FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automatisierte Prognose der Entwicklung von Kryptowährungspreisen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Lukas Aumayr, BSc
Matrikelnummer 01325536

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Ao.Univ.Prof. Ing. Mag. Dr. Horst Eidenberger

Wien, 28. März 2019

Lukas Aumayr	Horst Eidenberger

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automated Prognosis of the Development of Cryptocurrency Prices

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Lukas Aumayr, BSc
Registration Number 01325536

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Ing. Mag. Dr. Horst Eidenberger

Vienna, 28th March, 2019

Lukas Aumayr	Horst Eidenberger

Erklärung zur Verfassung der Arbeit

Lukas Aumayr, BSc
Mahlergasse 4, 3100 St. Pölten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. März 2019

Lukas Aumayr

Danksagung

Zunächst möchte ich mich bei Herrn Ao.Univ.Prof. Dr. Horst Eidenberger für die Möglichkeit, an einem so spannenden Thema zu forschen, sowie für seine Ermutigung und hervorragende Betreuung bedanken.

Außerdem möchte ich meinen Eltern für ihre bedingungslose Liebe, Hilfe und Begleitung in meinem gesamten Leben meinen Dank aussprechen. Meine Erfolge sind auch ihre. Ich bin auch dankbar für die Hilfe meines Bruders und die ausführlichen Diskussionen, die wir zum Thema dieser Arbeit hatten. Zu guter Letzt gilt mein Dank meiner Freundin, meiner Schwester, meinem Schwager, meinen Großeltern, meiner restlichen Familie und meinen Freunden für ihre Unterstützung.

— *Danke.*

Acknowledgements

I want to start by thanking Professor Horst Eidenberger for giving me the opportunity to conduct research on such an exciting topic and in particular for his encouragement and excellent support.

Additionally I want to express my gratitude towards my parents, for their unconditional love, support and guidance throughout my life. My accomplishments are also theirs. I am also grateful for the help from my brother and the thorough discussions we had about the topic of this work. Finally, I am thankful for the encouragement I received from my girlfriend, my sister, my brother-in-law, my grandparents, my remaining family and all my friends.

— *Thank you.*

Kurzfassung

Seit der Etablierung von Bitcoin sind Kryptowährungen als alternative digitale Zahlungsmethode und hochspekulative Investition sehr gefragt. Mit dem Anstieg der Rechenleistung und dem Wachstum der verfügbaren Daten, hatten tiefe neuronale Netze in den letzten Jahren auch eine steigende Popularität zu verzeichnen. Mit der Einführung der Long Short-Term Memory (LSTM) Architektur wurden neuronale Netze effizienter darin, langfristige Abhängigkeiten in Daten wie Zeitreihen zu erkennen.

In dieser Arbeit kombinieren wir diese beiden Themen, indem wir neuronale Netze verwenden, um eine Prognose der Kryptowährungspreise zu generieren. Insbesondere testen wir, ob LSTM-basierte neuronale Netze profitable Handelssignale für die Kryptowährung Ethereum vorhersagen können. Wir experimentieren mit verschiedenen Vorverarbeitungstechniken und unterschiedlichen Targets, sowohl für die Regression des Preises als auch für die Klassifikation von Handelssignalen. Wir evaluieren zwei LSTM-basierte Netzwerke und einen Convolutional Neural Network (CNN) LSTM Hybrid. Die für das Lernen verwendeten Daten sind historische Ethereum Preisdaten im Minutentakt von August 2017 bis Dezember 2018. Wir messen die Leistung der Modelle durch Backtesting, wobei wir den Handel auf Basis der Vorhersagen der Modelle mit historischen Daten simulieren, die nicht für das Lernen verwendet wurden. Wir analysieren diese Performance und vergleichen sie mit der Buy-and-Hold Strategie. Diese Simulation wird über einen Bullenmarkt, einen Bärenmarkt und einen stagnierenden Zeitraum durchgeführt.

In der Auswertung finden wir das leistungsstärkste Target und identifizieren zwei Vorverarbeitungskombinationen, die für diese Aufgabe am besten geeignet sind. Wir kommen zu dem Schluss, dass der CNN LSTM Hybrid in der Lage ist, Handelssignale für Ethereum profitabel zu prognostizieren und die Buy-and-Hold-Strategie um etwa 30% übertrifft, während die Performance der beiden anderen Modelle eher entäuschend war.

Abstract

Since the introduction of Bitcoin, cryptocurrencies have become very attractive as an alternative digital payment method and a highly speculative investment. With the rise in computational power and the growth of available data, the artificial intelligence concept of deep neural networks had a surge of popularity over the last years as well. With the introduction of the long short-term memory (LSTM) architecture, neural networks became more efficient in understanding long-term dependencies in data such as time series.

In this thesis, we combine these two topics, by using neural networks to make a prognosis of cryptocurrency prices. In particular, we test if LSTM based neural networks can produce profitable trading signals for the cryptocurrency Ethereum. We experiment with different preprocessing techniques and different targets, both for price regression and trading signal classification. We evaluate two LSTM based networks and one convolutional neural network (CNN) LSTM hybrid. As data for training we use historical Ethereum price data in one-minute intervals from August 2017 to December 2018. We measure the performance of the models via backtesting, where we simulate trading on historic data not used for training based on the model's predictions. We analyze that performance and compare it with the buy and hold strategy. The simulation is carried out on bullish, bearish and stagnating time periods.

In the evaluation, we find the best performing target and pinpoint two preprocessing combinations that are most suitable for this task. We conclude that the CNN LSTM hybrid is capable of profitably forecasting trading signals for Ethereum, outperforming the buy and hold strategy by roughly 30%, while the performance of the other two models was disappointing.

Contents

CHAPTER 1

Introduction

This chapter gives an overview over the thesis. It starts with a motivation on why this topic was chosen. The aim of this work is discussed along with the questions this thesis strives to answer. The methodological approach is laid out and the subsequent structure of the thesis is presented.

1.1 Motivation

Deep neural networks are a subset of *artificial intelligence* and their concept is also commonly referred to as *deep learning*. The core idea behind neural networks was inspired by the brain and has been around for decades. Due to improved learning methods, increasing computational power and large datasets, deep learning has risen in popularity over the last years and is a field of high interest in today's computer science (Figure 1.1) [1]. Neural networks have had success in solving very complex tasks, long thought to be out of reach for computers, e.g. speech recognition [2], image/video classification [3][4], automated translation [5], text generation [6], self-driving cars [7] and many more.

A very notable accomplishment for neural networks was when the technology company *Google* introduced a system called *Duplex* at their developer conference *Google I/O* in 2018. Duplex can allegedly make phone calls to book appointments or reservations with no human input necessary, having a natural sounding voice as well as using and understanding nuances of the English language. In the future, many more applications are conceivable and deep learning might replace call centers for providing automated support, assist in elderly care, aid during surgeries and so on.

Another IT topic which has been receiving an increasing amount of attention since 2008 is everything related to the *blockchain* technology [1]. While a blockchain is merely a cryptographically hashed, linked list, it creates possibilities for various applications. One

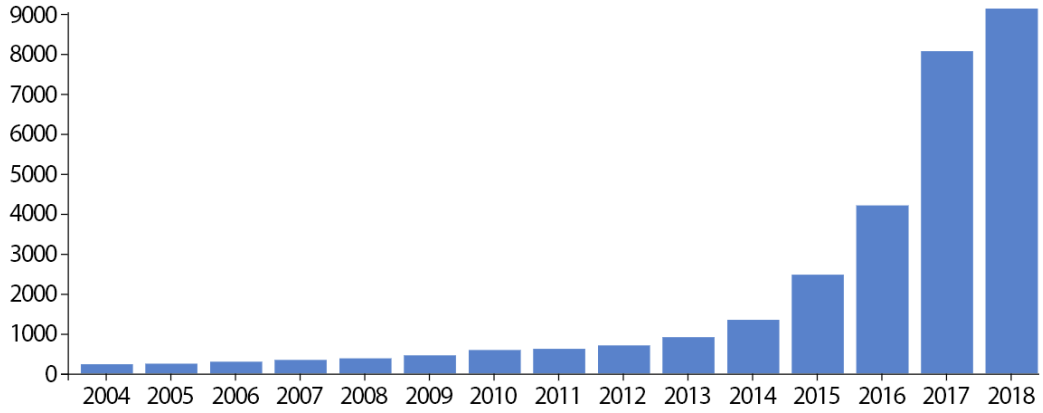

Figure 1.1: Number of scientific publications containing "deep learning", according to *Web of Science* [1].

of the many concepts arising from it are decentralized applications and *smart contracts* [8], agreements that are cryptographically enforced and therefore eliminating the need for notaries, first proposed by Szabo [9].

However one of the first and arguably the most popular use of blockchains was for cryptocurrencies, namely in *Bitcoin* as a public and tamper-proof ledger containing all transactions [10]. Cryptocurrencies have attracted many investors and have increased their market capitalization to over 100 billion US dollars. Aside from being a cryptographically tamper-proof form of digital payment, cryptocurrencies have several other advantages over conventional fiat currencies, such as not relying on a trusted third party for carrying out transactions or being able to transfer money using pseudonyms.

Bitcoin and other cryptocurrencies are traded on exchanges for one another or for fiat currencies like US dollars. The price fluctuation for cryptocurrencies is generally rather high. It is not clear, where this volatility stems from, which factors influence the price in what way and if it is possible to predict these fluctuations. Analyzing the price of cryptocurrencies using deep neural networks is an interesting way to combine aspects of both these topics, to investigate the performance of neural networks in that domain and to find out if the prices of cryptocurrencies can be predicted and to what extent.

1.2 Aim of the work

This work revolves around the following question: Are simple neural networks able to predict the price of cryptocurrencies? Cryptocurrency prices along with prices of stocks seem to behave rather randomly. Predicting the price of such an asset seems very hard at best, with some claiming that is not possible at all. If it is possible, then what features

influence the price in what way? The prospect of profit is enough for banks, finance companies, ordinary people and also researchers to look into automated trading and develop or test various methods and techniques for predicting such prices or generating trading signals in order to get a profitable trading strategy as a result. There are in fact numerous approaches for doing that [11] ranging from performing purely statistical analysis or applying mathematical concepts [12] over implementing simple algorithms to trade based on key financial figures [13] up to creating self-learning machine learning models like neural networks [14].

The vast majority of these attempts has focused on stock price forecasting. Analyzing the prices of cryptocurrencies is however quite interesting as well. Cryptocurrencies are very volatile and their trading volume is high with more than 15% of the overall market capitalization on January 30, 2019 [15]. Furthermore, the free availability of application programming interfaces (*API*s) and the relatively low trading fees make cryptocurrencies advantageous for automated trading for practical reasons. Historic price data is freely available in very fine granularity, providing a good data set to train on in later steps.

The goal of this thesis was to implement several neural networks, train them on previously fetched and then preprocessed historical price data as well as measuring their performance by simulating trades based on the predictions made by the neural networks. Leaving aside the inherent benefit of potential profit, we focus on several questions in this work and hope to gain a better insight on the price development, its influences and the neural networks:

- Is it possible to forecast the price of cryptocurrencies or generate profitable trading signals based on historic price data and if yes, to what extent?
- Are trading strategies based on these predictions or trading signals able to generate profit, perform better as the price development or both? How does this performance vary in different market situations?
- Are the prices of cryptocurrencies influencing one another? I.e. does the Bitcoin price influence the price of *Ethereum*?
- Which neural networks perform better and which worse?

To answer these questions we developed the following approach, which we present in the next section.

1.3 Methodological approach

The main task of this thesis is the implementation of the tool for constructing, training and testing neural networks as well as gathering and preprocessing the data and the conducting of experiments to evaluate these neural networks in their performance. For the realization

the following steps were taken. At first some background and literature research was done, in order to gain deeper insights about relevant and later used techniques and methods. Fields of interest included time series analysis, the stock market, automated trading, cryptocurrencies, machine learning, artificial neural networks (especially recurrent and convolutional neural networks and long short-term memory units) and their evaluation.

Subsequently, a concept for a software tool to carry out the goal of this thesis was drafted. That tool was to include means to aggregate historic cryptocurrency price data, preprocess it, create neural networks and train them on the data as well as evaluate the neural networks. In this step we settled on the type of data which was going to be used for training. The input features and four different targets were defined. The specific preprocessing techniques were chosen. The architecture of three different neural networks was picked. Finally a way to measure all different combinations of preprocessing techniques, targets and neural networks was selected, i.e. to simulate trading on the price data over a period of time.

Afterwards a suitable technology stack was selected, catering to the specific requirements of this task. The tool was constructed by implementing the different parts of the program in a modular fashion, so that the different targets, preprocessing techniques and neural network models can be combined at ease. The final tool works as follows. At first the data is gathered via the API of the cryptocurrency exchange *Binance*. For this dataset features are extracted and a target is constructed. The data is then preprocessed before it is fed into one of the priorly constructed neural networks. The neural network is trained on training data, a subset of all the available data, and then tested. This testing is done by simulating trading on the testing data, another subset disjoint to the first one. The trading is simulated by acting according to the prediction or generated trading signal of the neural network.

Furthermore, a plan for how to conduct the experiments was created. Ethereum was selected as a cryptocurrency for the evaluation. To have more conclusive results, three different price sectors were chosen on which to carry out the tests. We chose one sector for a period of time where the price was rising, another one for where the price was dropping and the last one for where it was staying more or less the same. Because of the vast search space along with the high computational effort, the search space was reduced in a way to focus on the most promising and fastest learning combinations. The tests were finally carried out and analyzed, focusing on the preprocessing techniques, targets and the three different neural network models. For the two best performing combinations, more extensive tests were carried out, i.e. they were trained across more epochs. The results were then compared to methods used in other studies.

In the end the work was summarized and some reflection was done on what conclusions can be drawn from it. Then final thoughts were given on how to continue and improve on this work and its results in possible future work.

1.4 Structure of the work

The structure of the thesis mirrors these steps and the order in which they were carried out.

- **Chapter 2 (Background)** gives a literature overview over the stock market, (automated) stock price prediction, cryptocurrencies, neural networks and their evaluation in order to provide background information on the concepts used throughout the rest of the thesis. Some related work is presented.
- **Chapter 3 (Constructing neural networks for predicting cryptocurrency prices)** goes over the concept of the software tool created in this thesis, considerations of different input data and features, output and preprocessing as well as various neural networks and performance measurement.
- **Chapter 4 (Implementation)** presents what technology was used and why it was selected, how the tool was implemented in code and goes over considerations about hardware and computation time.
- **Chapter 5 (Evaluation)** gives an overview over how the experiments and the evaluation were conducted, goes over the results in detail and compares the results to other methods.

