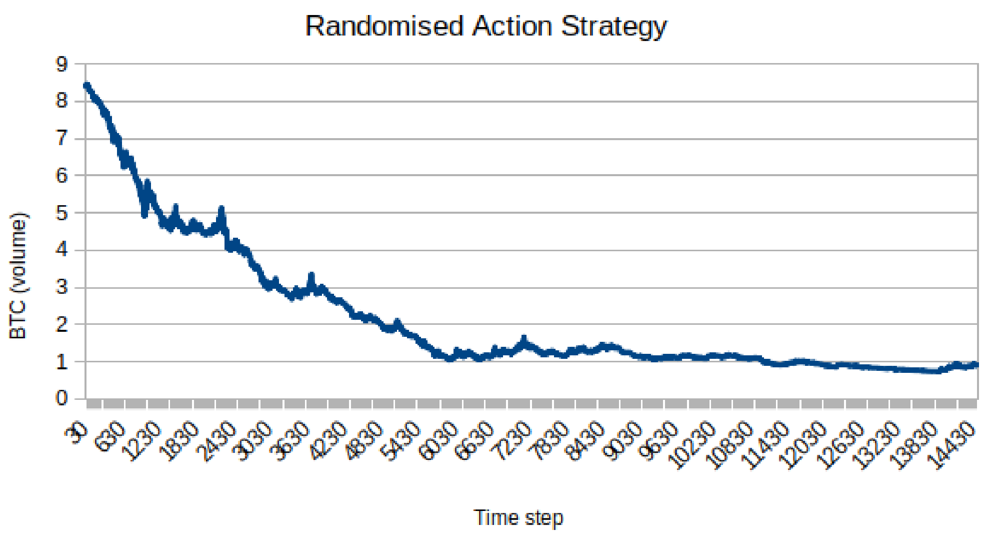


Figure 6.8: Random Walk - Average of 50 runs over the most recent 30% of the dataset.

A similar trend can be observed when backtesting a randomised strategy on only a recent section of the dataset. A sharp decline with intermittent periods of stable asset volume is observed. Approximately 10% of the initial volume remains after 6,000 time steps.

6.3.3. Rate of Change

The Rate of Change strategy considers the change of the predicted value relative to the last actual value. When the change is below, in between, or above a lower and upper threshold, a decision is made whether to sell, buy or hold.

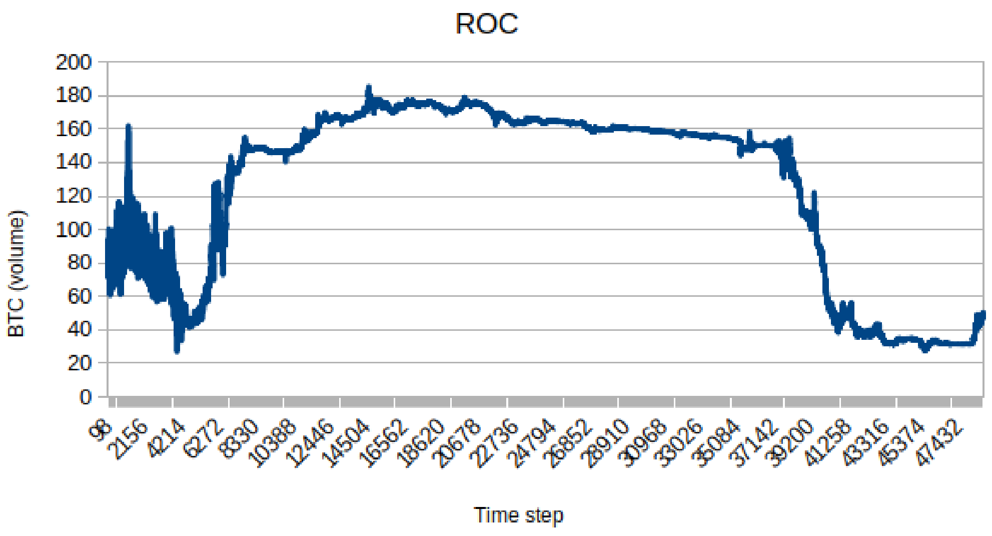


Figure 6.9: Performance of the ROC strategy over the entire the dataset.

The ROC Strategy suffers some initial decline in portfolio value, yet recovers from this loss and even manages almost to double the initial volume at some point. It manages to maintain a reasonably steady value throughout the majority of the subsequent time steps. However, a sharp decline in value can be observed around the 39,000th time step until the 41,000th time step. After this decline, the strategy appears once again capable of maintaining the value and starts making a minor recovery near the end of the period.

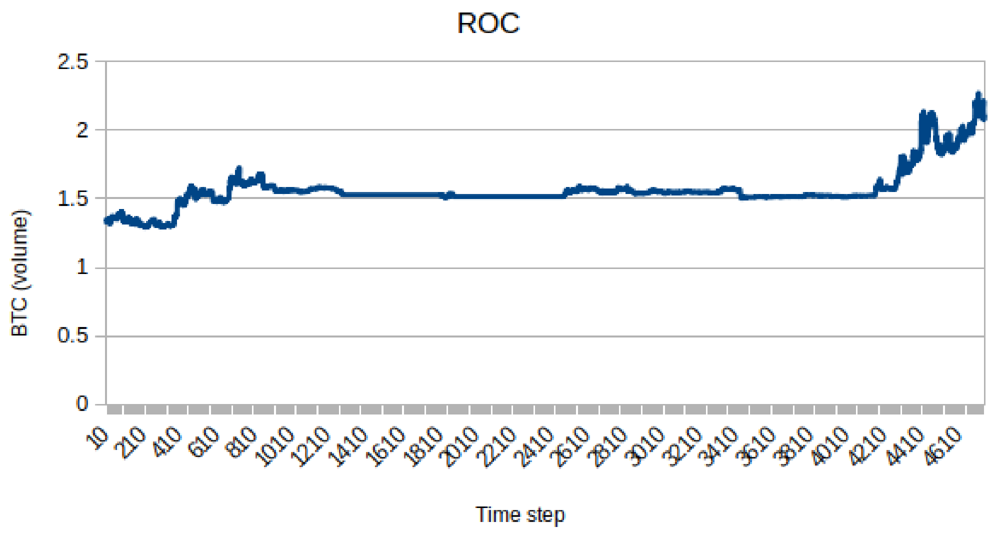


Figure 6.10: Performance of the ROC strategy over the most recent 10% of the dataset.

Across the most recent 10% of the dataset, the portfolio value under the ROC strategy remains relatively stagnant. Any trades that do occur do not result in significant profits or losses, until the end of the period where one can see an increase in portfolio value.

6.3.4. Relative Strength Indicator

The Relative Strength Indicator strategy tracks upward and downward trends in asset value. A sufficiently long series of positive value changes result in an asset being marked as overbought (indicating the holder should sell) and vice versa.

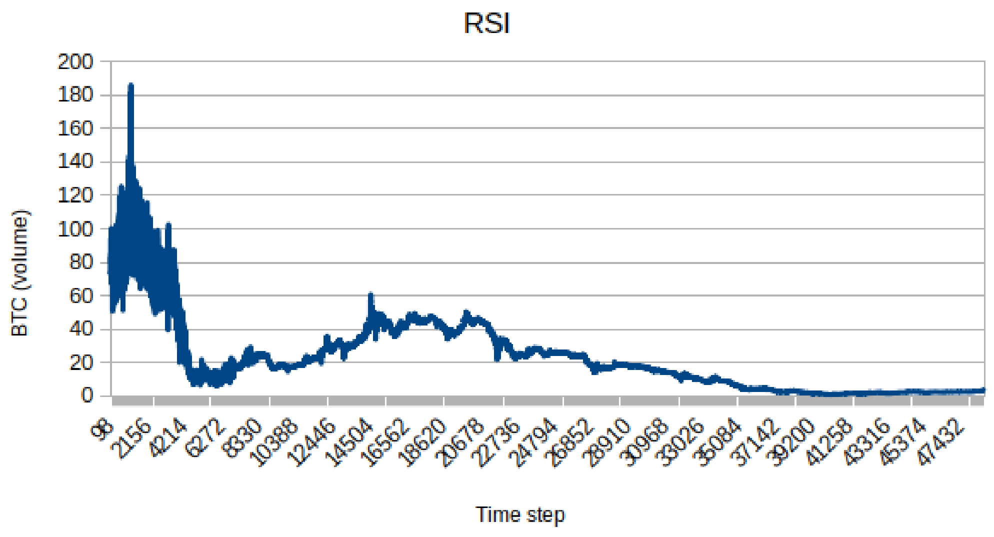


Figure 6.11: Performance of the RSI strategy over the entire dataset.

Immediately noticeable is the high variance during the first 4000 time steps of the backtesting process. Major profit is made and subsequently lost during this period. Afterwards, total value of the portfolio steadily rises to approximately 75% of the initial value, but is then lost over the last 25,000 time steps until a small recovery can be observed at the tail of the period.

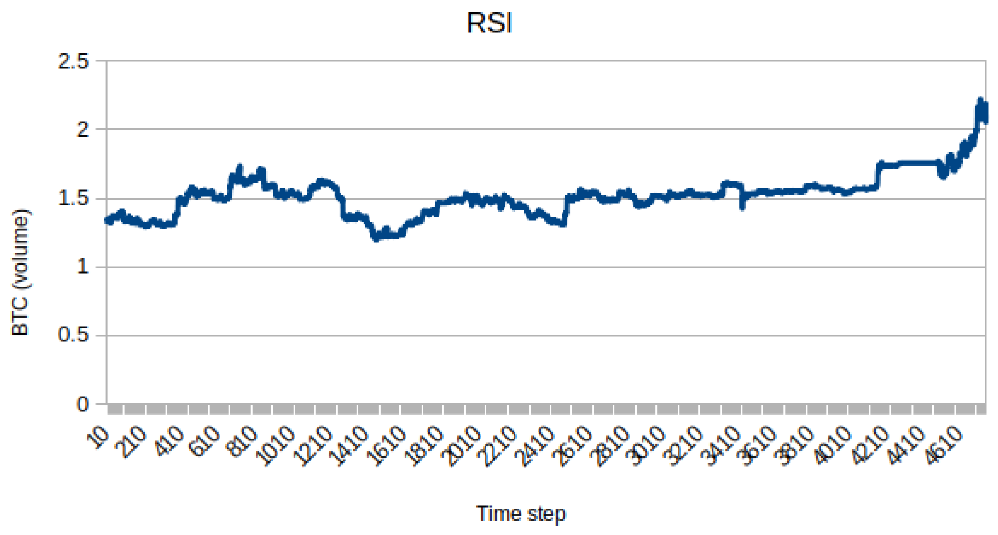


Figure 6.12: Performance of the RSI strategy over the most recent 10% of the dataset.

Rerunning the process across only the most recent 10% of data displays a reasonable ability to maintain and even slightly grow overall portfolio value.

6.3.5. Double Exponential Moving Averages

The Double Exponential Moving Averages strategy, or DEMA, utilises two moving averages to determine whether to buy, hold or sell an asset. Crossovers of the two moving averages indicate buy and sell moments as momentum between long-term and short-term value of the asset changes.