The thesis is summarized in the end, where we also suggest some improvements to our approach and encourage further research.

CHAPTER 2

Background

In this chapter we present background information about the methods used throughout this thesis. Section 2.1 focuses on time series analysis as well as on influences and methods for modeling and forecasting stock prices. In Section 2.2 the fundamentals of the cryptocurrency Bitcoin are laid out, what differentiates it from conventional money, present alternative cryptocurrencies and the differences in trading cryptocurrencies and stocks. Section 2.3 lists numerous machine learning techniques, goes into how artificial neural networks function and learn, especially recurrent neural networks, long short-term memory units and convolutional neural networks. In Section 2.4 we go over how the performance of neural networks can be evaluated. Section 2.5 presents related work.

2.1 Stock analysis

The stock of a business is the sum of all shares of a joint-stock company. Every shareholder is a partial owner of the company in proportion to the amount of shares owned. Shares can be bought and sold on the stock market, which in turn determines the price of the company stock. Stocks are a popular form of investment for reasons such as profit, diversification of one's wealth, securing against inflation and so on. Private and professional investors are trading with stocks and a lot of research and money is spent on analyzing the price development in order to have an advantage over other investors and a more profitable stock portfolio. The following sections give an overview over different approaches for analyzing stock prices.

2.1.1 Time series analysis

Time series can be defined as follows.

> *"A time series is a set of observations x_t, each one being recorded at a specific time t. A discrete-time time series (...) is one in which the set T_0 of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals. Continuous-time time series are obtained when observations are recorded continuously over some time interval, e.g., when $T_0 = [0, 1]$."*
>
> — Peter J. Brockwell et al., *Introduction to time series and forecasting (pp. 1-2) [16]*

Due to the nature of the thesis we will only consider discrete time series. Many signals can be classified as time series. Some examples are data over time about the weather, tourists, users of public transport, shipment numbers and many more. Especially interesting are prices of any kind, e.g. housing prices, air fares, prices for specific products.

Analyzing such time series is carried out to gain insights into the data itself or to generate predictions based on the observations. Finding such a connection within the data or finding a model that can make predictions with a higher than average accuracy is beneficial and has many practical applications in the example areas mentioned above.

There are numerous approaches to time series analysis. Some of them are purely mathematical, e.g. regression, moving averages, maximum likelihood estimation, autoregressive moving average model and many more [16][17]. Another approach would be to use machine learning techniques, like for instance neural networks [18][19].

Stock and commodity prices over time are perfect examples of time series. Stock trades do not happen at fixed intervals, but can be carried out at any time. However, stock price charts are still usually presented in the following way. The price and sometimes other values like volume or number of trades are presented at fixed intervals. For instance, this can be yearly, daily or in second intervals. Out of all the trades that are carried out within this year, day or second, the value for this interval is quantified as price at the beginning of the interval (open), highest price of the interval (high), lowest price of the interval (low) and price at the end of the interval (close). These data are often graphically represented in the form of candlestick diagrams.

2.1.2 Unpredictability of the market

There has been extensive research attention on the analysis of stock price development. While analyzing time series can have various different applications, analyzing stock prices has the inherent benefit of potential profit. Additional interest is sparked by the curiosity for understanding stock price movements, mitigating the risks of major market crashes and so on. It is disputed whether or not predicting stock prices or their movements is possible or not. The models presented in this section conclude that such a price prediction is not possible over longer time spans.

In 1900 Bachelier famously modeled price movements of Parisian stocks after the physical phenomenon known as *Brownian motion*, which is used to describe random movements of particles in liquids and gases, to model small price movements of stocks traded in Paris [20]. A stochastic model for Brownian motions was proposed in 1923 by Wiener and is known as *Wiener process* [21]. We use the definition from Dunbar [22], which is included here for the reader's convenience.

Definition 2.1.1. The standard Wiener process is a stochastic process $W(t)$, for $t \geq 0$, with the following properties:

1. Every increment $W(t) - W(s)$ over an interval of length $t - s$ is normally distributed with mean 0 and variance $t - s$, that is $W(t) - W(s) \sim N(0, t - s)$.
2. For every pair of disjoint time intervals $[t_1, t_2]$ and $[t_3, t_4]$, with $t_1 < t_2 \leq t_3 < t_4$, the increments $W(t_4) - W(t_3)$ and $W(t_2) - W(t_1)$ are independent random variables with distributions given as in part 1, and similarly for n disjoint time intervals where n is an arbitrary positive integer.
3. $W(0) = 0$.
4. $W(t)$ is continuous for all t.

— Steven R. Dunbar, *Stochastic Processes and Advanced Mathematical Finance: The Definition of Brownian Motion and the Wiener Process (p. 4) [22]*

The independent increments described in property 2 mean that additional knowledge of previous values has no impact on the probability of future values. In fact, the above definition directly implies the *Markov property*, making Wiener processes a subcategory of Markov processes [22]. First-order Markov processes are *memoryless*, which means that the next state depends only on the current state and not a history of states before that [23]. If Wiener processes are a valid model for stock prices, it is implied that historic data, for example price data, cannot have any influence on the future price.

The *efficient-market hypothesis* (EMH), often attributed to Eugene Fama, proposes that all the available data or information of a stock is already fully reflected in a stock's price [24] [25]. This is based on the assumption that the information is freely available and that traders act largely rationally. According to this theory, stocks are never sold too cheap and bought too expensive. A trader cannot expect to outperform (or underperform) the average market performance, no matter what analysis or prediction method is used.

A theory consistent with the EMH is the *random walk hypothesis* [26]. It states that stock prices follow the mathematical model known as *random walk*, implying in essence that price changes are completely random. Like the EMH, the random walk hypothesis

concludes that any analysis based on historic data is futile. The following famous quote is found in the 1973 book "A random walk down Wall Street" written by Burton G. Malkiel.

> *"A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts."*
>
> — Burton G. Malkiel, *A random walk down Wall Street (p. 24) [27]*

These models are contested and there are claims that it is possible to predict stock prices, at least to some degree. We will present counter-arguments and possible influences on stock prices in the next section.

2.1.3 Influences on the market

Professional investors as well as banks that invest a lot of money in managers for managed funds, market analysis and so on make it apparent that there are at least numerous people that believe in being able to outperform the market. The following are empirical findings as well as other considerations that are in contrast to the theories in Section 2.1.2.

Analyzing the stock market over the years has resulted in finding several anomalies. An example for such an anomaly is the *January effect*, an effect where stock prices perform better in January compared to other months [28], shown in Figure 2.1. This effect is sometimes attributed to selling one's assets at the end of the year and rebuying them at the beginning of the new year for tax purposes. Aside from occurring at a certain point in time, anomalies can also appear based on the historic price (e.g. *momentum effect*) or the company fundamental data (e.g. *earnings-price anomaly*). The momentum effect is the observation that increasing (decreasing) stock prices tend to continue to increase (decrease) [29]. The earnings-price anomaly is the observation that the stocks of comapnies with a smaller price/earnings ratio tend to perform better than others [30]. Finally, large speculative bubbles followed by market crashes stem from over-/undervalued stock prices.

The existance of professional investors like Warren Buffet who consistently outperform the market over long periods of time is obviously not disputed. According to theories like the EMH, these successes should not be possible over the long term. Furthermore, banks that invested more in any form of market analysis or alogrithmic trading should be at a major disadvantage. Algotrithmic trading utilizes pre-defined rules based largely on historic price and volume [32] and should therefore not be able to outperform the market according to the EMH. Yet the effort in algorithmic trading is so high, that in 2012 algorithmic trading accounted for 85% of the volume according to Glantz et al. [33]. Another claim is, that people acting based on common stock analysis strategies generate a situation where a prediction becomes a self-fulfilling prophecy [34].

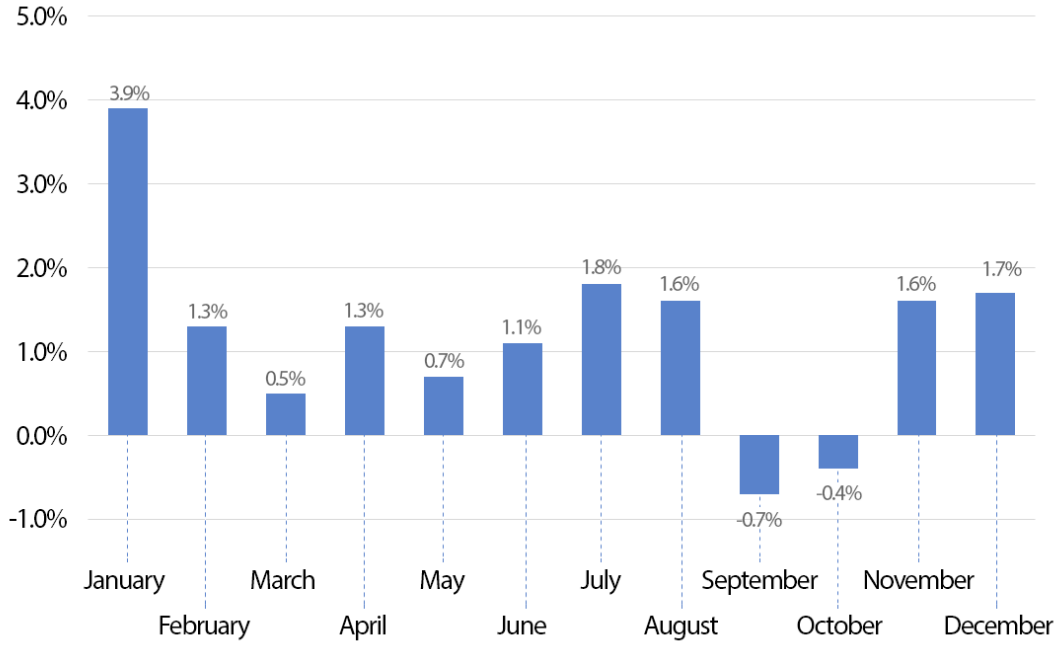

Figure 2.1: Average stock returns by month of the year, from 1927 - 2001; Haugen and Lakonishok, *The Incredible January Effect* [31]. January had significantly larger returns than the other months in that period of time.

These findings and considerations indicate the opposite of the EMH, namely that there is a way to outperform the market, at least to a certain degree. And while it remains unclear which school of thought is right and which is wrong, we will now go over the different kinds of data which could potentially have an influence on the prices of stocks.

There are several different techniques for predicting future stock price movements which are categorized by what kind of data they are based on. A method that uses the price and trading volume of a stock falls under the category of *technical analysis* [35]. The ones that focus on data like company assets, earnings, etc., are called *fundamental analysis* [36]. Still others center their attention on the sentiment of people about a company, expressed in newspapers articles or other media in what is called *sentiment analysis* [37]. While the latter one has been around only more recently, due to successes in machine learning and the availability of media through APIs, technical and fundamental analysis have both been around for a longer period of time. Fundamental analysis has had more support than technical analysis, for example by star investor Warren Buffet who uses it as part of his investment strategy [38], while technical analysis has often been regarded as hoax, as is illustrated in the following quote [39].

> *"It has been argued that the difference between fundamental analysis and technical analysis is not unlike the difference between astronomy and astrology. Among some circles, technical analysis is known as 'voodoo finance'."*
>
> — Lo, Mamaysky and Wang, *Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation (p. 1705) [39]*

Regardless of whether technical analysis is a legitimate approach for predicting stock price movements, numerous research has been done on the subject. We will give an overview over some of the methods and performance findings presented in other researches in the next section.

2.1.4 Technical analysis

As briefly mentioned above, technical analysis is a category of techniques for forecasting price movements based on historic market data such as price and volume. Technical analysis assumes that information relevant to making trading decisions can be found in the past price development of a company's stock. These decision can be observed as patterns in the price chart or as calculated indicators and based on how the market reacted in the past, an estimation on how the price will develop in the future can be made. There are countless different approaches for technical analysis. An overview can be found in the in the book *Technical analysis: The Complete Resource for Financial Market Technicians* by Kirkpatrick II and Dahlquist [35]. The prices of stocks change within fractions of seconds, which makes this type of analysis very important for making *high frequency trading* (HFT) decisions.

The price of a company's stock can be represented graphically in a diagram or chart, e.g. in a candlestick chart. These charts can be analyzed for (reoccurring) patterns such as gaps, spikes or waves. These patterns can be made visible by drawing trend lines or channels around the price movement. Technical or chart analysts will make predictions of future price movements if they find specific patterns in the price chart. One example for a very popular pattern is the *Elliott wave principle*, developed by Elliott in 1938 [40]. Within a trend the price develops in five impulsive waves going in one direction, followed by three corrective waves going in the opposite direction, both times in a zigzag pattern. This theory is based on the assumption that investors are alternating between optimism and pessimism and tend to overreact, creating the need for a correction [41]. This principle is illustrated in Figure 2.2. Other patterns include *head and shoulders*, *cup and handle* or *broadening top* [35].