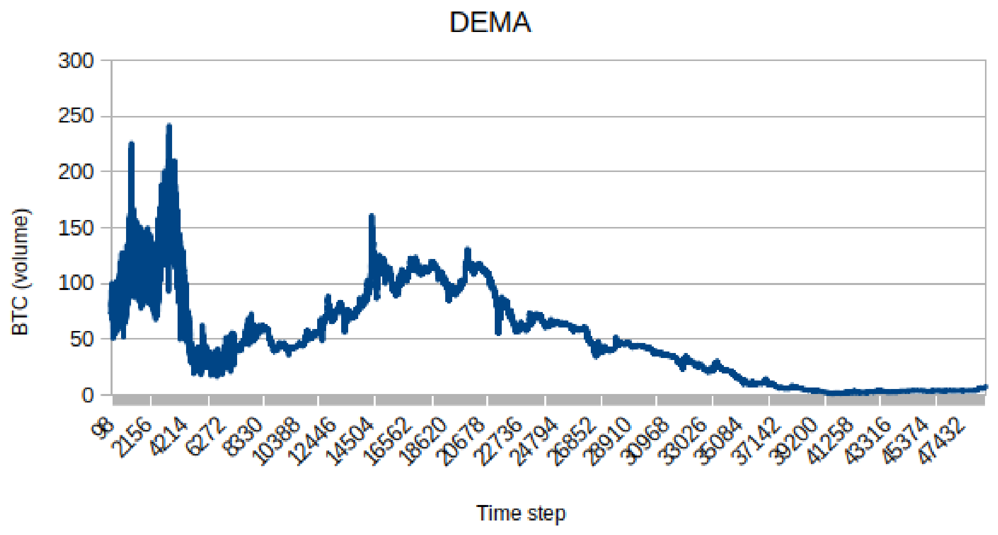


Figure 6.13: Performance of the DEMA strategy over the entire dataset.

The DEMA strategy incurs occasional heavy losses but seems to recover from them throughout a large number of steps. However, after approximately 20,000 time steps a persistent drop in volume is observed until a near-zero point is reached after 40,000 time steps.

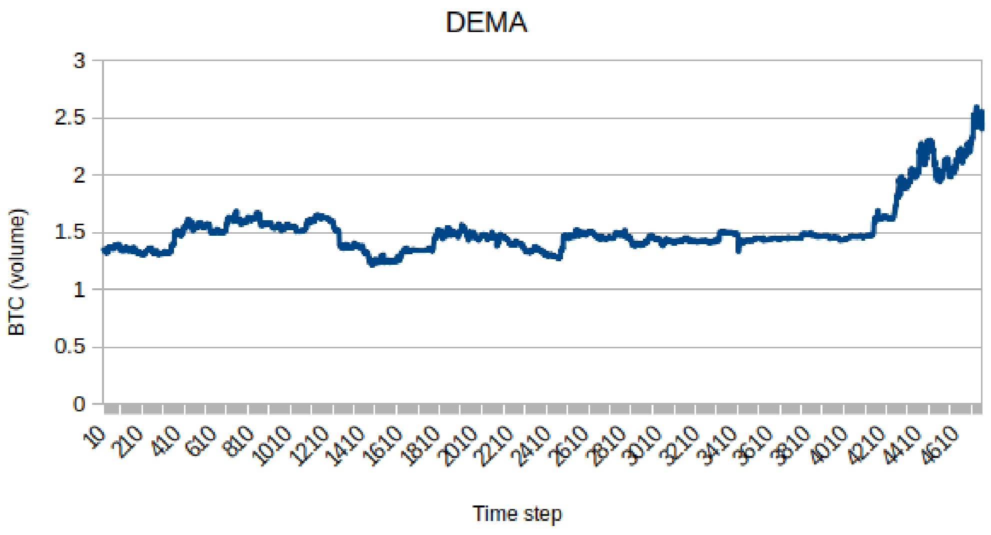


Figure 6.14: Performance of the DEMA strategy over the most recent 10% of the dataset.

When performing a backtest over the most recent 10% of the dataset, the DEMA strategy is subject to several minor periods of profit and loss. It manages to achieve a portfolio value slightly above the initial value and makes considerable gains during the final period of the test, nearly doubling the initial value.

6.3.6. Moving Average Convergence / Divergence

The Moving Average Convergence / Divergence strategy is based on the difference between two moving averages, with different lengths, called slow and fast moving average. This is then compared to the moving average of the difference itself, called the signal line. A decision to buy, hold or sell, is then made whenever the difference crossed the signal line.

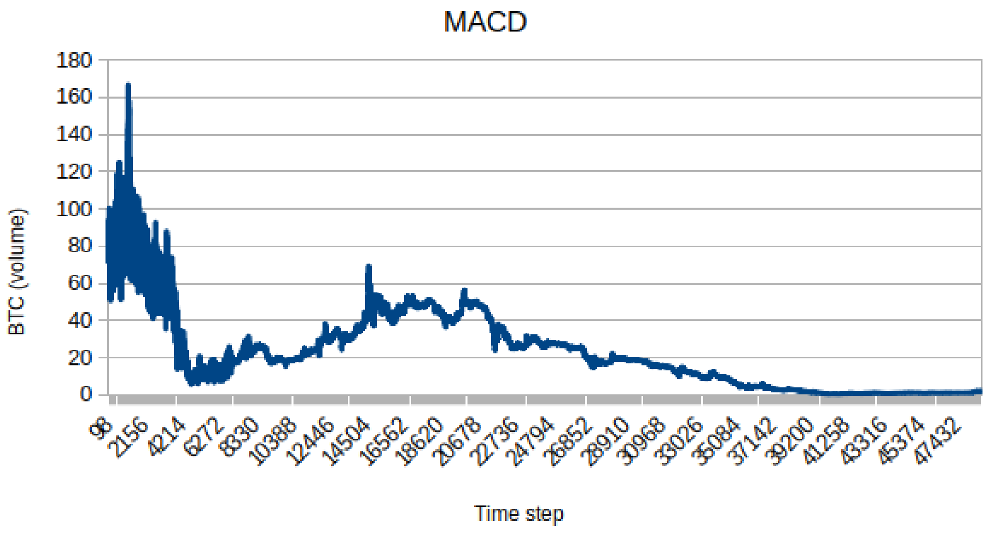


Figure 6.15: Performance of the MACD strategy over the entire dataset.

Similar to the DEMA strategy, the MACD strategy suffers initial losses, followed by a period of recovering (part of) the lost assets. Subsequently, the value of the portfolio declines over an extended period until reaching near-zero after 40,000 time steps.

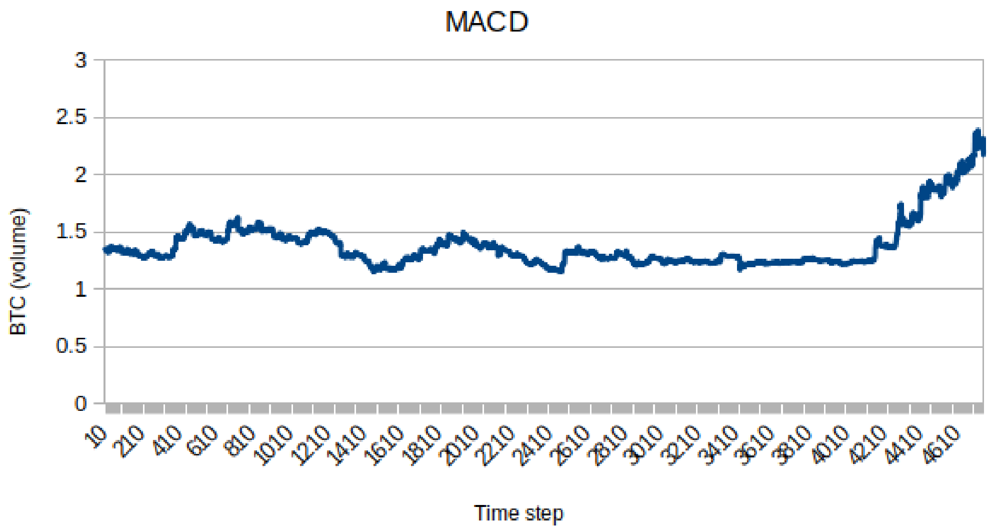


Figure 6.16: Performance of the MACD strategy over the most recent 10% of the dataset.

Performing a backtest over only a recent section of the dataset yields significantly better results. The MACD strategy is capable of maintaining an approximately equal portfolio value over several thousand time steps and ultimately manages to make significant profit.

6.4. Conclusion

In this chapter the most important findings of the research have been demonstrated. The quality of obtained predictions is presented, alongside impact of several network parameters on these predictions. The quality of the built-in trading strategies are evaluated in combination with predicted asset prices through a method known as backtesting, with varying outcomes.

7 SIG feedback

This chapter is dedicated to stating and reflecting upon the feedback provided by the Software Improvement Group on two consecutive versions of the product's code base. The first version was delivered as part of the deadline on December 21, 2018. The second version was delivered as part of a second deadline on January 18, 2019. Feedback on the first version was received on January 25, 2019, the afternoon of the final project deadline. It has therefore not been possible to incorporate this feedback into the final product. Nevertheless, some reflection on SIG's assessments is provided below.

7.1. Feedback

Your code scors 4 stars in our quality model for maintainability, meaning that your project scores above market average. You have not received a higher score due to a lower score for Unit Size.

For Unit Size we consider the length of a method. Methods that are smaller are generally easier to maintain, because they have a clearly defined responsibility and scope. Larger methods tend to contain multiple responsibilities, which can be problematic during maintenance as this tends to lead to large methods getting even larger over time.

In your project, addDataSlowly() in dummeHelpers.js would be an example of a method that contains multiple responsibilities. This method is of course not that large, but it's still noticeable that it both defines some test data as well as handling the promises for processing the data. This code appears to be part of some test/dummy/mock code, so maybe it is only used for tests. However, it is currently not marked as such, so we include it as part of the production code. You can address this by either rewriting the method, or by specifically including this file as part of the test code to make sure it is not used in production.

Another example is convertDataFormat() in graphHelpers.js. This method does have a clearly defined scope, but the repeating code makes the method larger than needed. Rather than repeating the same JSON block twice, create a new method that creates the block and then call that twice with different parameters to create the desired JSON data structure.

As you can see these are relatively small examples. Your maintainability rating is already quite high, which is why there are no glaring issues that need to be addressed.

Finally, it’s good to see that you’ve also written unit tests. The amount of tests is acceptable, but ideally you would have a slightly higher ratio as the relative amount of test code in a project tends to drop over time (as at some point people focus more on adding new functionality).

7.2. Reflection

The software development team generally agrees with the feedback provided by SIG. Some methods are too large, which construct options or input data of the implemented libraries, or they are dummy functions to test data or functionality. The ’addDataSlowly’ function has to be marked specifically as a dummy method, although the file, in which it is written, already is called ’dummyHelpers.js. Also, the group agrees with the feedback on the ’convertDataFormat’ function in the ’graphHelpers.js’ file.

The last point that they stated was increasing the number of unit tests in the code. There are quite some files in the code, which form a wrapper for the library, that was used. So in other words, it does not add any functionality, and that is also the reason, why there are no or little unit tests for it. However, the group has increased the number of unit tests in the next version of the code to provide a solution to this problem.

7.3. Code quality tools

The trading engine follows the code style of the Google [7] to apply one code style through the whole project. Furthermore, the code utilises Clang to analyse the code. It applies static analysing to identify possible bugs. Also, the code is fully documented and tested. Subsequently, the documentation could be generated using Doxygen.

7.4. Internal code conventions

Other than applying code quality tools, some internal conventions have been applied to the code base. The SOLID principles have been applied; an example is that the code has been split up in several packages, whereas one package has one responsibility. The same holds for classes, where a class has one responsibility. The goal of these conventions is improving the maintainability and readability.

7.5. Version control

Finally, the code has been updated using Git, using the branching model of Gitflow [6]. Gitflow introduces a branching model with prefixes so that it is clear what kind of changes are introduced in the branch. In case a branch has been finished, a pull request is made, and the code is reviewed extensively before the code is merged into the develop branch. The master branch will only contain production-ready releases, which will be fetched of the develop branch when the releases have been finished. In this case, it was the end of the mid-term period and the end of the project.

7.6. Conclusion

In summary, because SIG sent the feedback of the quality of the code in the afternoon of the day, that the report has to be handed in, the group did not process the feedback directly. However, after the first version of the code has been sent to SIG, the group already processed some of the feedback in the new version of the code. Therefore, this chapter also includes the introduction of the different implemented code quality tools, the internal code conventions, and our version control model, to state what has been done to improve the code quality.