Another approach is the use of *technical indicators*. These mathematically calculated indexes indicate future price movements, trends or the volatility of a stock. Indicators can be *leading* or *lagging*. Leading indicators precede the price and help to predict it, whereas lagging indicators are used to confirm stock price movements. Furthermore indicators can be bounded to a certain range (also called *oscillator*) or unbounded. An advantage over analyzing the chart for patterns is that technical indicators are quantifiable. For

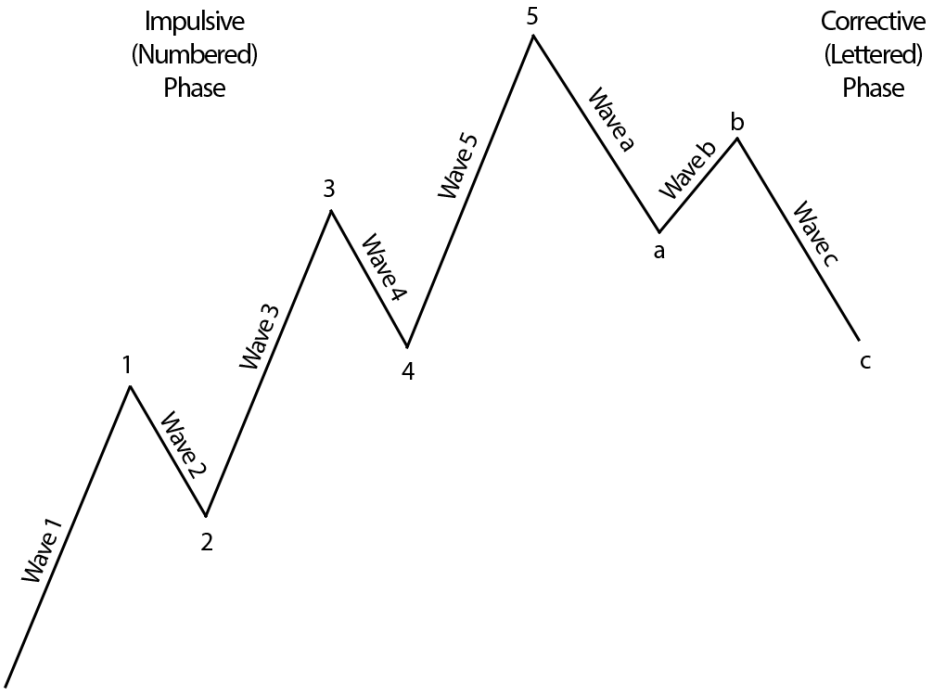

Figure 2.2: Depiction of the Elliott wave principle as an example of a pattern used in technical analysis; Frost and Prechter, *Elliott Wave Principle* [41]. A technical analyst might inspect a stock price to find out if it exhibits this pattern or a part of it, to predict the future price (direction).

most technical indicators, there is a formula that calculates the indicator at a specific point in time. The following are a few examples of the countless different indicators and what they can be used for. Note that this is just a very small selection and that we opted to omit a comprehensive explanation as well as a formula for computing the indicator, which can be found in the literature [35][42].

- **Moving average:** This is a simple mathematical method used to smooth stock price movements (and time series in general).
- **Average directional movement index (ADX):** This lagging indicator is used to determine the strength of a trend.
- **Relative strength index (RSI):** This leading indicator ranges from 0 to 100 used to identify oversold or overbought markets.
- **Moving average convergence/divergence (MACD):** This lagging indicator is used to determine the momentum, duration and strength of a stock's trend.

Note that some patterns or indicators contradict each other, i.e. they predict different outcomes. An analyst can make predictions based on one pattern or indicator or can combine several into a trading strategy. These strategies are usually *backtested* before being used, which means simulating trading on historic data based on the strategy. In the past this was performed manually and rather tedious, nowadays computers carry out backtesting over large periods of time and several different stocks. With backtesting, the effectiveness of strategies can be tested to some degree without risking any money.

Whether or not technical analysis works or is profitable has been subject of debate in many research papers. The following three survey papers each investigate several research results about the performance and profitability of technical analysis. In their 2007 article Park and Irwin investigate 95 studies about technical analysis, 56 exhibiting profitable results when applying technical analysis, 20 unprofitable results and 19 mixed results [43].

In 2017 Nazário et al. examined 85 papers. The papers were classified as 79 in favor of technical trading, 6 not in favor of technical analysis and 4 not applicable (a few papers fall into more than one category), while also classifying by developed and emerging markets, risks, transaction costs, etc. [44].

In 2009 Schulmeister analyzed 2680 different technical trading models and found that the profitability of technical trading has declined from 1960 to 2007. Over the whole period only 2.6% of the models had negative returns [45]. Schulmeister goes over the shift from trading daily over 30-minute intervals to even higher frequencies in recent years and tries to explain that phenomenon with the *Adaptive Market Hypothesis* [46] as follows:

> *"According to the Adaptive Market Hypothesis (AMH) of Lo (2004), markets become gradually more efficient in an evolutionary process. By learning to exploit profit opportunities, market participants will slowly erode these opportunities through an arbitrage mechanism. Once the 'old' and simpler rules have become unprofitable, new and more sophisticated trading strategies will emerge which will gradually also improve market efficiency."*
>
> — Stephan Schulmeister, *Profitability of technical stock trading: Has it moved from daily to intraday data? (p. 199) [45]*

The research on this subject is ongoing. And while it might be arguable if new trading strategies presented in studies are more sophisticated, they are certainly using many different and new approaches. We will present some in the next section.

2.1.5 Stock price prediction

We have already covered some of the different types of data that can be used for predictive models: Technical, fundamental and sentiment data. However the models themselves are very different. The methods include machine learning techniques such as neural

networks, (multi-)linear regression, *autoregressive (integrated) moving average model* (ARMA/ARIMA) models, genetic algorithms and so on. An overview was given by Atsalakis and Valavanis in 2009 [11].

The following papers are meant to give some examples for the different techniques. Gorgulho et al. used *genetic algorithms* (a process of improving solutions inspired by natural selection) on technical indicators to manage a portfolio [47]. Wang used a *principal component analysis* and *support vector machine* combination (a machine learning approach) to predict stock trends [48]. Wijaya et al. compared *ARIMA* and artificial neural networks as forecasting methods [49]. Lauren et al. combined simple moving averages with news sentiment in a neural network approach to predict stock trends [50]. Nadkarni et al. combined *NeuroEvolution* (a method used to evolve the structure of neural networks) with principal component analysis [51]. The effectiveness of different neural networks was also investigated by Nayak et al. [52], Ticknor [53], Rather et al. [54], Khan et al. [55], as well as many others.

Trading the stock market has some disadvantages. Relatively high fees and the cost for real-time data in small intervals make it less suitable for high frequency trading, especially if one does not have a large budget. Beside the stock market, several other assets are susceptible for predictive analysis and are potentially profitable. These assets include commodities, derivatives, foreign currencies and more recently, cryptocurrencies. We will present cryptocurrencies in the next section.

2.2 Cryptocurrencies

Cryptocurrencies are a form of digital asset that is secured by means of cryptography and are usually decentralized, meaning they do not rely on a trusted third party for transactions like banks or credit card companies. Instead they handle transactions in a public ledger, for example a blockchain. They generally have no intrinsic value, but are exchanged for fiat currencies like US dollars, Japanese yen or euros based on supply and demand. The prices fluctuate substantially. The following subsections will go over the fundamentals of cryptocurrencies using Bitcoin as example, some alternative cryptocurrencies (*Altcoins*) and the characteristics of trading cryptocurrencies.

2.2.1 Fundamentals

Cryptocurrencies are different and have their own characteristics compared to others. Some share more similarities than others and most are based on the concept of the first successful cryptocurrency, Bitcoin. The white paper for Bitcoin was published in 2008 under the pseudonym Satoshi Nakamoto [10]. Nakamoto published the first open-source client in 2009 which is when the Bitcoin network went online. The concept of Bitcoin was revolutionary for several reasons. It achieved distributed consensus and managed to prevent double spending attacks, a problem where an attacker spends his money more than once, present in distributed digital cash before Bitcoin. The following

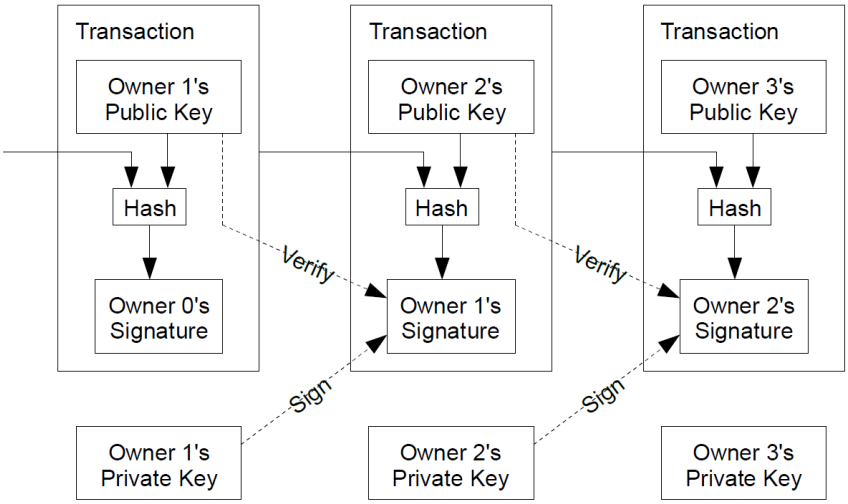

Figure 2.3: Illustration of the verification of transactions in Bitcoin; Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system* [10]. In the leftmost block, owner 0 sends a transaction to owner 1. This balance received in this transaction is used in the middle transaction from owner 1 to owner 2. The leftmost transaction along with owner 2's public key is hashed and signed by owner 1's private key and can be verified with owner 1's public key.

few paragraphs give a short overview over the functionality of Bitcoin, as presented in Nakamoto's white paper.

Bitcoin secures its transactions using asymmetric cryptography, i.e. the *Elliptic Curve Digital Signature Algorithm* (ECDSA). In asymmetric cryptography, a user generally has two keys, a public key and a private key. The first one can be made accessible to the public while the latter one needs to remain private. The keys are then used to perform a one-way function, where one of the keys is used to encrypt, and the other one to decrypt a message. In the case of Bitcoin, the private key is used to create a signature of a transaction, which can be verified with the user's public key. The transaction is signed by encrypting it with the private key and can be checked by decrypting it with the public key and then comparing it with the original message. Through this mechanism, only a person holding the private key can create valid transactions.

To carry out transactions, a user broadcasts them to the Bitcoin network. Several transactions are then combined into a block. Only valid transactions are allowed into a block, so the peers that receive a transaction have to check if the transaction has been signed by the sender and if the sender has enough balance, i.e. if the sender has unspent balance received in an earlier transaction. To be able to do that, the fully participating peers have to keep track of a history of all transactions. This process is depicted in Figure 2.3.

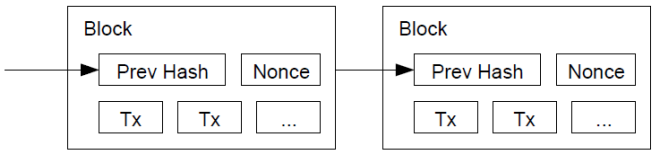

Figure 2.4: The blockchain of Bitcoin; Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system* [10].

In fact, every block and therefore every transaction since the beginning of Bitcoin is stored in a chronologically ordered, linked list called blockchain. One essential characteristic of the blockchain is, that every block contains a hash of the previous block. A hash is the result of another type of one-way function, when applied to a block, results in a collision-resistant fingerprint of it. Since every block contains a hash of the preceding block, the whole list is cryptographically secured up to the first block, see Figure 2.4. Changing a block somewhere in the blockchain would mean changing all following blocks as well in order to result in a valid chain.

This alone however is not enough to secure the blockchain. Bitcoin employs a proof-of-work system. For a new block to be created, a simple but computationally expensive problem has to be solved, i.e. applying a hash function to different values until the result lies within a target range. Now, if a block is changed, the work for that block along with the work of all following block would have to be redone. Bitcoin defines that the longest chain with all valid blocks (and all valid transactions) is the valid chain, representing the majority vote of the computational power in the network. Additionally to making the blockchain tamper-proof, this mechanism also prevents double spending attacks. The probability of a transaction being invalid, because another longer valid chain without the transaction emerges, becomes exponentially less likely, the more blocks are added.

The difficulty of this proof-of-work is set dynamically, so that the collective computational power of the Bitcoin network finds a solution and therefore a new block every ten minutes on average. This creation of new blocks is also referred to as *mining*. The person that mines a new block is rewarded new currency. This acts both as incentive for peers to behave honestly and as a distribution mechanism for the currency, especially in the beginning. As a counter measure against inflation, this reward is halved every 4 years and will eventually run out around the year 2140. Miners are also awarded all transaction fees, which users can voluntarily include in a transaction, so that miners will prioritize them. The mining reward and transaction fees are a strong incentive, so that it is game-theoretically more profitable for miners to act honestly, as long as they do not have a majority of the computational power.

As laid out above, the whole transaction history is included in the publicly available blockchain. As it would be undesirable to have one's real name associated to every transaction ever made, Bitcoin uses the public key of users (or the hash of that) as

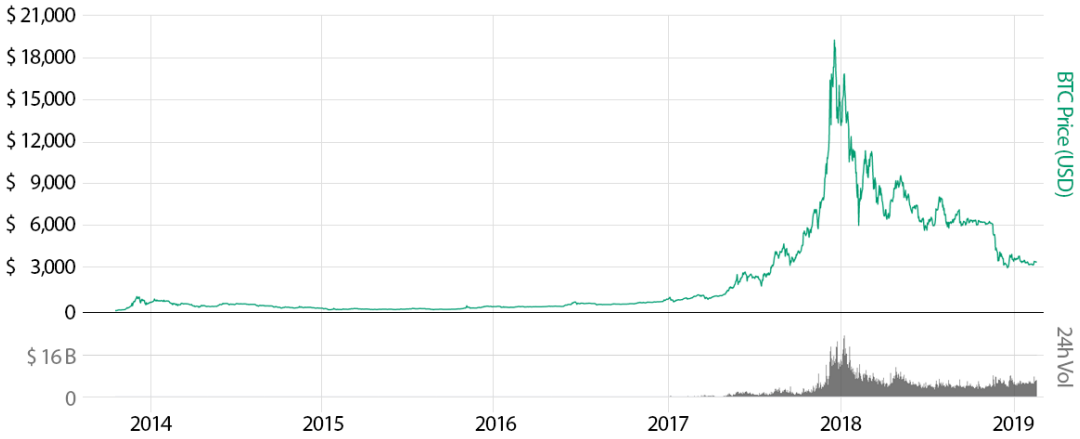

Figure 2.5: Bitcoin price in US dollars over the years, `https://coinmarketcap.com` [57].

pseudonyms for users. This way, their identity stays hidden (to some degree).

Bitcoin's popularity has increased over the last years. Its value in USD, while having been volatile from the beginning, rose steadily from a few cents until reaching its peak of almost 20,000 US dollars for one Bitcoin in December 2017. From then it dropped to around 3,500 US dollars as of January 2019, see Figure 2.5. Bitcoins can be traded for traditional currencies on Bitcoin exchanges. Because of this popularity and the relatively high trading volume, there have been many exchanges for Bitcoin over the years. Some of these exchanges were allegedly associated to illegal activities and shut down, the most prominent case being Mt. Gox [56]. This as well as Bitcoin being used for payments at numerous illegal darknet sites brought some negative publicity.

Bitcoin also has some inherent flaws. One of the problems is scalability. The blockchain's size surpassed 200 GB of size in early 2019 and will grow steadily. The size of blocks is limited and only about 4,000 transactions fit into one block. Because a block is mined every 10 minutes, this means that only around 7 transactions can be handled per second on average, a rather small number compared to for instance credit card systems. This can result in very long waiting times or high fees in the Bitcoin system when carrying out transactions. Finally, as mentioned briefly above, Bitcoin is not completely anonymous, only pseudonymous. There are ways of tying different Bitcoin addresses together (hashes of public keys) that belong to the same person. Because there generally has to be some sort of interface to conventional money, usually via exchanges, there are points where personal information could be leaked. At the very least these exchanges have to be regarded as trusted third parties, something that Bitcoin's goal was to avoid. If one's information gets linked with a Bitcoin address (or all of that persons Bitcoin addresses), every purchase ever made is visible in the (public) blockchain.

Implementing changes into Bitcoin is difficult, because the users have to agree and update their client. Otherwise a fork occurs, i.e. the cryptocurrency splits into two new ones, one that accepted the changes and one that did not. This happened for instance when Bitcoin split into Bitcoin and Bitcoin Cash. Due to this criticism and the difficulties faced when implementing changes into the Bitcoin protocol, along with the immense hype around cryptocurrencies in general, many alternative cryptocurrencies (altcoins) have emerged over the years. Some of them will be presented in the next section.

2.2.2 Altcoins

Since Bitcoin's release, over 5,000 different cryptocurrencies have been introduced as of early 2019 [58]. These usually take on one of the flaws of Bitcoin, improve in a very specific aspect or introduce totally new features.

Monero is a cryptocurrency based on a paper from Van Saberhagen [59] and improves on anonymity, making it impossible to link transactions. *IOTA* is a cryptocurrency that introduces the *tangle*, a directed acyclic graph, to combat the scalability problem of Bitcoin [60]. *Ripple* also improves on scalability, by replacing the blockchain with fewer server nodes that find consensus on transactions every few seconds. Ripple works closely together with banks and features representing other currencies and IOUs ("I owe you"s) [61].

A popular example of a cryptocurrency that introduces a completely new feature is Ethereum [8]. Like Bitcoin, Ethereum has a blockchain securing the transactions. However on top of that, Ethereum introduces a Turing complete scripting language, allowing the deployment and execution of distributed applications on the blockchain, and in particular smart contracts. As briefly mentioned in chapter 1, smart contracts allow that parties come to a cryptographically enforced agreement and eliminate the need for notaries (in some cases) [9]. As of early 2019, Ethereum was the second to third largest cryptocurrency in US dollar market capitalization, contested by Ripple [15][58].

This rapid development of altcoins and cryptocurrencies brings a high dynamic to the market. The question of whether or not the Bitcoin price influences all other cryptocurrencies is an interesting one, and will be looked into in this thesis.

2.2.3 Trading cryptocurrencies

Cryptocurrency exchanges act as an interface between cryptocurrencies and conventional money. Aside from mining, trading is the way for people to acquire cryptocurrencies. Since (most) cryptocurrencies have no tangible value, their price is determined at these exchanges based on supply and demand. Their price development is however very like that of stocks. For this reason, we can apply the concepts about time series and stocks mentioned in Section 2.1 to cryptocurrency price movements.

We can translate (most) of the concepts of technical analysis directly to trading cryptocurrencies. Sentiment analysis is more or less the same as well. However, cryptocurrencies do not have fundamental data, at least not in the way that stocks have data about their company's revenue, profit and so on. There are other numbers that represent the general acceptance of a cryptocurrency. In Bitcoin for instance, the number of currently active miners, the current number of transactions in the system, the transaction fees in the current block could give similar information.

There also are some differences between trading cryptocurrencies and trading stocks. When looking at the price development of cryptocurrencies from the beginning up until early 2019, it becomes apparent that most cryptocurrencies are (similar to *penny stocks*) very volatile, i.e. they have a lot of price movements over small periods of time. Unlike (most) penny stocks however, the trading volume of cryptocurrencies is rather high [15]. Another difference is, that cryptocurrency exchanges are open 24 hours seven days a week, unlike conventional stock exchanges that close on weekends and at night.

Having real-time stock data can be very expensive. The professional trading platform *Bloomberg Terminal* which provides real time data of stocks, analyses and many more services, costs around 24,000 US dollars per year [62]. There are of course cheaper alternatives or the data might be included with one's broker, but with cryptocurrency exchanges, this data usually comes for free. *Binance* for instance, one of the currently largest cryptocurrency exchanges, and most others make this data available in their free API. Alongside providing price data in very fine granularity (up to individual trades), the APIs offer ways to place orders. This is very useful for automated trading.

Like with stocks, the trading orders that are carried out are usually subjected to fees. Some exchanges distinguish between maker orders, orders that are placed into the order book and provide liquidity, and taker orders, which take orders out of the order book and remove liquidity. The fees vary from exchange to exchange. They are around 0.075% to 0.3% of the order for taker fees, while maker order fees range from 0 to 0.1%, often depending on one's monthly trade volume. The traded volume is so high, that despite these relatively small fees, the exchanges are making millions in daily revenue, according to an estimation of Bloomberg (Figure 2.6) [63].

The volatility despite the high trading volume along with relatively low fees and free APIs make cryptocurrencies an ideal candidate for testing automated trading strategies of any sort. Especially high frequency trading does not seem to be as penalized as with stock brokers or at least does not require a high investment in software or equipment. For these reasons, Bitcoin and other cryptocurrencies have become popular research candidates for analyzing and predicting their price movements. The benefits are similar to those of stock price prediction. We will present some of the approaches below. Note that research that is similar to this paper will be presented on its own in Section 2.5 and that most of the research has been done only on Bitcoin.

The attempts for predicting cryptocurrency prices are various in their methodology, relying on different models and data. In 2015 Kristoufek researched what different factors

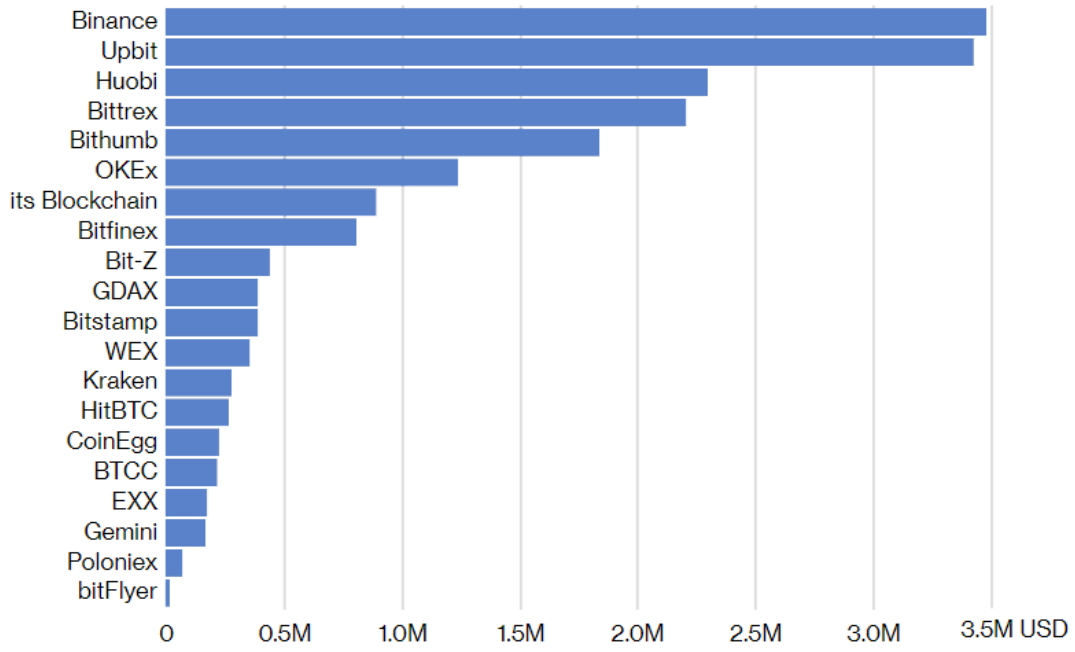

Figure 2.6: Estimated daily revenues of various cryptocurrency exchanges, according to a March 5, 2018 Bloomberg article *Crypto Exchanges Are Raking in Billions of Dollars* [63].

influence the price of Bitcoin [64]. There has been analysis on the influence of what we earlier defined to be similar to fundamental data in cryptocurrencies, also in 2015, when Greaves et al. tried using the transaction graph as means for predicting the price of Bitcoin with a price direction accuracy of 55% [65].

Sentiment analysis is a computer science field with numerous applications, where text is automatically analyzed and assessed based on the subjective feeling behind it, either leaning towards a positive or negative sentiment [37]. One of these applications is to analyze news, Twitter posts (*Tweets*) or other media in order to determine the sentiments towards an asset like cryptocurrencies or Bitcoin. In 2014, Kaminski analyzed the sentiment of Twitter posts and their relation with the Bitcoin closing price and volume [66]. Matta et al. studied the correlation of the Bitcoin price's spread and Tweets as well as data of Google Trends in 2015 [67]. Georgoula et al. also analyzed Twitter posts using *support vector machines* and found a positive correlation to the Bitcoin price [68].

There have been attempts using machine learning by Madan et al. in 2015 [69] and more specifically neural networks [70] [71] for forecasting the price of cryptocurrencies. These will be presented after the two following introductory sections about neural networks and their evaluation.

2.3 Neural networks

We will now discuss neural networks as a method for arbitrary function approximation, the main concept used in this thesis for predicting the price or trading signals for the cryptocurrency Ethereum. We will first go over machine learning in general as well as feature engineering and then introduce the theory behind neural networks and more specialized concepts such as recurrent neural networks, long short-term memory cells and convolutional neural networks.

2.3.1 Machine learning

Machine learning is a subset of artificial intelligence and is used for solving tasks based on learned patterns and models from previous data. In contrast to building a specific set of instructions or an algorithm, machine learning is learning to act by analyzing data. Machine learning is used to find patterns in data automatically. It is used in many different fields, such as biology, physics, medicine and of course computer science. Specific use cases include financial fraud detection, spam detection, face detection, speech recognition and so on [72].

The tasks that computers traditionally carry out might be complex and computationally expensive, but can be formalized in a step-by-step algorithm, that somebody came up with and a programmer implemented in machine-readable code. There are, however, tasks that cannot be formalized easily or not at all; tasks that require intelligent behavior or that may even exceed the intellectual capabilities of humans. While voice and image recognition are examples for the former, weather predictions and analyzing astronomical or microbiological datasets are instances of the latter, in a sense that the amount of data which has to be taken into account is beyond being a manageable task for the human brain. For these tasks, machine learning is used, which takes advantage of the rapidly increasing computational power, disk space and memory capabilities of modern hardware.

Problems solved or at least tackled by machine learning generally involve one or more input parameters, also called *features* and one or more outputs called *target(s)*. A large data set (of at least input values) is required for the learning process. Here, the goal is to find or approximate a function, which maps these input values to the right output values or to find patterns within the data.

Using a machine learning algorithm is usually divided into two phases: the training or learning period, where the algorithm uses a defined set of examples to learn from, and a testing period, where the algorithm's performance is tested on a different set of samples. Depending on the way in which the learning is accomplished, different machine learning algorithms can be divided into the following categories [73].

- **Supervised learning** is a class of learning algorithms trained on labeled data. In other words, for the input values of every sample in a training set, the right output value or (*ground truth*) *label* is given. In the learning process the used

learning algorithm generates a function that maps the input to the output as closely as possible, according to a defined metric. This function can then be applied on unobserved input values to test the trained model or predict the output.

- **Unsupervised learning** is, in contrast to supervised learning, not dependent on labeled input data. The algorithm analyzes the given input data to detect anomalies, group the data by similarities (*clustering*), etc.
- **Semi-supervised learning** is a mixture of the former two methods, where a small amount of the training data is labeled while the rest is unlabeled.
- **Reinforcement learning** is a technique where instead of providing the right output values (labeled data), a reward function is given. The learning algorithm has to find the right set of actions in order to maximize the reward, usually through means of trial and error. Learning to play games is one of the usage examples for reinforcement learning, for instance the chess program *AlphaZero* developed by *(Google) DeepMind* was trained by playing games against itself and within 24 hours (allegedly) managed to defeat the best chess programs [74].

Supervised learning can be divided into two categories based on the type of output data (targets) used. If the goal is to map the input values to a discrete and finite number of classes it is called *classification*. *Regression* refers to the case, where the output values are continuous. An example for a classification problem would be the recognition of cats on images, where the training set consists of labeled data with two possible classes "image contains one or more cats" and "image does not contain any cats". An example for a regression problem is forecasting a companies stock price based on historic values.

There are numerous different machine learning algorithms, each used in different areas and with its own advantages and disadvantages. The following list gives a brief overview over different methods.

- **Linear regression** is an elementary method and simple example for supervised learning [75]. The output function is modeled as a sum of the products of every input with a coefficient plus an offset. For p different inputs or predictor values, linear regression looks as follows.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon = \beta_0 + \sum_{i=1}^{p} (\beta_i X_i) + \epsilon$$

  Y is the dependent output value, $X_1, X_2, ..., X_p$ are the input values, $\beta_0, \beta_1, ..., \beta_p$ are unknown coefficients and ϵ is the error term. A common measure for closeness is the squared difference between the predicted value of the model and the actual value. The coefficients are generated by calculating the values $\beta_0, \beta_1, ..., \beta_p$ that minimize the sum of the squared difference for every training sample. An example for a linear regression with one input ($p = 1$) can be seen in Figure 2.7.