8 Discussion & Recommendations

This chapter examines the findings of the research and reviews the obtained results. An effort is made to answer the research question, explain the impact of the findings, and limitations of the research. Recommendations are made for future work on the subject.

8.1. Discussion

As evidenced by the results chapter, the network parameters play a considerable role in both accuracy of the obtained results, as well as convergence speed when training the network. The optimal rho value appears to be approximately 120, or 5 days of recent hourly intervals. An increase in the amount of data used for a prediction adds context, thereby increasing the accuracy. However, at some point, the context is no longer relevant and instead becomes noise. Similarly, an increase in the number of nodes gives the network the ability to learn more complex associations but too many nodes may yield incorrect results.

Results obtained from the TSKCV suggest that the network is capable of predicting new asset values with approximately 0.4% error across the four output channels. However, the strategies have varying rates of success in maintaining a steady portfolio value throughout the historical dataset. Although every single one appears to perform better than the random walks presented at the start of the previous chapter, none appear to predict large and sudden drops in asset value correctly. The ROC strategy seems somewhat capable of maintaining portfolio value throughout the backtesting period, while all other strategies lose the vast majority of the portfolio before the end of the test. The drop in value around the 39,000th step coincides with a significant price increase of the Bitcoin (Figure 8.1). This indicates that the network fails to predict and utilise the upward trend; a tendency that is observable across all strategies. As the network has been trained on the entirety of the dataset before predicting the first value (in the interest of execution time) it is conceivable that it fails to classify sequences from earlier points in the series correctly. Retraining the network after each prediction, as is done in cross-validation, could yield considerably better results when backtesting across the full size of the dataset.

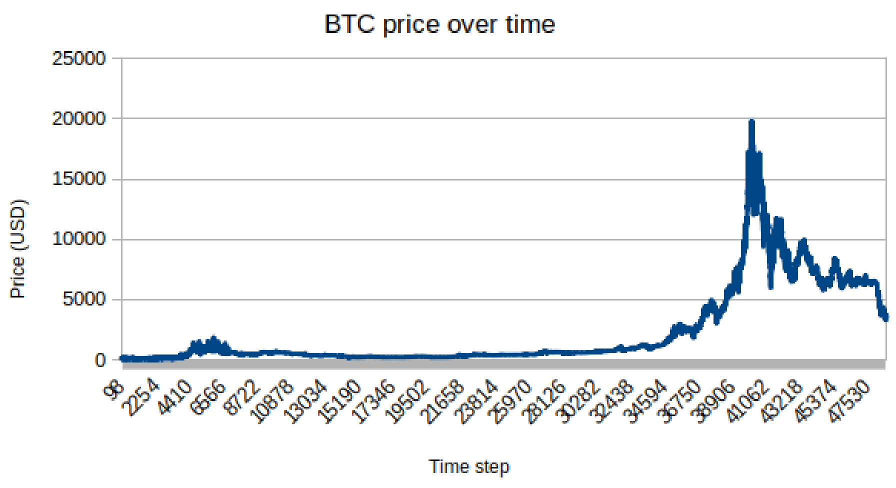


Figure 8.1: USD price of the Bitcoin throughout the backtesting dataset.

When comparing the results obtained by backtesting across the recent partition alone, the network does however seem capable of predicting decline in asset value to some extent. Major increases in portfolio value can be observed at the end of the period, which coincides with a price decline of the Bitcoin. When comparing these results to those obtained by an 'optimal' variant of the product (that is; one that is capable of predicting future asset prices with 100% accuracy), significant discrepancy is still noticeable (figures 6.16, 8.2).

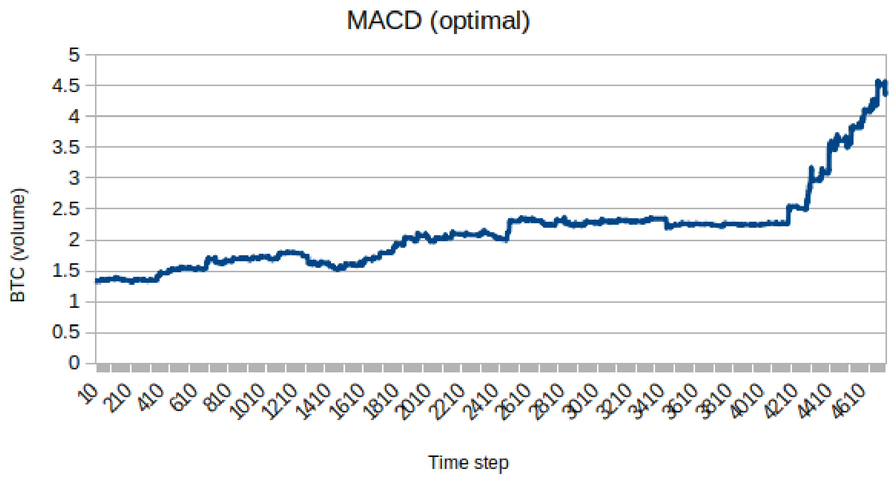


Figure 8.2: Results of backtesting the MACD strategy over the most recent 10% of the dataset when optimal predictions are provided.

8.2. Recommendations

Due to time limitations, it was not possible to fully develop the whole application, as Blockrise envisioned. In other words, the system could be improved, by not only focusing on the trading engine but also on the application around the trading engine. Additionally, the trading engine could be extended and improved, to give better predictions. Following the predictions, the decision-making process of deciding what kind of decision signals should be given, could also be improved. The details of how the systems could be improved, are elaborated in the following sections.

8.2.1. Add volume data

Volume measures in monetary value what quantities an asset has been traded in a specific given period of time. It is used as a technical indicator, to indicate the relative strength of a market movement [4]. In the current system, the volume data was excluded, due to not knowing from which exchanges the volume data originated. Therefore, if the system would pull the data of a new hourly candle, the volume that it would receive differs significantly from the input data, the neural network would train on. If the system would have implemented it this way, it could lead to wrong predictions.

Therefore, a recommendation is to add the volume data to the input again. However, to do this, it is necessary to research which exchanges the volume data of the input data originated. As aforementioned, the reason for this is that the problem is that the candles of the live market environment, cannot be mapped back to the same standard as the training data. However, if it is known, which exchanges are used to pull the volume data from, then it also can be reproduced in the live market environment, to fetch the live volume data.

8.2.2. Add order book data

Order book data shows the list of all orders placed by buyers and sellers for a given specific asset. The orders are grouped by price, and follow the principle: ”first in, first out”. In other words, if a person x places an order for the same price as a person y, then person x will have his order fulfilled first. When there is an imbalance in orders, it shows the spread, and will most likely give about the course the price will take [2].

Although the recommendation is not to add the order book data to the input data list, a recommendation is to use it to decide a trade. Since the price of an asset, is the price of the last trade, that occurred, it does not mean, it reflects the price on which you can buy or sell the asset. For a trade, that the system suggests, it has to be validated by looking if an order exists in the order book so that it can be satisfied. In the worst case, it would result in the scenario, that the system would place an order, that would be satisfied. Therefore, adding order book data as a deciding factor for a trade is an improvement to the system.

8.2.3. Add other samples

Other sample data, such as the data of the futures market of the Bitcoin could be added as well. The future market of the Bitcoin adds another perspective to the market, by offering futures. These financial contracts are contracts which obligate the user to either buy or sell an asset. A future, which obligates the user to buy an asset, is called a long, and a future, which obligates the user to sell the asset, is called a short [1]. Since the contracts represent the Bitcoin market as well, an imbalance could also give hints of the price.

Therefore, a recommendation is to add other sample data, such as the long and short data. Other sample data could also be used, such as other similar assets, so that the trading engine can find relations in those two samples. The current implementation contains one perspective of data, which is the hourly candlestick data. However, if the system has more data, it is possible that it could find relations between the different samples as well. Consequently, this could lead to better predictions.

8.2.4. Add other exchanges

Adding multiple exchanges comes with multiple benefits. By solely depending on one exchange, much crucial information about the asset can be missed. This varies from different prices to different volumes and availability of assets. When multiple exchanges are added, the engine can look for better, more efficient trades on other markets, and be more profitable.

Hence, the recommendation is to add other exchanges as this has multiple benefits. In the current solution, the use of exchanges is limited to one, Binance to be specific. Binance is the most reliable exchange for this purpose; the API was fitting and matched best with the implementation. Adding other exchanges might be a bit harder to do, due to compatibility issues, but the benefits it comes with is only affecting the engine for the better.

8.2.5. Use other intervals

Where using smaller intervals results in more but less trustworthy signals, using longer intervals, results in less, but more trustworthy signals. However, combining multiple time intervals could introduce an increase of intelligence for the trading engine. An option would be to use a longer time interval to predict the general trend and use the data of a smaller interval to indicate a possible reversal of the trend. The other way around could also be possible, where the system would use the smaller time interval to find possible reversals, and use the larger time interval to confirm the reversal [3].

Thus, a recommendation is to use other intervals as well, the current system solely uses the hourly candlestick data, but this could be extended to use other intervals as well. It could be done by adding it as another sample but it can also be done by introducing other neural networks for each interval. These neural networks could be combined to complement each other. Besides, multiple neural networks could also be used to predict different data, such as specific neural networks that predict reversals in trends.

8.2.6. Combine multiple strategies

Different strategies lead to different outcomes. Some strategies might perform better with certain assets and at different time intervals. Combining strategies could also lead to better performance, and could be a good addition to the current system. There are two possibilities when implementing this. You could let the AI determine the best(yields the best results) combination of strategies, or let the user make the selection and decide for themselves which combination suits them best.

The recommendation is thus to provide either that the AI determines which combination of strategies should be used, or that the user can figure out for themselves what combination to use. This all comes down to what the user would like to see in the final product, and this is further explored in 8.2.8.

8.2.7. Optimise network parameters

The trading engine contains a lot of parameters, which have been optimised individually, they have not been optimised in combination with each other. However, if all the network parameters would have been optimised in combination with each other, it would mean that the total amount of combinations is exponential. In other words, testing all possible combinations of the network parameters is almost impossible.

Therefore, a recommendation is to write a program, that optimises the parameters of the network. This program could also utilise AI to determine the optimal parameters. Consequently, optimal parameters should have resulted in better predictions, and better strategies as well.

8.2.8. Change the interface to emphasise UX

With the final application in mind, interviewing with end-users could be beneficial. Users might prefer several features over others and might give input about the functionalities, which the developers could have missed. The user also could help the developers out with design choices, for example, what feature could be included, and what feature is best left out.

Thus it is essential that an end-user interview is done, which provides better insights on how well target users perceive the application. Next to that, it could help with design choices, that are inevitable, and it could identify possible faults or errors.

8.3. Conclusion

The results as presented in the previous chapter were reviewed; the differences between the results obtained when predicting new values, and those obtained when backtesting were examined. Recommendations are given for modification and extension of the product, aimed at improving the accuracy of predictions, and usability related to customisation of the product and enhancing the user experience.

9 Conclusion

This project was centred around building an AI-based Trading Engine for the company Blockrise. The engine should be able to manage a portfolio of digital assets, in the most cost-efficient way with a base goal of loss minimalisation. Product requirements were set based on the business requirements formulated by Blockrise, and a research question was formulated:

> *Can an AI trading engine utilising trading indicators maintain a stable monetary value of a digital asset in a live market environment?*

Following a period of research and implementation, the product has been built and tested and could be considered for inclusion in a consumer-ready application as envisioned by Blockrise. The requirements, as set beforehand with the stakeholders, were divided into categories based on priority. The most important requirements, stated in 3.2.1, have all been implemented successfully. From the requirements stated in 3.2.2, all have been implemented, with the exception of the backup functionality. The last category of features, as stated in 3.2.3, have not been explicitly implemented due time constraints. Nevertheless, extension of the product to include these features should be trivial.