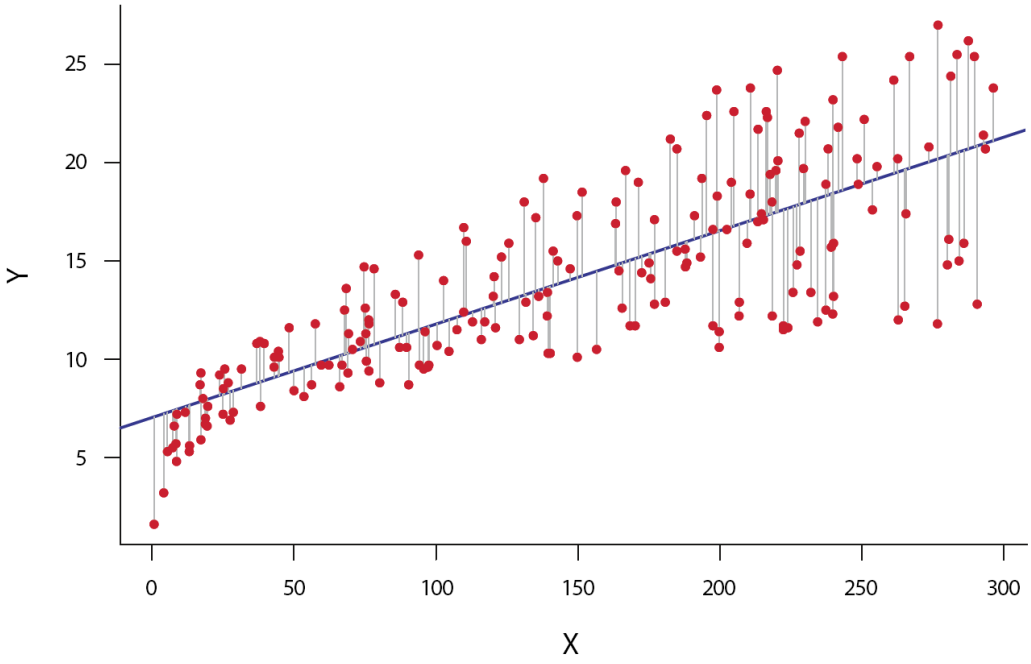

Figure 2.7: Linear regression for one-dimensional input; James et al., *An introduction to statistical learning* [75]. The red dots are the data samples, the blue line the approximated function and the vertical lines represent the errors. The sum of the squares of these errors is to be minimized.

- **Logistic regression** is another regression model that (in its simplest form), instead of approximating a function of the output, estimates a probability of two possible outcomes [75]. The model looks as follows.

$$P(Y = 1) = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{i=1}^{p} (\beta_i X_i)\right)}}$$

  $P(Y = 1)$ is the probability of the output being of class 1. β_0 or β_i are the coefficients and X_i the input values, as with linear regression. In fact the (negative) exponent is the term that is used in linear regression (without the error).

- **Support vector machines (SVM)** use lines, or in the more general multi-dimensional case hyperplanes, to group data into classes [73]. The hyperplanes are selected so that the distance between the hyperplane and the nearest point or vector of each class (*margin*) is maximized. These nearest points are called support vectors. The margin is maximized so that new unobserved data is more likely classified correctly. Separating data into two classes with a line is very simple if the data is linear separable. If it is not, then the SVM can still be applied by using *slack variables* and/or the *kernel trick*, where the data is lifted to a higher

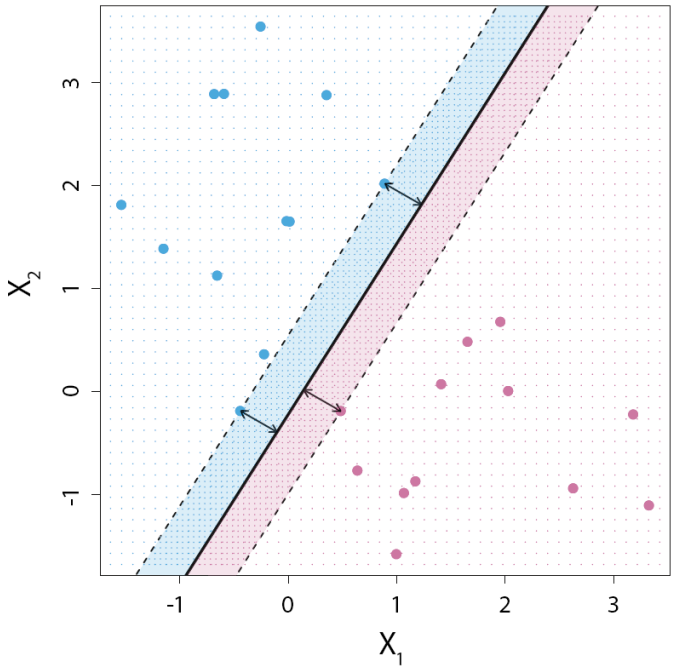

Figure 2.8: Support vector machine for two dimensional input; James et al., *An introduction to statistical learning* [75]. There are two classes of data, blue and purple. The dots on the dashed line are support vectors. The hyperplane, in this case a line, is drawn as a solid black line. The distance between the support vectors and the hyperplane is maximized.

dimension in order to make the data linear separable. An example for a support vector machine can be seen in Figure 2.8.

- **Decision trees** are used as a predictive model [72]. At every split, a condition is checked and the algorithm continues in the according branch. A simple example for such a condition or splitting rule is a threshold on the value of a feature. At every split, the best feature according to some measure is selected. For instance the feature with the highest information gain could be selected by calculating the entropy. Decision trees can be used for classification or regression.

- **Random forests** are an ensemble of decision trees [76]. Every tree generates a prediction and the prediction supported by the majority of trees is chosen. For this method, several decision trees are built. At every tree at every split, the best feature out of a random subset of all features is selected. Random forests have the advantage of being efficient and fast learning at the price of high overfitting.

- **Bayesian networks** represent the conditional dependencies of features in an directed, acyclic graph. Features are symbolized as nodes and (directed) conditional

relationships as edges. They are used to infer the probability of an output and to visualize conditional (in)dependence. There are algorithms for generating the structure of the Bayesian network and the parameters, that would go beyond this brief synopsis, but can be found in an article by Friedman [77].

- Eventually, **Aritficial Neural Networks** are also a type of machine learning algorithm. They will be explained in Section 2.3.3.

2.3.2 Feature engineering

To be successful, machine learning algorithms typically require large amounts of data to be trained on. This data in its raw form is often not suitable for machine learning. There might be too many different input dimensions, the data might be scaled in a wrong way because of a small number of outliers or errors in measurement, or the data might simply not be usable in its current form for some other reason. An essential part of training a successful machine learning model for solving a certain task is preprocessing this raw data by engineering good input features from it, in the sense that the features should be expressive of the outcome, and then transforming them to suitable formats or ranges. This process is called feature engineering [78].

Not all features influence the target output in the same way. For example when trying to forecast if it is going to snow the next day, the current time of the year, e.g. January, might be more helpful than the current wind speed. Selecting good features is hard and requires expert knowledge in the specific domain, which in regard to the weather example the author of this thesis certainly does not have. Good features are desirable, whereas features that have little to no impact at all on the outcome should be discarded. Raw data can come in many different forms, such as images, text or numbers. And while some form of transformation is certainly necessary for formats like images and texts to be readable by the model, even numerical data often has to be transformed in some way to be made usable.

To expand upon the weather example, assume we want to forecast the temperature of the next day. In our hypothetical raw data, we have the daily temperature measurements over the last 40 years. The labeled data is two dimensional consisting of the commonly used *Unix timestamp*, which is measuring the seconds that have elapsed since January 1, 1970, and the temperature in degrees Celsius as label. The Unix timestamp is not going to be very helpful in predicting the weather. If we transform the timestamp through simple calculation into two new features, the month and day in which the temperature was measured however, the situation changes. Suppose it is hot in the northern hemisphere's summer months, e.g. in July (7^{th} month), the new data now has numerous data samples with higher temperature in combination with the month 7. The machine learning algorithm might be able to learn faster and make more accurate predictions. This fictional example is shown in Figure 2.9.

The process of selecting, extracting, combining or reducing features is often done manually and can be time consuming. The steps can involve iterations of brainstorming and deciding

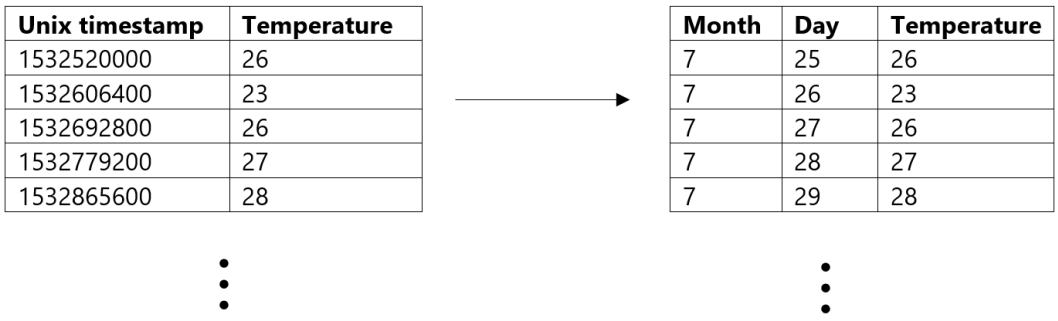

Figure 2.9: Simple, fictional example of feature engineering in the context of temperature forecasting. The raw data's (left) timestamp is converted to month and day (right). Note that the month and day act as the features (input), while the temperature is the label (output).

which features to use and testing how the impact on the model's performance [79]. To aid the process, a variety of mathematical concepts are used. An example for a technique helping to reduce the input dimensions is *principle component analysis* (PCA) [80]. Too many input dimensions can lead to overfitted models and bad performance. PCA is a statistical method which can simplify datasets. The raw and possibly correlated input data is approximated by a smaller set of uncorrelated features, the principle components [81]. Other methods for best feature selection such as the *analysis of variance* (ANOVA), F-tests or cross-correlation calculation are included in popular machine learning libraries (e.g. scikit-learn [82]). Dingli et al. used these methods for selecting the best technical indicators for stock prediction in 2017 [79].

The values and scale of the features are other very important aspects. If the values of the various inputs are different in sizes of several orders of magnitudes, the larger features can outweigh the other features. Linear functions for instance are very sensitive to this phenomenon [78]. A solution to this problem is to normalize the data by scaling the different features to a common range, in a certain way. Doing so increases the learning rate and overal performance of machine learning algorithms, e.g. of neural networks [83]. There are several different ways to do this. The preprocessing approaches which are used in this thesis will be explained in Section 3.4.

2.3.3 Overview over neural networks

Neurons, a mathematical model inspired by their biological counterparts found in the human brain, were proposed by McCulloch et al. in 1943 [84]. Based on these neurons, Rosenblatt et al. introduced the *Perceptron* by connecting several of them [85], which are the foundation for neural networks developed in later years.

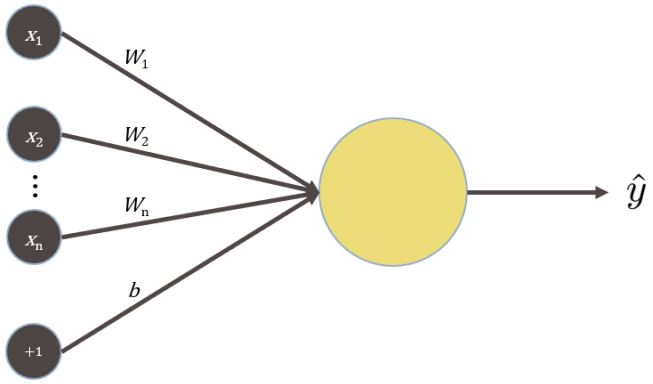

Figure 2.10: Illustration of a single Neuron; Andrew Ng, *Sparse autoencoder* [86]. The neuron has n inputs $(x_1, x_2, ..., x_n)$ each with its own weight $(W_1, W_2, ..., W_n)$ and a bias b. The output $\hat{y}$ is $\hat{y} = f\Big(\sum_{i=1}^{n}(W_i x_i) + b\Big)$.

Neural networks are mathematical models for approximating non-linear functions. They have several neurons and connections between them. Every neuron calculates an output based on the given input and the weight of its incoming connections. The (supervised) learning is accomplished by running a backpropagation algorithm that adjusts these weights based on the supplied output value (label). Several neurons are stacked inside layers. Neural networks that consist of an input layer, a hidden layer and an output layer are sometimes referred to as shallow neural networks, while deep neural networks consist of more hidden layers. The following paragraphs will go over this concept in more detail, based on Ng's article of 2011 [86].

A neuron has several incoming connections from previous nodes, inputs or the bias, as well as one output. The products of every input (including the bias) and its corresponding weight are summed up and an activation function is applied. The result of one neuron is calculated as follows, an illustration is shown in Figure 2.10.

$$\hat{y} = f\Big(\sum_{i=1}^{n}(W_i x_i) + b\Big) \tag{2.1}$$

Some commonly used activation functions $f(\cdot)$ include the following.

- **Sigmoid function:**

$$f(x) = \text{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

- **Hyperbolic tangent:**

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

- **Rectifier:**

$$f(x) = \max(0, x) \tag{2.4}$$

Note that when using the sigmoid function, the result $\hat{y}$ equals the one from logistic regression. The computation is usually sped up by vectorizing. Given the input vector $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and the weight vector $W = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_n \end{bmatrix}$, the calculation of $\hat{y}$ can be expressed as:

$$\hat{y} = f(W^\top x + b) \tag{2.5}$$

with $W^\top$ being the transposed weight vector.

Neurons are combined to form larger neural networks, as seen in Figure 2.11. To demonstrate the calculation of a neural networks output, we will use a popular notation (in this form or slightly differently used for instance by Ng [86]): $o^{[l]}$ means o in layer l. Calculating $a^{[l+1]}$, the activation (output) vector of the layer $l+1$, or in other words, the output of every neuron in that layer, is computed as follows.

$$a^{[l+1]} = f(W^{[l]} a^{[l]} + b^{[l]}) \tag{2.6}$$

$W^{[l]}$ is the weight matrix of the previous layer l, i.e. the stacked, transposed weight vectors for every neuron in that layer. $a^{[l]}$ is the activation (output) of the last layer l, with the first one being $a^{[1]} = x$.

In general, several input samples are to be calculated. Calculating the result of several samples can be vectorized in a similar fashion.