Throughout the project an effort was made to answer the main research question and sub-questions as posed in section 3.3. The final product utilises an FastLSTM-based recurrent neural network to predict future stock prices. Network inputs are a combination of direct data (four values representing asset value) and seven values derived from this data (technical indicators), while the four outputs represent these same four asset values for the point in time that is being predicted. The decisions to place orders, or to withhold from doing so, are clarified in section 5.1.4 and based on popular trading strategies. Performance is measured using the Root Mean Squared Error function, which emphasises the gravity of large prediction errors.

The results show that the product is capable of accurately predicting new values at the current point in time, and thereby maintaining or growing portfolio value. These results are however only empirically proven for the specific setup used in the project, and therefore may not hold in all cases. It is not unlikely that the product fails to predict one-off events which could nevertheless significantly impact the user's portfolio value. Further research is required to properly investigate the boundaries of the product and the validity of these hypotheses. Improvement of the product's capabilities could provide further aid in this investigation. As an overall conclusion the results appear promising given the limited time span of the project and team's knowledge of the field. Therefore, further pursuit of the notion of AI-powered asset trading is encouraged.

A Infosheet

Title of the project: AI Trading Engine for Digital Assets
Name of the client organisation: Blockrise
Date of the final presentation: February 01, 2019
Description: Stock markets for asset trading have been part of both local economies and the global economy for centuries. Extensive research has been carried out to discover the rationale and patterns behind their functioning. The recent rise of machine learning has provided new ways of uncovering such patterns, increasingly sparking the interests of researchers and stakeholders worldwide. This project aimed to apply such methods in an attempt to make accurate prognoses about the stock market. The resulting product is a digital asset trading engine capable of making market predictions, providing trading advice based on these predictions and managing digital assets.

Team members:

Name	R. van Gurp
Interests	Artificial Intelligence, (Product) Management
Main contributions	Architecture, auxiliary functionality, unit testing
Name	Y. J. Hu
Interests	Finance, Complex Algorithms, Web Development
Main contributions	Architecture, Articial Intelligence, complex algorithms, data visualisations
Name	H. V. Kooijman (Woyak)
Interests	Complex Algorithms, Web Development
Main contributions	AI, data processing,
Name	A. Somai
Interests	Big Data, Entrepreneurship, Stock Market
Main contributions	Auxiliary functionality, unit testing, performance testing

All members contributed to writing the research report, the final report and preparing the final presentation.

Client: M. L. C. Jacobs and J. Lazet, Founders of Blockrise

TU Coach: Prof. Dr. J. S. Rellermeyer

Contact Person: Y. J. Hu, y.j.hu@student.tudelft.nl

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

B System Decomposition

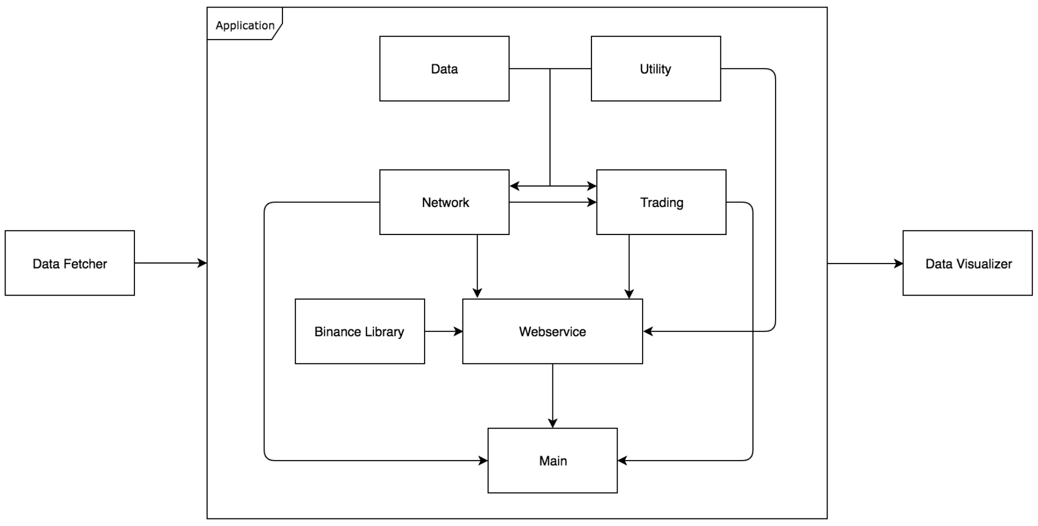


Figure B.1: Illustration of the system decomposition of the trading engine.

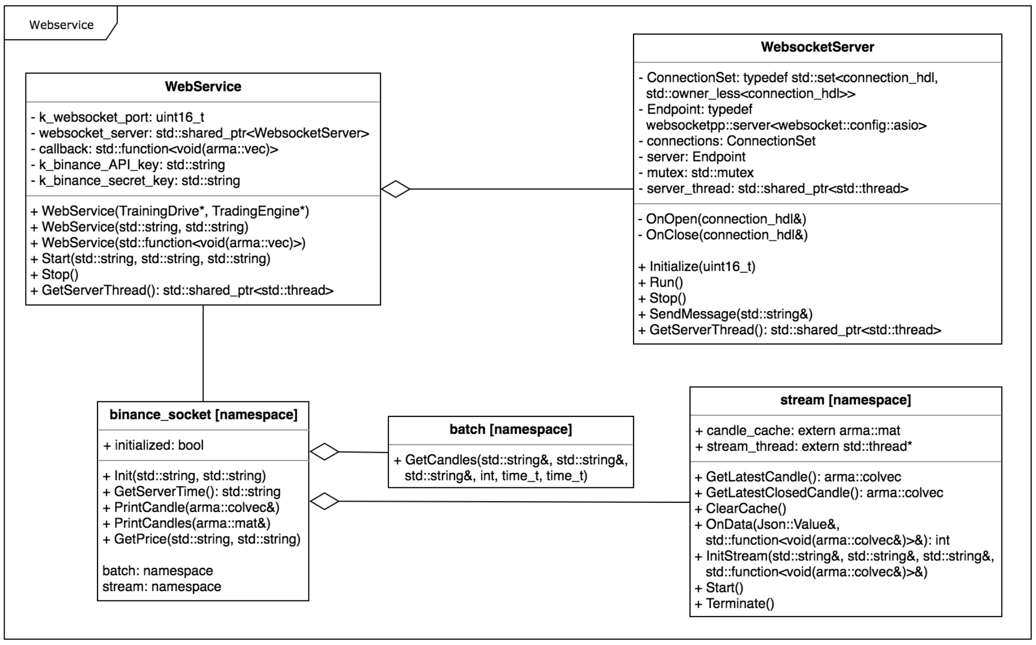

Webservice
WebService
- k_websocket_port: uint16_t
- websocket_server: std::shared_ptr<WebsocketServer>
- callback: std::function<void(arma::vec)>
- k_binance_API_key: std::string
- k_binance_secret_key: std::string
+ WebService(TrainingDrive*, TradingEngine*)
+ WebService(std::string, std::string)
+ WebService(std::function<void(arma::vec)>)
+ Start(std::string, std::string, std::string)
+ Stop()
+ GetServerThread(): std::shared_ptr<std::thread>
WebsocketServer
- ConnectionSet: typedef std::set<connection_hdl, std::owner_less<connection_hdl>>
- Endpoint: typedef websocketpp::server<websocket::config::asio>
- connections: ConnectionSet
- server: Endpoint
- mutex: std::mutex
- server_thread: std::shared_ptr<std::thread>
- OnOpen(connection_hdl&)
- OnClose(connection_hdl&)
+ Initialize(uint16_t)
+ Run()
+ Stop()
+ SendMessage(std::string&)
+ GetServerThread(): std::shared_ptr<std::thread>
binance_socket [namespace]
+ initialized: bool
+ Init(std::string, std::string)
+ GetServerTime(): std::string
+ PrintCandle(arma::colvec&)
+ PrintCandles(arma::mat&)
+ GetPrice(std::string, std::string)
batch: namespace
stream: namespace
batch [namespace]
+ GetCandles(std::string&, std::string&, std::string&, int, time_t, time_t)
stream [namespace]
+ candle_cache: extern arma::mat
+ stream_thread: extern std::thread*
+ GetLatestCandle(): arma::colvec
+ GetLatestClosedCandle(): arma::colvec
+ ClearCache()
+ OnData(Json::Value&, std::function<void(arma::colvec&)>&): int
+ InitStream(std::string&, std::string&, std::string&, std::function<void(arma::colvec&)>&)
+ Start()
+ Terminate()

Figure B.2: Illustration of the system decomposition of the web-service package.

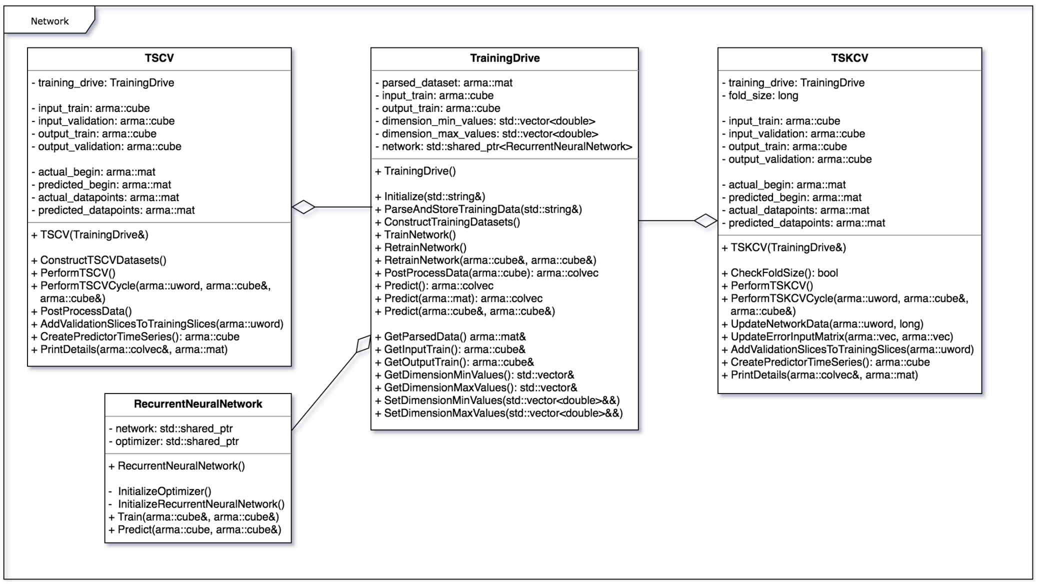

Network
TSCV
- training_drive: TrainingDrive
- input_train: arma::cube
- input_validation: arma::cube
- output_train: arma::cube
- output_validation: arma::cube
- actual_begin: arma::mat
- predicted_begin: arma::mat
- actual_datapoints: arma::mat
- predicted_datapoints: arma::mat
+ TSCV(TrainingDrive&)
+ ConstructTSCVDatasets()
+ PerformTSCV()
+ PerformTSCVCycle(arma::uword, arma::cube&, arma::cube&)
+ PostProcessData()
+ AddValidationSlicesToTrainingSlices(arma::uword)
+ CreatePredictorTimeSeries(): arma::cube
+ PrintDetails(arma::colvec&, arma::mat)
TrainingDrive
- parsed_dataset: arma::mat
- input_train: arma::cube
- output_train: arma::cube
- dimension_min_values: std::vector<double>
- dimension_max_values: std::vector<double>
- network: std::shared_ptr<RecurrentNeuralNetwork>
+ TrainingDrive()
+ Initialize(std::string&)
+ ParseAndStoreTrainingData(std::string&)
+ ConstructTrainingDatasets()
+ TrainNetwork()
+ RetrainNetwork()
+ RetrainNetwork(arma::cube&, arma::cube&)
+ PostProcessData(arma::cube): arma::colvec
+ Predict(): arma::colvec
+ Predict(arma::mat): arma::colvec
+ Predict(arma::cube&, arma::cube&)
+ GetParsedData() arma::mat&
+ GetInputTrain(): arma::cube&
+ GetOutputTrain(): arma::cube&
+ GetDimensionMinValues(): std::vector&
+ GetDimensionMaxValues(): std::vector&
+ SetDimensionMinValues(std::vector<double>&&)
+ SetDimensionMaxValues(std::vector<double>&&)
TSKCV
- training_drive: TrainingDrive
- fold_size: long
- input_train: arma::cube
- input_validation: arma::cube
- output_train: arma::cube
- output_validation: arma::cube
- actual_begin: arma::mat
- predicted_begin: arma::mat
- actual_datapoints: arma::mat
- predicted_datapoints: arma::mat
+ TSKCV(TrainingDrive&)
+ CheckFoldSize(): bool
+ PerformTSKCV()
+ PerformTSKCVCycle(arma::uword, arma::cube&, arma::cube&)
+ UpdateNetworkData(arma::uword, long)
+ UpdateErrorInputMatrix(arma::vec, arma::vec)
+ AddValidationSlicesToTrainingSlices(arma::uword)
+ CreatePredictorTimeSeries(): arma::cube
+ PrintDetails(arma::colvec&, arma::mat)
RecurrentNeuralNetwork
- network: std::shared_ptr
- optimizer: std::shared_ptr
+ RecurrentNeuralNetwork()
- InitializeOptimizer()
- InitializeRecurrentNeuralNetwork()
+ Train(arma::cube&, arma::cube&)
+ Predict(arma::cube, arma::cube&)

Figure B.3: Illustration of the system decomposition of the network package.