Remember x as the input vector of a single neuron. Now let $x^{(i)}$ be the i^{th} of m input sample vectors and X be the matrix of horizontally stacked training samples.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

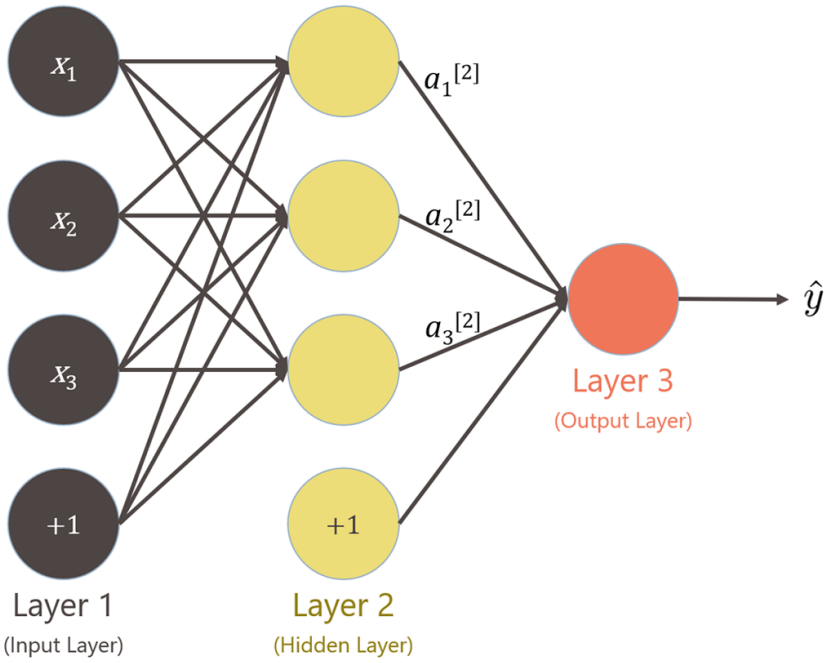

Figure 2.11: Example of a (shallow) neural network with an input layer, a hidden layer and an output layer; Andrew Ng, *Sparse autoencoder* [86]. There are three input features x_1, x_2, x_3, three neurons in the hidden layer, one neuron in the output layer. The +1 nodes denote the biases, which get their value from their bias weight.

We can now calculate the second (first non-input) layer with:

$$A^{[2]} = f(W^{[1]}X + b^{[1]}) \tag{2.7}$$

$A^{[2]}$ is a matrix of the outputs of layer 2 for every training sample stacked horizontally.

$$A^{[2]} = \begin{bmatrix} | & | & & | \\ a^{[2](1)} & a^{[2](2)} & \dots & a^{[2](m)} \\ | & | & & | \end{bmatrix}$$

Consequently, subsequent layers are calculated using:

$$A^{[l+1]} = f(W^{[l]}A^{[l]} + b^{[l]}) \tag{2.8}$$

This vectorization provides an efficient calculation over all data samples. The process of calculating the output of a neural network is also called *forward propagation* or *forward pass*.

The weights of the connections of neural networks are initialized randomly, which will possibly result in a very poor performance. To improve a neural network's performance, learning is required. Since we are looking at supervised learning, we have a whole set of labeled data, i.e. the label $y^{(i)}$ for each input vector $x^{(i)}$ is given. The – via forward propagation calculated – output $\hat{y}^{(i)}$ is supposed to be an approximation of the label $y^{(i)}$. We need to define a *loss function* (or *error function*) $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ to measure how accurate this output is compared to the label. We then define the *cost function* as an average of the losses over all samples. The total cost is $J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}, y)$. Note that in some references the terms *loss function* and *cost function* are used synonymously and are sometimes defined slightly differently. Numerous different loss functions can be selected, often depending on whether it is a classification or regression problem. A common loss function for regression is the squared error while for classification one can measure how far the probability of the predicted class diverges from the actual label.

The loss for each sample, should be as small as possible. *Gradient descent* is a method used for finding (local) minima of a function by calculating the gradient of the function and stepping in the negative direction of it [87]. We use gradient descent in order to minimize the loss, by subtracting the gradient with respect to a specific weight from that weight for every weight and the bias. In particular, we use a method called *backpropagation* to efficiently compute the gradients in directed computational graphs such as neural networks (*backward pass*). The method can, for example, be found in the book *Artificial Intelligence: A Modern approach* by Russell and Norvig (pp. 726-737) [88]. We will present it here as well. To illustrate how backpropagation works, we will show an example for one neuron. We will calculate the gradient of the loss function with respect to one weight W_1 for one sample. We begin by splitting the computation for the output of one neuron (Equation 2.1) into two parts:

$$z = \sum_{i=1}^{n}(W_i x_i) + b$$

$$\hat{y} = \sigma(z)$$

We chose the sigmoid activation function σ. Now let the loss function $\mathcal{L}$ be:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

We want to derive $\mathcal{L}$ with respect to W_1, $\frac{d\mathcal{L}}{dW_1}$. By applying the chain rule twice we get:

$$\frac{d\mathcal{L}}{dW_1} = \frac{\partial\mathcal{L}}{\partial\hat{y}} \cdot \frac{\partial\hat{y}}{\partial z} \cdot \frac{dz}{dW_1} \tag{2.9}$$

These derivatives are:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z} = \sigma(z)\Big(1 - \sigma(z)\Big)$$

$$\frac{dz}{dW_1} = x_1$$

Putting them together we arrive at:

$$\frac{d\mathcal{L}}{dW_1} = (\hat{y} - y)\sigma(z)\Big(1 - \sigma(z)\Big)x_1$$

Given a learning rate α, the weight W_1 is now updated to the new weight W_1' like this:

$$W_1' = W_1 - \alpha \frac{d\mathcal{L}}{dW_1} \tag{2.10}$$

The process works analogous for the other weights and the bias. For other activation functions or loss functions and a weight (or bias) W_i we use:

$$\frac{d\mathcal{L}}{dW_i} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot x_i \tag{2.11}$$

Applying backpropagation on a neural network with more layers and several neurons per layer works similarly. The weights and bias of every layer are adjusted from the last to the first, while propagating the error. This calculation can be vectorized as well, making it a rather efficient learning algorithm. Neural networks have various applications, which were already listed in Section 1.1. In their standard form, i.e. the form we explained above, neural networks are unable to learn sequential or temporal dependencies. They are sometimes called *feed forward* neural networks. The next section will introduce an approach for dealing with these problems.

2.3.4 Recurrent neural networks

So far, the neural networks we looked at were acyclic. And even though samples affect the weights in the learning process, it is very difficult for a conventional neural network to recognize sequential or temporal dependencies in the data. Dependencies in data sequences like time series can be found everywhere and the assumption that there is one in stock (or rather cryptocurrency) price data is of essence for this thesis.

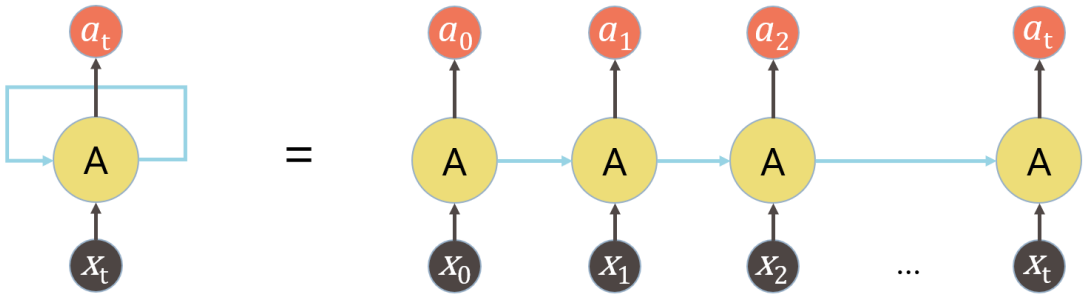

Figure 2.12: Neuron with recurrent connection (left) and unrolled (right); Christopher Olah, *Understanding LSTM Networks* [90].

Assume a chef at an imaginary cafeteria. The chef is bored and even though the only meal always costs the same, he hands out differently sized portions, large and small ones, depending on his mood. If it is sunny outside, exactly every second portion is large. If it is cloudy or raining, only every third portion is large. A feed forward neural network trying to predict the size of the next meal the chef is going to hand out will possibly perform poorly and will not be able to perform much better than guessing, even though this task appears rather simple. The decision of the feed forward neural network is only affected by the current input, no matter how complex it is.

To solve this problem, *recurrent neural networks* (RNNs) have been invented. The idea was to loop information that will serve as a memory. They are neural networks that allow recurrent connections between neurons [89]. Recurrent connections are connections from a neuron to either itself or a neuron that came before it, making the network no longer acyclic. The way these connections work is that the output of the neuron is used as input in the next sample (time step), so that previous data can potentially influence the output of the network. A connection where the output of a neuron is simultaneously used as its input is graphically illustrated in Figure 2.12. For a better understanding, the connections are usually unrolled in time, which means that for every time step the same node is drawn, forming a long chain of neurons.

Recurrent neural networks are successful in solving tasks that are based on sequences, like the example with the cafeteria chef above. They are widely used for sequential problems, such as text analysis, speech recognition, video analysis, natural language processing and time series analysis [91]. Recurrent neural networks can be trained with gradient descent, the most common approach for this is *backpropagation through time* (BPTT). On unfolded recurrent neural networks, BPTT works conceptually similar to backpropagation of multi-layer feed forward neural networks. We will not explain this approach here, but refer the reader to the paper of Werbos published in 1990 [92] or a very compact explanation in the book *Supervised Sequence Labelling with Recurrent Neural Networks* by Graves (pp. 19-20) [89].

2.3.5 Long short-term memory (LSTM)

When training recurrent neural networks, the error is backpropagated for every (unrolled) time step. At every step, the impact the gradient has on the weights gets exponentially smaller. This problem is known as the *vanishing gradient problem* [93]. Hochreiter was the first to identify this behavior and its cause [94]: The derivatives of common activation functions have a range between 0 and 1 and are backpropagated by applying the chain rule (i.e. multiplying the derivative) in every layer. Unrolling RNNs over n time steps is analogous to having n layers, therefore the backpropagated gradient becomes exponentially smaller for every step in time [95]. Due to the weight changes becoming so small, recurrent neural networks using gradient descent learning are unable to learn long-term dependencies.

In 1997, Hochreiter and Schmidhuber introduced *long short-term memory* (LSTM) [96] as an approach that tackles the vanishing gradient problem. The internals of a single LSTM unit (or cell) are more complex than those of simple neurons, but they can be connected in a similar fashion. Figure 2.13 shows a single LSTM unit analogous to an unrolled neuron of a recurrent neural network shown in Figure 2.12. Instead of a single output, the LSTM cell has two outputs values, the memory c_t and the actual output a_t. Both these outputs are recurrent connections to the cell itself, i.e. time step t will have the memory and output of time step $t-1$ along with the input of time step t as inputs. Besides being fed to the cell in the next step, the output a_t is the actual output of the cell at time step t.

The essence of an LSTM cell is the memory c_t of the unit. The memory can store vital information about (long-term) dependencies over long periods of time. New information is stored in it according to the old memory, the old output and the input. The mechanisms that control this flow are referred to as *gates* or *valves*. The forward pass functions as follows (compare Figure 2.13 and compare [90]).

1. The forget gate controls what information of the memory from the last state is passed into the current state. It is calculated like this:

$$f_t = \sigma(W_f \cdot [a_{t-1}; x_t] + b_f) \tag{2.12}$$

   The input vector x_t is concatenated with the output of the previous block a_{t-1}, this new vector is multiplied with the weight matrix W_f, the bias vector b_f is added and then sigmoid function is applied (element-wise). This forget gate output is then element-wise multiplied with the memory of the previous state. Because of the sigmoid function, the values of that vector are between zero and one. If a value of the output vector of this forget gate is close to one, that value of the previous memory state is carried over, if it is close to zero, that value will be "forgotten".

2. After the forget gate, the memory is updated. A candidate vector $\tilde{C}$ is calculated along with the output of the input gate i_t. The input gate will decide, which

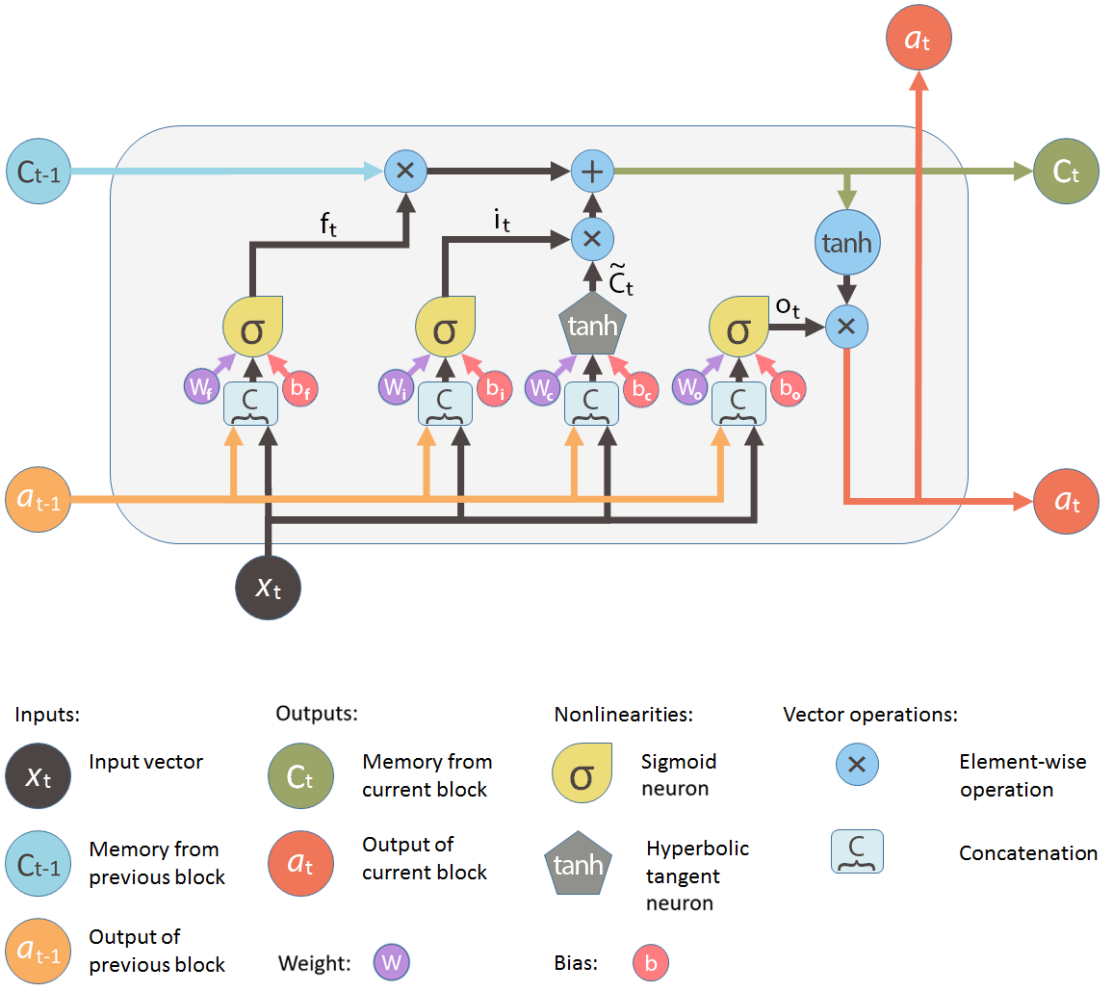

Figure 2.13: Visualization of the internals of an LSTM cell; Shi Yan, *Understanding LSTM and its diagrams* [97].

parts of the memory will be updated, by an element wise multiplication with the candidate vector, resulting in the new memory state C_t. This process, as well as the calculation of these values works analogous to the forget gate:

$$i_t = \sigma(W_i \cdot [a_{t-1}; x_t] + b_i) \tag{2.13}$$

$$\tilde{C}_t = \tanh(W_c \cdot [a_{t-1}; x_t] + b_c) \tag{2.14}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{2.15}$$

3. Finally, the output gate's output o_t is calculated and multiplied element-wise with the *tanh* of the memory state C_t, resulting in a_t. This is the new output of the LSTM unit:

$$o_t = \sigma(W_o \cdot [a_{t-1}; x_t] + b_o) \tag{2.16}$$

$$a_t = o_t \times \tanh(C_t) \tag{2.17}$$

Note that instead of using input vectors, several training samples can be stacked into a matrix and fed at once to speed up calculation, as with feed forward neural networks above. There are many variations of LSTM units, some for instance containing *peepholes*, where the different gates are also concatenated with the previous memory state. The forward and backward pass can be easily calculated, the concept is similar to feed forward neural networks. The calculation varies for every different part (gate) of the LSTM unit. We will not list this here, but rather link to a short overview and collection of all required equations for both forward and backpropagation (again) by Graves (pp. 37-38) [89], who used a slightly different notation. LSTM networks are able to learn long-term dependencies and are used for instance to analyze time series. The computational complexity of the proposed training process for LSTM units is $O(W)$, with W being the number of weights [96]. This complexity is equal to the one of backpropagation through time (BPTT).