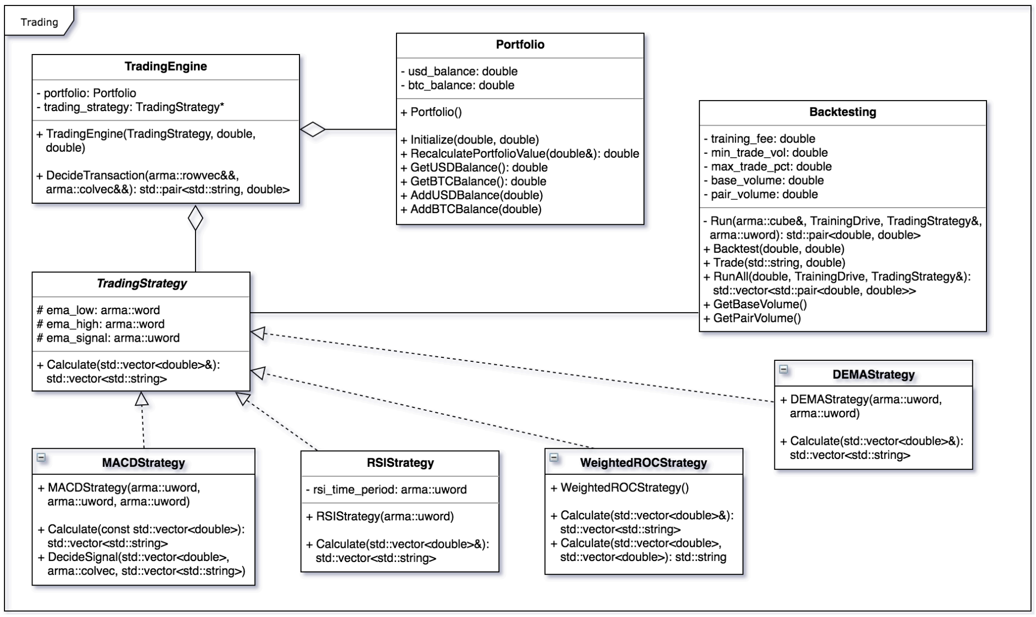

Trading
TradingEngine
- portfolio: Portfolio
- trading_strategy: TradingStrategy*
+ TradingEngine(TradingStrategy, double, double)
+ DecideTransaction(arma::rowvec&&, arma::colvec&&): std::pair<std::string, double>
Portfolio
- usd_balance: double
- btc_balance: double
+ Portfolio()
+ Initialize(double, double)
+ RecalculatePortfolioValue(double&): double
+ GetUSDBalance(): double
+ GetBTCBalance(): double
+ AddUSDBalance(double)
+ AddBTCBalance(double)
Backtesting
- training_fee: double
- min_trade_vol: double
- max_trade_pct: double
- base_volume: double
- pair_volume: double
- Run(arma::cube&, TrainingDrive, TradingStrategy&, arma::uword): std::pair<double, double>
+ Backtest(double, double)
+ Trade(std::string, double)
+ RunAll(double, TrainingDrive, TradingStrategy&): std::vector<std::pair<double, double>>
+ GetBaseVolume()
+ GetPairVolume()
TradingStrategy
# ema_low: arma::word
# ema_high: arma::word
# ema_signal: arma::uword
+ Calculate(std::vector<double>&): std::vector<std::string>
DEMAStrategy
+ DEMAStrategy(arma::uword, arma::uword)
+ Calculate(std::vector<double>&): std::vector<std::string>
MACDStrategy
+ MACDStrategy(arma::uword, arma::uword, arma::uword)
+ Calculate(const std::vector<double>): std::vector<std::string>
+ DecideSignal(std::vector<double>, arma::colvec, std::vector<std::string>)
RSIStrategy
- rsi_time_period: arma::uword
+ RSIStrategy(arma::uword)
+ Calculate(std::vector<double>&): std::vector<std::string>
WeightedROCStrategy
+ WeightedROCStrategy()
+ Calculate(std::vector<double>&): std::vector<std::string>
+ Calculate(std::vector<double>, std::vector<double>): std::string

Figure B.4: Illustration of the system decomposition of the trading package.

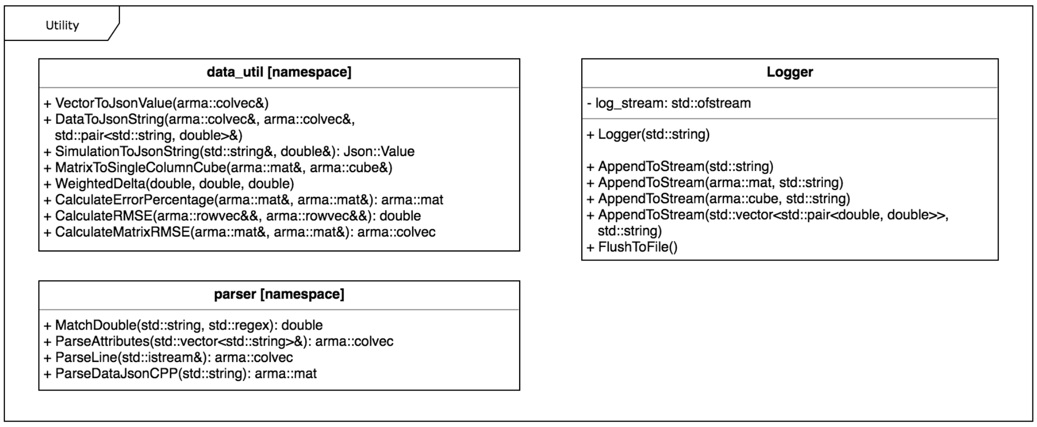

Utility
data_util [namespace]
+ VectorToJsonValue(arma::colvec&)
+ DataToJsonString(arma::colvec&, arma::colvec&, std::pair<std::string, double>&)
+ SimulationToJsonString(std::string&, double&): Json::Value
+ MatrixToSingleColumnCube(arma::mat&, arma::cube&)
+ WeightedDelta(double, double, double)
+ CalculateErrorPercentage(arma::mat&, arma::mat&): arma::mat
+ CalculateRMSE(arma::rowvec&&, arma::rowvec&&): double
+ CalculateMatrixRMSE(arma::mat&, arma::mat&): arma::colvec
Logger
- log_stream: std::ofstream
+ Logger(std::string)
+ AppendToStream(std::string)
+ AppendToStream(arma::mat, std::string)
+ AppendToStream(arma::cube, std::string)
+ AppendToStream(std::vector<std::pair<double, double>>, std::string)
+ FlushToFile()
parser [namespace]
+ MatchDouble(std::string, std::regex): double
+ ParseAttributes(std::vector<std::string>&): arma::colvec
+ ParseLine(std::istream&): arma::colvec
+ ParseDataJsonCPP(std::string): arma::mat

Figure B.5: Illustration of the system decomposition of the utility package.

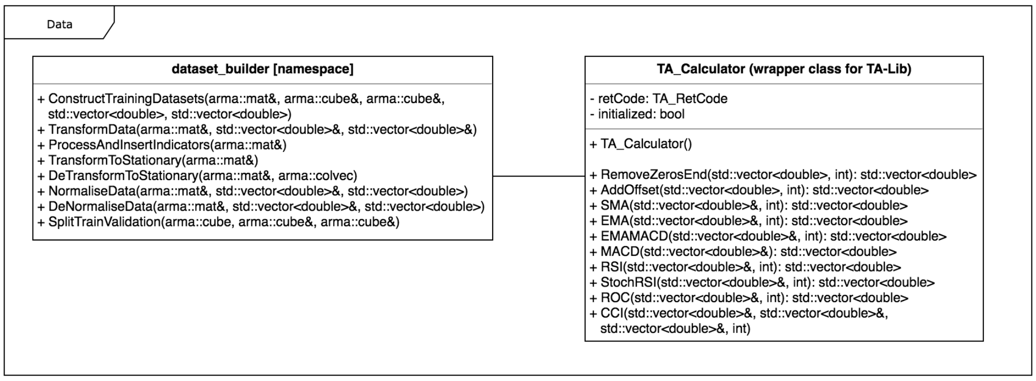
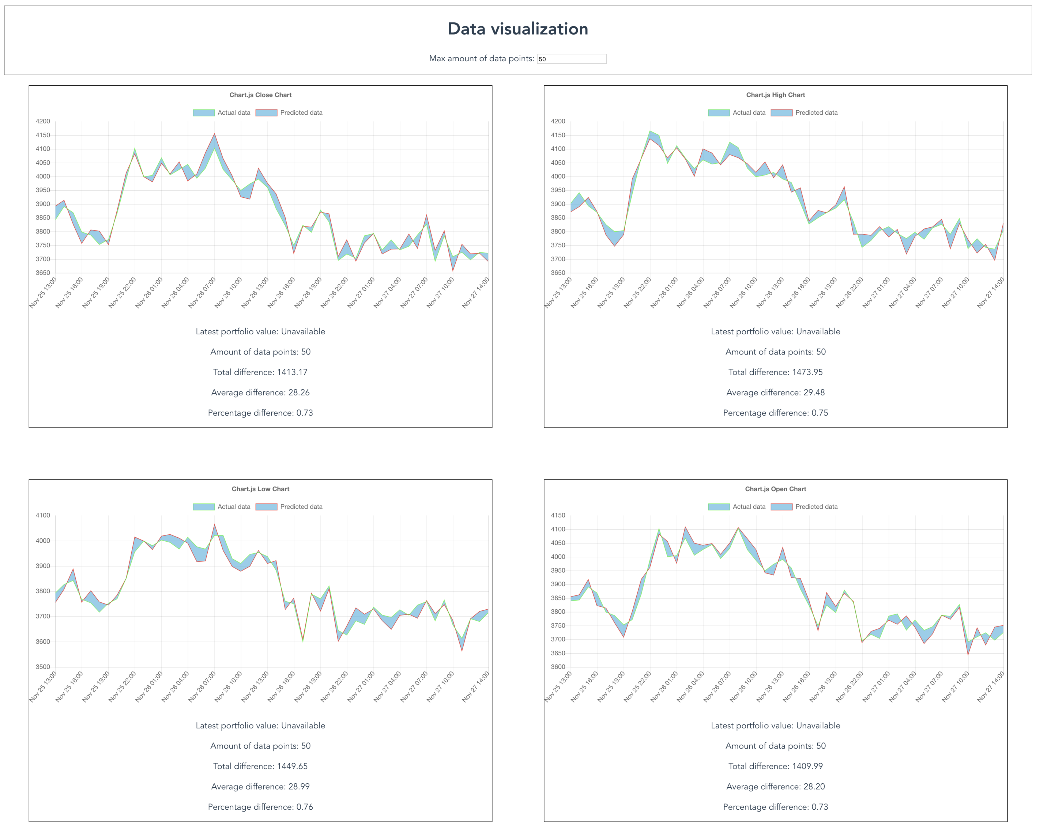