2.3.6 Convolutional neural networks

Another improvement of neural networks accredited to LeCun et al. is to apply a mathematical *convolution* (or filter) to the input [98]. Networks based on this approach are known as *convolutional neural networks* (CNNs) and were inspired by the visual cortex of the human brain [99]. At first CNNs were mostly used for computer vision, but they are also applicable to virtually any other domain such as natural language processing, speech recognition and other areas. One of the advantages of convolutional neural networks is that applying those filters minimizes the need for preprocessing of the data. The CNN learns to filter the input data in a way that eliminates or rather reduces the need for hand engineered features based on expert domain knowledge.

Typically, CNNs are structured like multi-layer feed forward neural networks, but in addition have one or more convolution layers and *pooling* layers placed at the beginning [100]. The input is connected to a convolution layer that applies a discrete convolution based on a filter kernel. The output of this is fed to the pooling layer. These two steps can be repeated several times by feeding the output of the pooling layer into another convolution layer and so on. The input can have one or more dimensions. The dimensions of the kernel are chosen in accordance with the input. In image processing the data is usually two or three dimensional ($width \times height \times channel$ for RGB color images). A

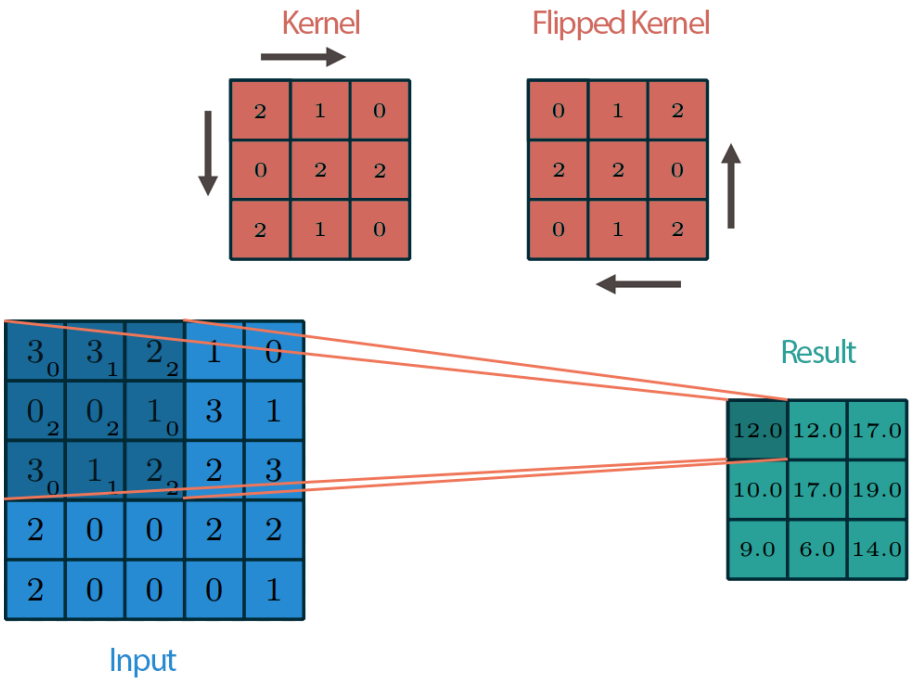

Figure 2.14: Illustration of a 2D convolution; Dumoulin and Visin, *A guide to convolution arithmetic for deep learning* [100]. The values of the timeframe (dark blue) of the input (blue) are element-wise multiplied with the flipped kernel (red) and added up to get the corresponding value of the result. Edges are ignored.

two dimensional example is shown in Figure 2.14. Like other neural networks, CNNs can be trained with backporpagation [101].

In the convolution layer a convolution between the kernel and the corresponding cluster (window) of input values is calculated. In other words the (flipped) kernel and the window, which are the same in size, are multiplied element-wise and then all these values are summed up into a single (scalar) value. On this resulting value, an activation function can be applied. The window is then moved by the size of the stride and applied to that cluster of inputs. For instance, if the input is a 2d image and the stride is 1, the convolution starts at the top left, and is moved to the right until hitting the edge, then moved one down and from the left to the right again until it hits the bottom right corner. The edges can be handled differently if required, where the values are e.g. duplicated outside of the image (so-called *padding*), so that the values are not empty if the kernel sticks out over the image's edges. The calculation of this two dimensional case between an input I and a kernel K with $K \in \mathbb{R}^{m \times n}$ is as follows. For an illustration of an example see Figure 2.14.

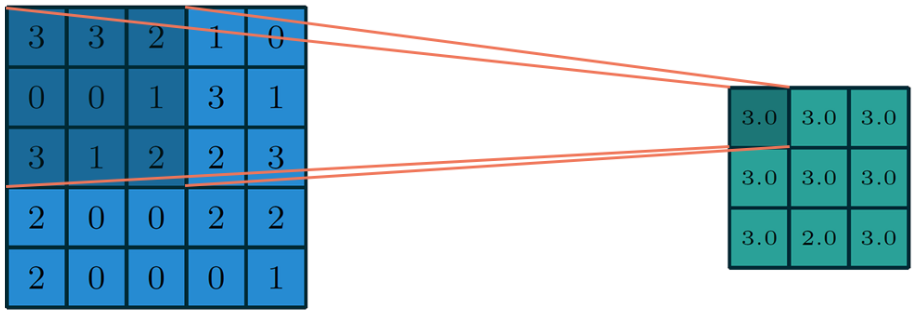

Figure 2.15: Illustration of max-pooling; Dumoulin and Visin, *A guide to convolution arithmetic for deep learning* [100]. The maximum value of the window is selected.

$$R(x,y) = (I * K)(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left(I(x+i, x+j) K(m-i, n-j) \right) \tag{2.18}$$

Pooling is sometimes also referred to as downsampling and is used to reduce the dimensionality of the features and discard superfluous information. In this step, the values of a cluster of defined size of inputs are reduced to one value, according to the type of pooling. For instance, max-pooling selects the maximum of these values, while average pooling takes the average. An illustration of max-pooling can be seen in Figure 2.15. Again, an activation function can be applied on the result. The purpose of pooling layers is to prevent overfitting.

2.4 Evaluation of neural networks

In Section 2.3 we briefly discussed the concept behind neural networks and how they are trained (how they learn). If we have a trained neural network model and we think about using it, another important aspect comes to mind: We need to measure the performance of the neural network.

Generally, when evaluating a neural network, we split our dataset into two parts. A large training set (70% to 95%) and a small validation set (5% to 30%). The training set is used for training the neural network. The neural network runs through all data samples and learns by adjusting the weights through backpropagation. Through vectorization, we can feed the network several data samples at once. Large datasets can be divided into fixed sized batches. Because the backpropagation optimizes the weights in a way, so that the cost function is minimized with regard to the training set, the training set cannot be used for measuring the performance, as the neural network is biased towards this set. Unobserved data, i.e. the validation set is used to accomplish unbiased measurements.

We already introduced the cost function as a way to measure the performance of a neural network over a sample. In the case of a regression problem, a common measure is the mean squared error, while for classification problems, categorical crossentropy can be used. We go over these cost functions more thoroughly in Section 3.6.

In classification problems, there are some additional performance metrics we can measure, usually after the training has happened. We can measure the *overall accuracy*, i.e. the percentage of correctly predicted classes. The *mean/average accuracy* is the average of every accuracy per class. In binary classification, there are four cases: The first (positive) class can be predicted truly or falsely and the second (negative) class can be predicted truly or falsely. *Precision* measures the amount of true positive predictions in proportion to all (true and false) positives. *Recall* (sometimes referred to as *sensitivity*) calculates the amount of true positives in relation to true positives and false negatives. *Specificity* is the number of true negatives in proportion to false positives and true negatives. *Fallout* is false positives divided by true positives and false negatives, *miss rate* is false negatives divided by false positives and true negatives [102].

Training over the whole training set once is called an *epoch*. A neural network can be trained for an arbitrary amount of epochs. If trained for a large amount of epochs, the performance on the training set will likely improve. The performance on the validation set might actually get worse, in a phenomenon called *overfitting*. To prevent overfitting, the number of epochs has to be carefully defined or an approach like *dropout* has to be deployed. Dropout prevents overfitting by randomly dropping nodes and their connections while the neural network is being trained [103].

Instead of splitting the data only once, it can be split into k parts, where each part is used as a validation set, while the rest is used as training set. This method is called *k-fold cross-validation*. Another approach is to set a ratio for training vs. validation data, and then (randomly) select several different splits, test each one and average their performance, called *Monte Carlo cross-validation* [104].

Basically, any statistical indicator can be used for training or evaluating neural networks. One can make use of domain knowledge to create a suitable measure for the problem at hand. In the example of forecasting daily stock returns with neural networks, White [105] suggested evaluating neural networks built for market trading purposes by the profit generated on the basis of their predictions instead of their squared error, which is what we do in this thesis.

2.5 Related work

In this section we will present some related work that is either similar in goal or methodology to this thesis. As stated in Section 2.1.5 there have been many attempts to predict stock prices, some based on neural networks. And while forecasting cryptocurrencies has been tried in the more recent years as well, the attempts have almost exclusively

been focused on Bitcoin only. The following methods are based on machine learning algorithms and use technical data to predict cryptocurrencies.

In their 2015 paper *Automated Bitcoin Trading via Machine Learning Algorithms*, Madan et al. [69] tried to predict the price movement of Bitcoin using machine learning methods. They used support vector machines (SVM), random forests and a binomial generalized linear model (GLM). The authors used several different data sets, daily data of price and information about the blockchain and Bitcoin network as well as 10-minute and 10-second data of the price. The performance of the different machine learning methods was measured in regard to these different data sets. They used a training and validation split of 70% to 30%. The goal of the authors was to predict the sign of the in the price development rather than the price itself, thus reducing the problem to a (binary) classification problem rather than a regression. The GLM performed very well for the daily data with an accuracy of 98.79%, while achieving only 8.5% for the 10-second and 53.9% accuracy for the 10-minute data. The support vector machine was used on the daily data and performed worse with 27.16% accuracy, whereas the random forests performed with 94.98% accuracy for the daily and 57.4% for the 10-minute data. It is worth pointing out that these high accuracies are most likely due to substantial overfitting, since the number of training samples for the daily data is rather low. All in all the authors come to the conclusion that the evaluated methods are suitable for different granularity of data, since the daily data somewhat dampens the high volatility of Bitcoin.

McNally et al. made a similar attempt in their 2018 paper *Predicting the Price of Bitcoin Using Machine Learning* [70]. Their goal was as well to predict the sign of the price development. The used data included price data (open, high, low, close) as well as mining difficulty and hashrate. For feature selection, an algorithm called *Boruta* based on random forests was chosen. For prediction they used a (Bayesian optimized) recurrent neural network and an LSTM network and compared their performance with the ARIMA model. The authors used a 80% to 20% training and validation split and utilized dropout to prevent overfitting. While both the recurrent neural network and the LSTM network were able to outperform the ARIMA model, the LSTM model had both the highest accuracy 52.78% and smallest root mean square error 8%. The recurrent neural network had an accuracy of 50.25%, compared to the 50.05% of ARIMA. The authors also investigated the performance difference of the different models on both the CPU and GPU, where the GPU is significantly faster for training and the LSTM networking taking longer to train. The authors come to the conclusion that LSTM is slightly better for finding long-term dependencies, but does overall not outperform the used recurrent neural network significantly.

In the 2017 paper *Prediction of Bitcoin exchange rate to American dollar using artificial neural network methods* by Radityo et al. [71] the authors tried to predict the closing price of Bitcoin of the next day based on historical daily price data combined with technical indicators. This resulted in a (relatively small) dataset of 1278 rows after the calculation of technical indicators and normalization. The dataset was split into training and validation set by 80% to 20%. The authors used neural networks which were trained

differently, one with backpropagation (BPNN), one with genetic algorithm (GANN), one with a hybrid of those two methods (GABPNN) and the last one getting evolved with neuroevolution of augmenting topologies (NEAT). The performance was measured using the mean absolute percentage error (MAPE). The GABPNN had the best performance, with a MAPE of 1.883(±0.066)%, the BPNN was a close follow-up with 1.998(±0.038)%, while NEAT was at 2.175(±0.096)% and GANN at 4.461(±0.49)%. The author also measured the training time, where BPNN learned faster and needed only around 22.5% of the time of GABPNN. They conclude that for their experiment, BPNN was the best choice.