Data
dataset_builder [namespace]
+ ConstructTrainingDatasets(arma::mat&, arma::cube&, arma::cube&, std::vector<double>, std::vector<double>)
+ TransformData(arma::mat&, std::vector<double>&, std::vector<double>&)
+ ProcessAndInsertIndicators(arma::mat&)
+ TransformToStationary(arma::mat&)
+ DeTransformToStationary(arma::mat&, arma::colvec)
+ NormaliseData(arma::mat&, std::vector<double>&, std::vector<double>)
+ DeNormaliseData(arma::mat&, std::vector<double>&, std::vector<double>)
+ SplitTrainValidation(arma::cube, arma::cube&, arma::cube&)
TA_Calculator (wrapper class for TA-Lib)
- retCode: TA_RetCode
- initialized: bool
+ TA_Calculator()
+ RemoveZerosEnd(std::vector<double>, int): std::vector<double>
+ AddOffset(std::vector<double>, int): std::vector<double>
+ SMA(std::vector<double>&, int): std::vector<double>
+ EMA(std::vector<double>&, int): std::vector<double>
+ EMAMACD(std::vector<double>&, int): std::vector<double>
+ MACD(std::vector<double>&): std::vector<double>
+ RSI(std::vector<double>&, int): std::vector<double>
+ StochRSI(std::vector<double>&, int): std::vector<double>
+ ROC(std::vector<double>&, int): std::vector<double>
+ CCI(std::vector<double>&, std::vector<double>&, std::vector<double>&, int)

Figure B.6: Illustration of the system decomposition of the data package.

C

Data-visualizer



Data visualization

Max amount of data points: 50

Chart.js Close Chart

Latest portfolio value: Unavailable

Amount of data points: 50

Total difference: 1413.17

Average difference: 28.26

Percentage difference: 0.73

Chart.js High Chart

Latest portfolio value: Unavailable

Amount of data points: 50

Total difference: 1473.95

Average difference: 29.48

Percentage difference: 0.75

Chart.js Low Chart

Latest portfolio value: Unavailable

Amount of data points: 50

Total difference: 1449.65

Average difference: 28.99

Percentage difference: 0.76

Chart.js Open Chart

Latest portfolio value: Unavailable

Amount of data points: 50

Total difference: 1409.99

Average difference: 28.20

Percentage difference: 0.73

Figure C.1: Screenshot of the data-visualizer tool

D

Research Proposal

Exploring the capabilities of AI in digital asset trading

Ashay Somai, Ralph van Gurp, Jasper Hu and Hugo Kooijman

Bachelor Thesis Project - Delft University of Technology

1. INTRODUCTION

Financial markets have historically been a place of interest for those looking to expand their wealth. As a result, these markets have been extensively studied in an effort to discover the rationale and patterns behind their functioning. Lately, machine learning based prediction methods have been winning researchers' interest [28, 18, 43] as machine learning, in general, has become increasingly popular. While some early research was done in the context of (relatively simple) artificial neural networks with good results [34, 29, 45], the increasing availability of computing power has recently lead to the development of even more powerful and promising deep neural networks.

Simultaneously, the inception of cryptocurrencies and digital assets has introduced the notion of crypto exchanges and reinvented the financial markets. However, prediction of digital assets trends is relatively understudied, especially in the context of deep neural networks.

1.1. Research question

Can an RNN-based trading engine utilising trading indicators maintain a stable monetary value of a digital asset in a live market environment?

1.2. Design goals

Two general requirements carry significant weight in the design and choice of components. Primarily, the predictions made by the systems must be sufficiently accurate. That is; they must approximate the desired values within a reasonable margin, and do so consistently. Secondarily, the time required to make a prediction must be minimised where possible, considering the rapid dynamic of financial markets; milliseconds could cost millions.

In addition to delivering accurate results quickly, the system should be reliable. The system will have to be extensively tested and validated to ensure safety and robustness.

Lastly, the system shall be developed with a large degree of modularity. It should be easy to extend the application to cover different setups, such as introducing new sets of digital assets and exchanges or configuring the network to include or exclude specific outputs.

The next section describes an underlying implementation of the intended software product with a focus on the predictive aspects of the system and proposes extended functionality.

1.3. Requirements

Both the product owners and our team determined a set of functional requirements that the trading engine should possess. The Must Haves are features that at the very least have to be fulfilled before any other others. Should Haves are features that we expect the bot to have at the end of the project period. Moreover, Could Haves are features that might be added should there still be time left.

1.3.1. Must Haves

The trading engine

- must work in a LINUX environment.

- must mainly be written in C++.
- must work in a live trading environment.
- must be modular and scalable for future development.
- must be able to place buy and sell orders, with at least one pair of currencies (e.g. BTC-USD).

1.3.2. Should Haves
The trading engine

- should be able to test its performance with historical datasets.
- should be able to make decisions based on at least three (3) financial market indicators.
- should make one (1) decision per hour time frame whether to HOLD, BUY or SELL.
- should have a backup function, whereas the order is not triggered due to an unexpected fall/rise of the price, to recreate the order in the new circumstances.

1.3.3. Could Haves
The trading engine

- could have paper trading possibilities in a live trading environment.
- could be considered successful if it can maintain the starting monetary value with a lower limit of -10%.

1.4. System design

The approach that we are taking is, using a Recurrent Neural Network, or in short RNN. This type of neural network has the property that it feeds the information back to its hidden layer. Since it can use the previous state to help to make a decision for the next state, we find it a fitting solution to the prediction of the stock prices.

Next, we have chosen to use one hidden layer. Although many papers use many hidden layers and thus also a large amount of hidden neurons [27, 25, 49], one paper supports our view [39]. Our view is that digital assets have a higher level of fluctuation than traditional stocks, and thus that the chance of sudden unexpected movements is more significant in digital assets, compared to traditional stocks. Although using more hidden layers increases the accuracy of the prediction, according to the training data, its ability to predict a sudden movement is worse [39].

Furthermore, we have chosen to start with 16 hidden neurons. We have looked at the rule-of-thumb methods, mentioned in [37] to determine the number of hidden neurons. As mentioned before, other neural networks, predicting stock prices, use a more massive amount of hidden neurons, but the paper, supporting our point of view, which was mentioned before, only uses one hidden layer and 16 hidden neurons. The author of that paper, [39], found out that using that configuration, it also could predict the unexpected price movements pretty well. Therefore, we have decided to take that as a baseline and will optimise it in a later stage.

The 13 input neurons of the neural network consist of the price data, volume, and indicators. These different kinds of data are described in chapter 2.3. The input neurons are enumerated below:

- Open
- Close
- High
- Low
- Trading Volume
- SMA
- EMA
- MACD
- RSI
- SRSI
- ROC
- CCI

The 5 output neurons, will consist of the predicted price data and the volume. This results in the following list:

- Open
- Close
- High
- Low
- Trading Volume

1.5. Validation

Cross validation The main task of the trading engine is to minimise potential losses. Therefore our baseline validation tool will be the monetary value of a digital asset at a certain point in time, compared to the value of the assets after a period of trading. Since the course of an asset may inherently bear an unpredictable loss, a second baseline validation will be the course of the asset itself.

As a secondary set of validation baselines, we aim to compare the developed trading engine to existing systems with similar functionality, such as Gekko [3]. Further validation can be done through training and prediction using historical data.

1.6. Project Planning

The Bachelor Project consists of three phases [2]. The first phase is the research phase, the second phase is the actual development of the software, and the final phase is the completion phase. Following is the breakdown of the planning of the next weeks:

- **Research Phase**(week 1-2)
  In this phase, we will start with the research on this project. The goal is to gain a deeper understanding of the problem along with learning methods to solve the problem. The research phase is concluded with a report containing the research done along with an outline of the developed product and a research question.
- **Development Phase**(week 3-8)
  This phase contains the actual software development, and the product is constructed according to the requirements set beforehand. Then the implementation will be done along with the testing. Since neural networks may take much time to crunch all the data and be fully trained, this will be initiated in week 5, so there will be enough time to evaluate the results and make changes when needed.
- **Completion Phase**(week 9-10)
  In the completion phase, the attention will be set on several things. The code has to be evaluated by SIG and changes to the code have to be made according to the report SIG delivers. Furthermore, the final report has to be written. In this report, the whole process of the project is explained along with the results and discussion. The final thing in the completion phase is the presentation. Here we will present the final report and talk about the project in general.

The following chapters will further define the context of the research project and motivate design choices related system design, choice of framework, and the research question.

2. FINANCIAL MARKET

Financial markets have existed for a long time, and the stock market has been the primary market of it. However, the white paper of Satoshi Nakamoto has introduced a new pillar in this financial market. The Bitcoin, which was introduced in the white paper by the author mentioned above, was released in 2008 [35], and it has been operational since 2009. Consequently, exchanges opened where this ’new’ digital asset could be traded. These exchanges have evolved to the level, where it contained $831,871,000,000 on 07 January 2018 [4]. This money is traded on the exchanges. The traders, which are actively trading on the market, may earn a profit, by analysing the market, followed by buying and selling the assets at the right time. Since the AI also has to trade in the market of digital assets, this chapter will give a more thorough explanation and our choices of the tools that the AI will use.

2.1. Market analysis

To be able to analyse the market, it is required to understand what is happening, and how the current price has been established. Subsequently, the two main types of analyses, that are used in analysing the market are fundamental analysis and technical analysis.

2.1.1. Fundamental analysis

Fundamental analysis is the observation and analysis of the fundamentals of an asset. Instead of just looking of the graphs, you analyse the other streams of information and thus are digging deeper into the asset itself. Examples of a stream of information could be the releases of new products, news, financial statements, etc [44]. When analysing these streams of information, it is usual to extract several kinds of intelligence, such as profit and growth potential, relative riskiness, and finally, if the asset is under-, over-, or fairly valued in the current market. Evaluating the fundamentals of an asset is mostly used to compare it with itself, competitors, or the broader market [7].

2.1.2. Technical analysis

Whereas the fundamental analysis emphasises on analysing the fundamentals of an asset to extract the intrinsic value, technical analysis focuses on analysing the price, volume, and the trend [31]. Using a more elaborate explanation, technical analysis focuses on patterns of price movements, trading signals and a variety of other analytic tools to evaluate an asset's strength or weakness [8].

2.1.3. Combination

When a choice is made, to only choose to use fundamental analysis and not to use technical analysis or the other way around. There is much extra unused information, of which you also could have extracted information. To get the advantage over other people and making the decision just a bit earlier than the mass, results in a profit for the trader. So that is the reason that conventionally both fundamental and technical analysis is used to analyse an asset. Both approaches have their advantages, so using both could give you that small advantage compared to other people [33].

The time of this project is limited, and the AI around natural language processing is not advanced enough yet, to easily and accurately implement a processing algorithm, for example, news articles [48]. Thus we have chosen to start with solely implementing technical analysis in our AI trading engine. If there is enough time to implement a processing algorithm for fundamental analysis, we will have a look at that as well.

2.2. Trading tools

When we look at the financial market, there are several tools which can assist you with making a decision regarding buying or selling stock. There are indicators, which are metrics to manipulate and visualise the data available using a mathematical formula. When you have decided to buy or sell an asset, an order has to be placed. There are different kinds of orders, which each have their usability.

2.2.1. Indicators

Technical indicators use historical trading data such as price, volume and open interest to analyse short-term price movements and are mostly used by active users [9]. There are thousands of different indicators available, and traders create a combination what works best for them. Different combinations yield different results and finding a combination that suits the traders' strategy varies a lot. A selection of indicators [11] that will be included is:

- Simple Moving Average (SMA)
  The average price of a given time period, with equal weighting given to the price of each period.
- Exponential Moving Average (EMA)
  The average price of a given time period, with more weight given on recent prices.
- Moving Average Convergence/Divergence (MACD)
  Based on the differences between two moving averages of different lengths, a fast and slow moving average, called the trend. This is compared to a moving average of the MACD itself, called the signal line. If the MACD falls below the signal line, a 'sell' should be considered and vice versa.
- Relative Strength Indicator (RSI)
  The RSI represents the current price relative to preceding prices of a given time period.
- Stochastic Relative Strength Indicator (SRSI)
  Indicates whether an RSI value is oversold or overbought. Especially useful when the RSI is confined within the typical signal

levels of 20 and 80.

- Rate of Change (ROC)
  Compares the current price to the price of n periods ago. The current price is divided by the previous price and is represented as a percentage.
- Commodity Channel Index (CCI)
  Determines whether an asset is in the condition of being oversold or overbought. It compares the current mean price with the average mean price over a given time period.

2.2.2. Different kinds of orders

When the decision is made for which asset to buy, or to sell, an order has to be placed. Just using the buy or sell button may cause slippage. Slippage is the difference of the price what you expect and the price at which the trade is filed. Slippage can be substantial and may be the difference between winning or losing a trade. Certain order types allow you to specify the exact price and thus to minimise the slippage [5]. The most common order types are:

- Market Order
  The most basic type of order. It is used for buying or selling at the best available price. If there is enough liquidity, this is executed immediately.
- Limit Order
  An order to buy or sell at a specified price, and prevents negative slippage. However the execution is not guaranteed, the trade will only be filled if the price reaches the specified price.
- Stop Order
  A Stop Order is used to trigger a Market Order or Limit Order once a specified price has been reached. It is useful when it is important to know whether the price is rising or declining, by placing the Stop Order above or below the current price.

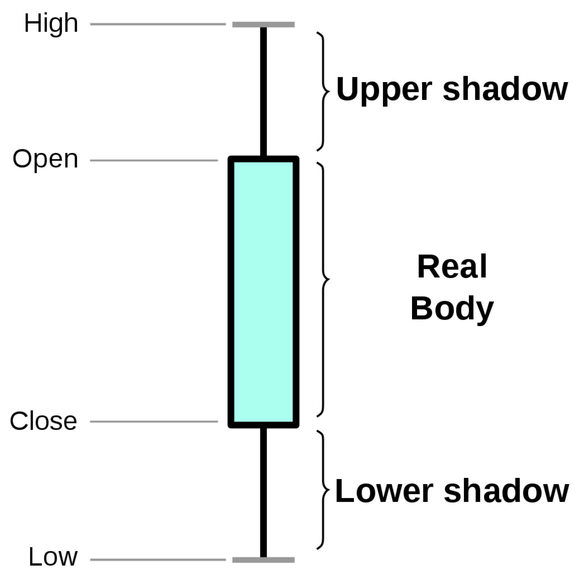


FIGURE 1. Visualisation of a candlestick, showing the different components, adopted from Wikipedia, the free encyclopedia [1].

- Trailing Stop Order
  A sell Trailing Stop sets the stop price at a fixed amount below the current price with a trailing amount. When the market price rises, the stop price moves along with the rise, but when the market prices decline, the stop price is not changed. When the market price equals the stop price, the asset is sold. For a buy Trailing Stop the opposite is done, so the stop price is set above the current price and moves along with declines.

These are the types of orders that will be included. Each of them has their own merits and combining this with the previously mentioned indicators, having the choice to select different indicators and orders can result in different strategies each user can use, thus creating a wide variety of strategies with each having their own benefits.

2.3. Different kinds of data

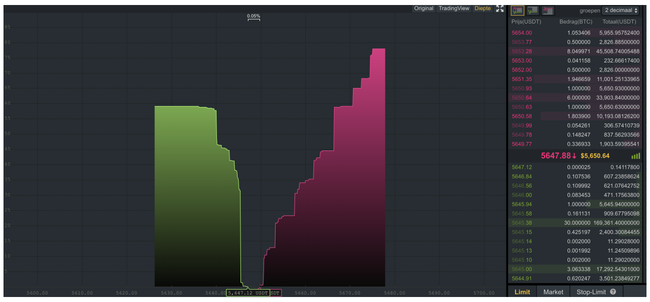
In the trading setting, there are different kinds of data streams. The primary data streams are price, volume, and order book data. Each of these data streams provides intelligence to the trader. However, the price data is used the most. Almost all indicators solely utilise the price data to extract new forms of intelligence, such as buy and sell signals. Also, when combining a signal of an indicator with another signal of another information stream, it could give an extra confirmation. In the following sections, the three primary varieties of data, and our choice of data, used in the engine, are explained.

2.3.1. Price

When an ask and demand order match, an equilibrium price arises and these equilibria prices over a specific period form the basis of a candlestick [6]. An example of a candlestick can be found in Figure 1 on page 5. This price data is usually visualised in a candlestick chart, where a candlestick is a snapshot of 4 pieces of information at a specific time frame:

- High
  This is the highest price that the candlestick has reached in the specific time frame. Conventionally, this is displayed as the upper shadow of the candlestick.
- Low
  This is the lowest price that the candlestick has reached in the specific time frame. Conventionally, this is displayed as the lower shadow of the candlestick.
- Open
  This is the price at the opening time of the candlestick in the specific time frame. Conventionally, this is displayed at the top of the body of the candlestick.
- Close
  This is the price at the ending time of the candlestick in the specific time frame. Conventionally, this is displayed at the bottom of the body of the candlestick.

This data is mainly collected in the form of candlestick data, and since candlesticks are relative to a specific period, the data also differ, when you choose other time frames for a candlestick.



Prijs(USDT)	Bedrag(BTC)	Totaal(USDT)
5654.00	1.053406	5,955.95752400
5653.77	0.500000	2,826.88500000
5653.28	8.049971	45,508.74005488
5653.00	0.041158	232.66617400
5652.00	0.500000	2,826.00000000
5651.35	1.946659	11,001.25133965
5650.93	1.000000	5,650.93000000
5650.64	6.000000	33,903.84000000
5650.63	1.000000	5,650.63000000
5650.58	1.803900	10,193.08126200
5649.99	0.054261	306.57410739
5649.78	0.148247	837.56293566
5649.77	0.336933	1,903.59395541
5647.88↓	$5,650.64	
5647.12	0.000025	0.14117800
5646.84	0.107536	607.23858624
5646.56	0.109992	621.07642752
5646.00	0.083453	471.17563800
5645.94	1.000000	5,645.94000000
5645.58	0.161131	909.67795098
5645.38	30.000000	169,361.40000000
5645.15	0.425197	2,400.30084455
5645.14	0.002000	11.29028000
5645.13	0.001992	11.24509896
5645.10	0.002000	11.29020000
5645.00	3.063338	17,292.54301000
5644.91	0.620247	3,501.23849277

FIGURE 2. Visualisation of an order book, including the depth chart.

2.3.2. Trading volume

Like a candlestick, the volume is also subjective to a specific period. The volume is the monetary value of the occurred trades in a specific time. Also, usually, it is associated with the candlestick of the same time frame. Furthermore, the volume data is used to measure the relative strength of a market move [10].

2.3.3. Order book (market depth)

The order book or market depth is the list of all buy and sell orders for a specific asset, sorted by price. Subsequently, the order book is used to see the balance of price and orders. When there is an imbalance in orders, it shows the spread and will most likely give hints about the direction of the price of the asset [12]. An example of an order book is given in Figure 2 on page 6.

The choice for the thesis project is to start with using the price data and try to find patterns in the price data, in combination with the indicators. Afterwards, we can feed additional data, such as volume and order book data, to verify the predicted price of the AI. Also, financial data is scarcely available, and the price data is the easiest to fetch.

3. ARTIFICIAL INTELLIGENCE

One of the primary objectives of this research is to predict the course of digital assets successfully. That is, to discover the rules and patterns embedded in financial markets. Human traders often base their predictions on a myriad of indicators [31], as described in the previous chapter. Conventional algorithms, although capable of making such predictions suffer from an inability to adjust their predictions based on new information like a human expert would. We, therefore, propose implementing a form of artificial intelligence, or AI [42].

The term artificial intelligence is commonly used to describe the ability embedded in automated systems to make autonomous decisions. This decision-making process is usually fuelled by information received from the environment and may develop over time according to a learning process. Numerous forms of AI exist with varying applications, ranging from genetic algorithms [23] that evolve through trial-and-error, to rule discovery through Bayesian networks [40], and expert systems [46]. The next section introduces the concept of machine learning; a subset of AI techniques popularly used in pattern recognition and classification tasks [36].

3.1. Machine Learning

ML techniques employ statistical methods [15, 36, 38] to improve their performance after observing an input and (usually) an associated output. The system adjusts as more data is observed, thereby achieving a learning process without the need for explicit programming. Typically, ML systems are initialised with random values and subsequently trained using a training dataset. The training process can be supervised, semi-supervised or unsupervised depending on whether the training input contains a classification of the desired output or not. Generally, supervised learning aims to discover a mapping between input and output, while unsupervised learning aims to discover hidden patterns in data. One especially powerful technique that has gained traction in recent years is the artificial neural network [47].

The Artificial Neural Network, or ANN, is an interconnected system of nodes, each of which may be tailored to a specific task or implement a specific algorithm. These nodes loosely represent the neurons of the biological brain. Each layer processes the input received from the previous layer of neurons and outputs the result to the next layer. Complex hierarchies of nodes allow ANNs to identify different features of the original input. A dynamic set of weights allows the network to adjust the relevance of certain features in order to approximate the desired output. Neural networks composed of many layers and neurons are collectively known as Deep Learning (DL) techniques. DL has achieved significant results in many fields [20], rivalling or outperforming human experts [41, 21].

3.2. Deep Learning

As mentioned, deep learning networks are composed of many successive layers of processing units [32]. A large number of hierarchical layers allows the network to distinguish multiple levels of abstractions. Varying the number of layers and layer sizes can provide different degrees of abstraction [14].

Globally, deep neural networks can be further divided into two categories; convolutional neural networks (CNN) and recurrent neural networks (RNN). The main difference between these two forms of networks can be simplified as follows: CNNs recognise components across space (e.g. pixels of an image that make up a car) [30]. RNNs instead recognise patterns across time, such as in speech analysis [24] where previous words partially define the meaning of those that follow. Although either method can be used to solve many problems interchangeably, RNNs allow for some measure of memory not present in CNNs.

Long-Short Term Memory (LSTM) is another popular machine learning implementation with similar properties to the RNN. In RNNs the gradient function decays exponentially over time, thereby minimising the impact of data not in recent history. However, in the proposed application the increased importance of recent data is a desired property. Therefore, we opt to implement a recurrent neural network.

3.3. AI Frameworks

Numerous frameworks and libraries implementing functionality for neural networks and machine learning are currently in existence, some with very general specifics and others specialised for a set number of tasks. We have mostly explored some well-known and widely recommended frameworks in order to determine which one would best suit our goals. Due to efficiency requirements and personal preference, we sought to find a framework written in C++ or could be accessed through a C++ API.

3.3.1. Tensorflow

Tensorflow is the most used open source software library for high-performance calculation by data scientists worldwide. Developed by Google, it is used by big corporations such as eBay and NVIDIA. It is flexible and uses a high-level API, meaning it is compatible with numerous online tools and usable for a wide variety of applications [13].

However, such flexibility and extensibility cost performance, and due to our specific requirement of only machine learning software, we would not use the vast majority of this library.

3.3.2. Caffe

This deep learning framework is well known for its speed, transposability and modelling Convolution Neural Networks. Its biggest benefit comes from its ability to access the deep net repository "Caffe Model Zoo", where pretrained networks are stored and immediately usable for user applications. Caffe is primarily used in visual analysing data (such as image recognition systems) [26].

However, Caffe does not support the finely tweaking of network layers, making deep learning applications harder to build. Again, it is optimised for applications that have little to do with stock price prediction.

3.3.3. Scikit-Learn

Scikit-Learn is a well known open source library focused on machine learning. It is intended for small projects and for beginners to get experience with AI and focuses on simplicity and speed [16].

Sadly, Scikit-Learn does not support deep learning, something we are planning to use for accurate prediction. Furthermore, the library is entirely written in Python. During stock market trading, speed is of utmost importance, so the few milliseconds advantage that well written C++ code might offer over Python code could be essential in turning a profit.

3.3.4. MxNet

Written in C++, MxNet is a flexible and efficient deep learning library. It is known for its great scalability, so it's mainly used for handwriting recognition, forecasting, and neuro-linguistic programming [17].

It has a few downsides. It boasts a much smaller community than frameworks such as Tensorflow and is not well known within the research community.

3.3.5. MLPack

MLPack is another machine learning library written in C++. It is focused on speed, efficiency, and ease-of-use. The library's contents encompass mainly just the bare essentials, remaining fairly low-level and thus has the potential to offer low latency. Such is of great importance for our specific project [19].

The only problem is that it is relatively unknown and the documentation is minimal.

After some consideration, we chose to implement an RNN, because this type of neural network takes the factor of time into account, as should be done when dealing with fluctuating market prices of (digital) assets. To reduce latency, we selected to work with the MLPack library, as it contains just the functionality we need in a relatively low-level language. We consider the other more extensive libraries and frameworks as bloatware that would unnecessarily slow down the trading engine.

REFERENCES

[1] Candlestick chart - wikipedia. `https://en.wikipedia.org/wiki/`

Candlestick_chart. Accessed: 18 November 2018.

[2] Course browser searcher. https://studiegids.tudelft.nl/a101_displayCourse.do?course_id=45696. Accessed: 20 November 2018.

[3] Gekko. https://gekko.wizb.it/. Accessed: 20 November 2018.

[4] Global charts — coinmarketcap. https://coinmarketcap.com/charts/. Accessed: 20 November 2018.

[5] Introduction to order types. https://www.investopedia.com/university/intro-to-order-types/. Accessed: 16 November 2018.

[6] Investopedia - equilibrium. https://www.investopedia.com/terms/e/equilibrium.asp. Accessed: 16 November 2018.

[7] Investopedia - fundamentals. https://www.investopedia.com/terms/f/fundamentals.asp. Accessed: 16 November 2018.

[8] Investopedia - technical analysis. https://www.investopedia.com/exam-guide/series-3/studyguide/chapter5/technical-analysis.asp. Accessed: 16 November 2018.

[9] Investopedia - technical indicator. https://www.investopedia.com/terms/t/technicalindicator.asp. Accessed: 16 November 2018.

[10] Investopedia - volume. https://www.investopedia.com/terms/v/volume.asp. Accessed: 16 November 2018.

[11] List of technical indicators. https://www.tradingtechnologies.com/help/x-study/technical-indicator-definitions/list-of-technical-indicators/. Accessed: 16 November 2018.

[12] Order book definition — investopedia. https://www.investopedia.com/terms/o/order-book.asp. Accessed: 16 November 2018.

[13] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[16] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[17] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[18] Rohit Choudhry and Kumkum Garg. A hybrid machine learning system for stock market forecasting. *World Academy of Science, Engineering and Technology*, 39(3):315–318, 2008.

[19] Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong Zhang. mlpack

3: a fast, flexible machine learning library. *Journal of Open Source Software*, 3:726, 2018.

[20] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(34):197–387, 2014.

[21] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[22] Justin Fox and Alan Sklar. *The myth of the rational market: A history of risk, reward, and delusion on Wall Street.* Harper Business New York, 2009.

[23] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[24] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[25] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied soft computing*, 11(2):2510–2525, 2011.

[26] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[27] Ken-ichi Kamijo and Tetsuji Tanigawa. Stock price pattern recognition-a recurrent neural network approach. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 215–221. IEEE, 1990.

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

[29] Miroslaw Kordos and Andrzej Cwiok. A new approach to neural network based stock trading strategy. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 429–436. Springer, 2011.

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

[31] Ramon Lawrence. Using neural networks to forecast stock market prices. *University of Manitoba*, 333, 1997.

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

[33] Yu-Hon Lui and David Mole. The use of fundamental and technical analyses by foreign exchange dealers: Hong kong evidence. *Journal of International Money and Finance*, 17(3):535–545, 1998.

[34] Leonardo C Martinez, Diego N da Hora, Joao R de M Palotti, Wagner Meira, and Gisele L Pappa. From an artificial neural network to a stock market day-trading system: A case study on the bm&f bovespa. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 2006–2013. IEEE, 2009.

[35] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[36] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[37] Gaurang Panchal, Amit Ganatra, YP Kosta, and Devyani Panchal. Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers. *International Journal of Computer Theory and Engineering*, 3(2):332, 2011.

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

[39] Akhter Mohiuddin Rather, Arun Agarwal, and VN Sastry. Recurrent neural network and a hybrid model for prediction of stock returns. *Expert Systems with Applications*, 42(6):3234–3241, 2015.

[40] Christian Robert. Machine learning, a probabilistic perspective, 2014.

[41] Jon Russel. Googles alphago ai wins three-match series against the worlds best go player, 2017. [Online; accessed 16 November, 2018].

[42] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[43] Nicholas I Sapankevych and Ravi Sankar. Time series prediction using support vector machines: a survey. *IEEE Computational Intelligence Magazine*, 4(2), 2009.

[44] Jack D Schwager. *A complete guide to the futures markets: fundamental analysis, technical analysis, trading, spreads, and options.* John Wiley & Sons, 1984.

[45] Jian-Zhou Wang, Ju-Jie Wang, Zhe-George Zhang, and Shu-Po Guo. Forecasting stock indices with back propagation neural network. *Expert Systems with Applications*, 38(11):14346–14355, 2011.

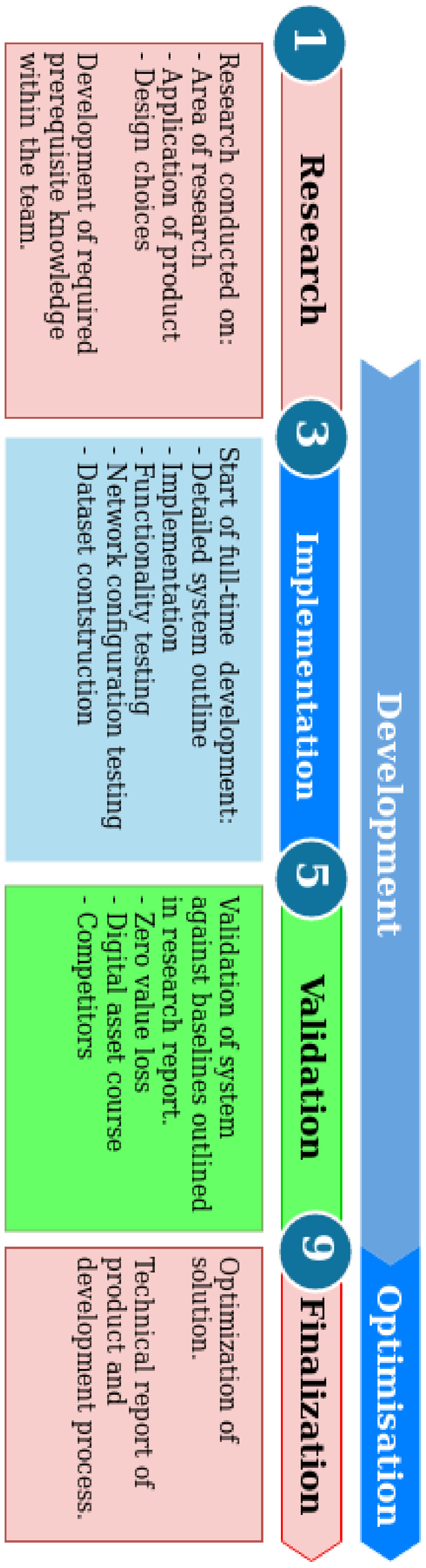
[46] Sholom M Weiss and Casimir A Kulikowski. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* Morgan Kaufmann Publishers Inc., 1991.

[47] B Yegnanarayana. *Artificial neural networks.* PHI Learning Pvt. Ltd., 2009.

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

[49] Yudong Zhang and Lenan Wu. Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network. *Expert systems with applications*, 36(5):8849–8854, 2009.

4. ATTACHMENTS



Research conducted on:
- Area of research
- Application of product
- Design choices

Development of required prerequisite knowledge within the team.

Start of full-time development:
- Detailed system outline
- Implementation
- Functionality testing
- Network configuration testing
- Dataset contstruction

Validation of system against baselines outlined in research report.
- Zero value loss
- Digital asset course
- Competitors

Optimization of solution.

Technical report of product and development process.

FIGURE 3. Project planning

Bibliography

[1] Futures definition| investopedia. https://www.investopedia.com/terms/f/futures.asp. Accessed: 25 January 2019.

[2] Order book definition | investopedia. https://www.investopedia.com/terms/o/order-book.asp. Accessed: 25 January 2019.

[3] Multiple time frames can multiply returns. https://www.investopedia.com/articles/trading/07/timeframes.asp. Accessed: 25 January 2019.

[4] Investopedia - volume. https://www.investopedia.com/terms/v/volume.asp. Accessed: 25 January 2019.

[5] The dsdm agile project framework (2014 onwards), Jun 2017. URL https://www.agilebusiness.org/content/moscow-prioritisation.

[6] Git flow, jan 2018. URL https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

[7] Google c++ style guide, jan 2018. URL https://google.github.io/styleguide/cppguide.html.

[8] Sanjoy Basu. Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The journal of Finance*, 32(3):663–682, 1977.

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

[10] Binance. Binacpp, March 2018. URL https://github.com/binance-exchange/binacpp.

[11] BusinessDictionary.com. Definition of stakeholder, January 2019. URL http://www.businessdictionary.com/definition/stakeholder.html.

[12] Peter Christoffersen and Denis Pelletier. Backtesting value-at-risk: A duration-based approach. *Journal of Financial Econometrics*, 2(1):84–108, 2004. doi: 10.1093/jjfinec/nbh004. URL http://dx.doi.org/10.1093/jjfinec/nbh004.

[13] Ryan R. Curtin, Marcus Edel, Mikhail Lozhnikov, Yannis Mentekidis, Sumedh Ghaisas, and Shangtong Zhang. mlpack 3: a fast, flexible machine learning library. *Journal of Open Source Software*, 3:726, 2018. doi: 10.21105/joss.00726. URL https://doi.org/10.21105/joss.00726.

[14] Marcus Edel. Fast lstm layer documentation, June 2018. URL https://github.com/mlpack/mlpack/blob/master/src/mlpack/methods/ann/layer/fast_lstm.hpp.

[15] Mario Fortier. Ta-lib, 2007. URL http://ta-lib.org/.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[17] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[18] Eric Han. Intra-exchange cryptocurrency arbitrage bot. 2018.

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

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

[21] NVIDIA. Nvblas, 2015. URL `https://docs.nvidia.com/cuda/nvblas/index.html`.

[22] open-source parsers. Jsoncpp, January 2019. URL `https://github.com/open-source-parsers/jsoncpp`.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[24] Conrad Sanderson. Armadillo c++ linear algebra library, June 2016. URL `https://doi.org/10.5281/zenodo.55251`.

[25] Jim Y.F. Yam and Tommy W.S. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1):219 – 232, 2000. ISSN 0925-2312. doi: https://doi.org/10.1016/S0925-2312(99)00127-7. URL `http://www.sciencedirect.com/science/article/pii/S0925231299001277`.