In the 2018 paper *Anticipating cryptocurrency prices using machine learning* by Alessandretti et al. 1,681 different cryptocurrencies were analyzed [106]. The authors collected the daily average price, volume and market capitalization for a period between November 11, 2015 and April 24, 2018. These prices where then expressed in Bitcoin instead of US dollar, to discard the development of the cryptocurrency market. Three different models where tested, two using (gradient boosting) decision trees and one using an LSTM network. The models were used to predict the currencies with the best return and then split an initial capital to buy a portfolio of the best n cryptocurrencies, where n and other parameters (e.g. time window, number of epochs) were selected based on the *sharpe ratio* and *geometric mean* optimizers. The portfolio's increase in value of the tested period of time is measured and compared to a simple moving average model. All three models are able to outperform the simple moving average model. The LSTM model had the best performance and was able to recognize dependencies over larger windows of time.

Sin et al. published a paper named *Bitcoin price prediction using ensembles of neural networks* in 2017 [107]. The authors used an ensemble of multi-layer feed forward neural networks between different Bitcoin features and the Bitcoin price change of the next day. The used data included 190 different features including price, volume and market capitalization of several different cryptocurrency exchanges as well as many fundamental features about the Bitcoin network, such as hashrate, block size, etc. The dataset consisted of 780 samples. Five different neural networks were created, which differed only in their number of nodes in the two hidden layers. These networks were trained over 30 epochs. After training, the importance of each neural network was determined according to the Genetic Algorithm based Selective Neural Network Ensemble (GASEN) method. The final output is the weighted and averaged output of every neural network in the ensemble. This method was backtested on 50 days not used for training and had a 60% prediction accuracy over that time. The increase in value outperformed the *previous day trend following* strategy used for comparison and increased its value by about 85%, compared to the 38% of trend following.

The presented papers give a broad overview on the current state-of-the-art and the present approaches for this or slightly different problems. Most of these references are inclined towards supporting the possibility of predicting cryptocurrency prices to some degree. The different approaches all have their own advantages and disadvantages. In this thesis, we tried to adopt the good aspects of these papers, while improving on the oversights.

Most of them use a method for selecting good features. However, some of the presented papers suffer from a severe lack of training samples and are highly overfitted. We try to use a larger amount of training samples and make an effort to prevent overfitting with dropout. The LSTM networks were usually the best performing models in these papers, at least out of the neural networks. We focus on three different LSTM neural networks for prediction, one of which also has convolution. We experiment with different preprocessing techniques and use Ethereum instead of Bitcoin. Another substantial difference is, that we use the profit achieved by simulating trading with fees according to the predictions to measure the performance of our neural network models. This is done to see the real-life application of these predictions and to compare models with different targets.

CHAPTER 3

Constructing neural networks for predicting cryptocurrency prices

In this chapter we discuss the concept and the methodology used in the practical part of this thesis. In Section 3.1 we outline the concept and goal. Section 3.2 goes over what kind of data were chosen as input features and why. Section 3.3 lists the different targets that we want our neural networks to predict as output. In Section 3.4 we present the steps taken for preprocessing the data. Section 3.5 focuses on the architecture of the neural networks and how we train them. Eventually, Section 3.6 describes how we measure the performance of the neural networks.

3.1 Concept

For this part of the thesis, the general objective was to create a modular software tool which allows for quick prototyping of different neural network models, data preprocessing techniques and targets. The models try to predict the price or other trading signals for any cryptocurrency based on the preprocessed historic price data. The goal is to find the model and preprocessing combinations that are the most profitable ones when trading based on the model's predictions. The main feature requirements can be summed up in the following list. The tool has to ...

- collect raw historic data of any cryptocurrency.
- extract features and build targets from this data.
- preprocess the data.
- provide means for easy implementation of new neural network models.

- train these models on the preprocessed training set.
- evaluate models based on a measure.

With the completed tool, several different configurations can be tested and evaluated quickly. To retain a manageable workload we need to narrow down some parameters. For the cryptocurrency to predict we selected Ethereum for several reasons. Ethereum has a very promising concept and a large market capitalization. Most of the similar research has been done on Bitcoin. We wanted to see if Bitcoin influences the Ethereum price.

The next steps are as follows and will be described in-depth over the rest of this chapter. We gather the historic price data of Ethereum and Bitcoin in minute intervals, extract features, implement four different targets, six different preprocessing combinations and three different neural networks, two using LSTM cells and one using an LSTM and CNN hybrid. We use these building blocks for carrying out our evaluation (see Chapter 5) in order to answer the questions asked in Section 1.2. We are mainly interested in the question if we can predict trading signals for Ethereum in a way so that we can derive a profitable trading strategy from these predictions.

3.2 Input data

Choosing the input data for a neural network is an essential step. The importance of a large number of training samples along with good features cannot be overstated. The outline of this thesis already narrows the data down, but there are still several other aspects to be considered. Our endeavor is based on the assumption that the price of cryptocurrencies is influenced by previous data or information. Therefore the predictions are based on sequential historic data, which fits our use of LSTM neural networks to find these temporal long-term dependencies. For this historical data we need to set a time range in which the data lies and the frequency for how far the single data points are apart in time.

The paper by Schulmeister [45] mentioned in Section 2.1.4 suggests that (for stocks) the profitability of trading approaches that exploit profit opportunities based on longer time intervals decreases. This along with the low trading fees and fast APIs made us settle for a one-minute frequency for our data, i.e. the data we are going to feed to our neural networks is from every minute of the chosen time range.

Another important decision was what different kinds of data we deem influential on Ethereum. Although there may be more, we have pinpointed three different categories in Section 2.2.3 which are possibly influential on cryptocurrency prices (or Section 2.1.3 for stock prices). To quickly recapitulate, these categories are the following.

- **Technical data**

  Historic prices (open, high, low, close), trading volume and number of trades of every interval until now.

- **Fundamental data**

  Data about the cryptocurrency network, such as hashrate, number of transactions, number of miners, etc.

- **Sentiment data**

  Information about the sentiment people have about the cryptocurrency. This can be from newspaper articles, Tweets, web searches, etc.

In this thesis we will consider mainly technical data. For our chosen one-minute intervals, technical data is easily available and we use the API of Binance to collect it [108]. Fundamental data is also broadly available on popular sites (e.g. the *Block explorer* [109]), however the granularity of it is too coarse. This site provides the data daily, so heavy interpolation would have to be applied to fit the granularity of the technical data. Therefore fundamental data is not used at all in this thesis.

Sentiment data has been shown to be relatively powerful for price prediction (Sections 2.1.5 and 2.2.3), and while text sentiment analysis tools are widely and freely available, the aggregation of relevant historic news or Tweets is very time-consuming and expensive. So despite sentiment data being potentially useful, we will also not use it.

We will however investigate the claim, that as a currency, Bitcoin represents the general mood about and acceptance of cryptocurrencies. Because it is still the most popular cryptocurrency in market capitalization as of January 2019 [57], Bitcoin is said to have influence on other cryptocurrencies. If that claim is true, the Bitcoin price has an inherent sentiment value for other cryptocurrencies. To examine this statement, we will include the Bitcoin closing price in our data and see if the predictions improve compared to not including it.

The amount of data is of essence to any machine learning problem. Having a large number of training samples is beneficial to the training process. As a range for our data we chose August 2017 to December 2018. This range was chosen both out of convenience, as it was the earliest available data on Binance, and because we wanted recent historic price data, which may depict the current situation more accurately than older data. Together with the one-minute intervals this results in roughly 705 000 data samples.

All in all, the following features were selected: open, high, low, close, volume of Ethereum as technical data and optionally the Bitcoin closing price data with supposed inherent sentiment value. Other technical data, such as number of trades, and the fundamental data were discarded and (other) sentiment data was not fetched. Every data sample contains values for all the selected features. The target generation process is shown in the next section.

3.3 Target

At this point it is still unclear what output we want our neural network to produce. In supervised learning the training process consists of providing the model with input/output pairs and inferring a function that best maps this input data to the given output data. The input data has already been explained in the last section. The output, or target, will be generated from this input data. There are several approaches to take, reducing the problem to either a regression or a classification problem.

The first and possibly most straightforward approach is to define the closing price in n time steps as target and try to predict it directly. This regression approach is easy to calculate. However, there are other ways that work as well. Another simple one is to generate a target which indicates the direction of the price, i.e. if the price is going to rise or fall in the future. This target reduces the problem to a binary classification problem.

The idea is to test if some targets perform better than others for this problem and if yes, which ones they are. The targets are defined and created dynamically. The neural network models are then dynamically altered, so that their output layer accommodates these targets and has the right amount of output nodes, i.e. one output node for regression and two (or more) for classification. Four different targets are tested, including the two briefly mentioned above. They are listed below, briefly explained, possible advantages and drawbacks are discussed and finally, we describe how a trading signal can be generated from them. We will henceforth refer to them by these given names.

- **Binary classification**

  This target represents the direction of the price. The idea is to have the network predict two different classes, namely whether the price is rising or falling in n time steps. These classes can be directly translated into the trading signals *buy* and *sell*. This results in a simple prediction for the neural network with just two possible outcomes, producing a rather high prediction accuracy, since random guessing already has a 50% chance of being correct. Even though there is no *hold* signal, this is obviously accounted for because *buy* amounts to *hold* when one already holds the asset, while *sell* amounts to *hold* when one does not hold it. What this target fails to achieve, however, is to incorporate a trading fee, because even for price jumps smaller than the fee, the target would recommend a *buy* or *sell*.

  For generating this target the price is shifted n time steps and a boolean function is applied. For a given time m, the target will be 0 if the price at time $m + n$ is smaller (=sell) or 1 if it is bigger (=buy) than the price at time m.

- **Regression of raw price**

  With this target, the network tries to predict the (closing) price in n time steps directly. To generate a trading signal, the price prediction merely needs to be compared to the current price. To incorporate fees or reduce the number of trades, a certain threshold can be defined. If the change is smaller than the threshold, the

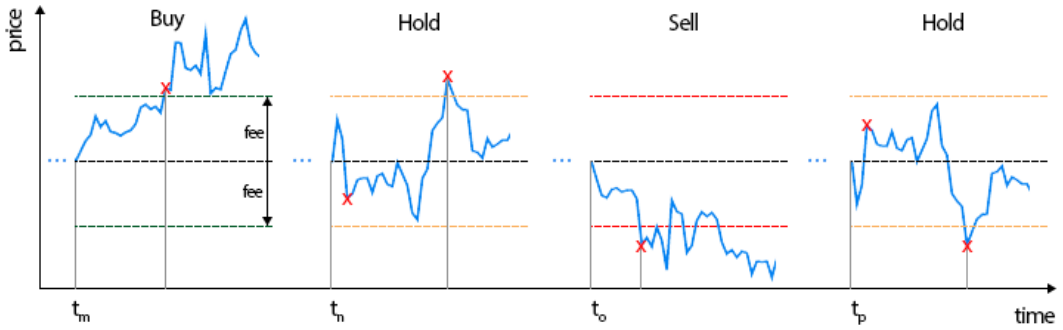

Figure 3.1: Illustration of best strategy target generation for some points in time t_m, t_n, t_o, t_p, each representing one of the four main cases. The fifth case happens at the end of the data: if the price change is never larger than the fee, then the decision is to hold.

trading signal *hold* is produced, otherwise *buy* or *sell* is produced depending on whether the price rises or falls.

For generating this target the price is shifted n time steps. For a given time m, the target will be the price at time $m + n$.

- **Regression of relative price change**

  This target is similar to the last one, but the network tries to predict the relative price change in n time steps instead of the price directly. For generating this target the relative price change is calculated and then shifted n time steps.

- **Best strategy classification**

  For this target, at each time step the best trading decision for a given fee is calculated (buy, hold or sell) and assigned as either 0, 1 or 2 respectively. The network tries to predict these best trading decisions at each point in time. This makes it easy to incorporate trading fees. In this model however, the percentage of the right predictions the model makes is not the exact accuracy, because if the (crypto) asset is held, buy and hold amount to the same, whereas if the asset is not held, sell and hold amount to the same operation.

  To generate this target the (training) data is analyzed as follows. For every data record it is checked whether the price first goes above the current price plus the fee or below the current price minus the fee. If none applies, e.g. at the end of the data, the decision is *hold*. If it is higher and there is no smaller price between the current price and it, then the decision is *buy*. If a smaller price is in-between, it is *hold*. The same works analogous for *sell*. Figure 3.1 illustrates this procedure, Algorithm 3.1 demonstrates in detail how the calculation works for one price.

Algorithm 3.1: Function that generates the best trading signal for a price, called on every price, see Figure 3.1.

```
Input  : price p at time t and a list of all prices priceList
Returns: best trading signal at time t
for every price q in priceList between p and priceList.length do
    if q > p * (1+fee) then
        for every price r in priceList between p and q do
            if r < p then
                return HOLD;
            end
            return BUY;
        end
    else if q < p * (1-fee) then
        for every price r in priceList between p and q do
            if r > p then
                return HOLD;
            end
            return SELL;
        end
    else
        continue;
    end
end
return HOLD;
```

3.4 Preprocessing

The feature values themselves generally span over various different ranges. Preprocessing the data is an essential step to scale the input data to a common range that is not larger than the neural network parameters. This helps to improve the learning of neural networks [83]. The tool should make the integration of new preprocessing techniques into the workflow simple. We implemented the following preprocessing techniques in the software and experimented with them, to see how they impact on the performance of the neural networks.

The first normalization decision that can be taken in this tool is whether or not to transform the input data to the percentage change with regard to the immediate previous value. Only suitable data is transformed here, since this method on its own cannot deal with zero values, like the trading volume. The calculation for a transformed value x'_i at time i is as follows.

$$x'_i = \frac{x_i}{x_{i-1}} \tag{3.1}$$

The second step is to decide on one of following three further normalization techniques. This normalization is then performed on all values for every feature individually, regardless of the previous step. These scaling techniques are commonly implemented in machine learning libraries (e.g. scikit-learn [82]).

- **Min-Max scaling**

  Every feature is scaled to a common range between a and b, e.g. to be between -1 and 1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} * (b - a) + a \tag{3.2}$$

- **Standard scaling**

  Every feature is scaled to have a mean of 0 and a variance of 1.

$$x' = \frac{x - \overline{x}}{\sigma_x} \tag{3.3}$$

- **Robust scaling**

  Every features is scaled to the interquartile range, i.e. to be between the first and third quartile, in a fashion analogous to the first method. This is method of scaling is more robust to outliers.

$$x' = \frac{x - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{3.4}$$

The combination of these two steps results in six possible ways to preprocess the data. We test these combinations at a later step to see, if they impact the performance.

3.5 Neural networks and training

To measure the performance of different neural network models, an important requirement for the software tool is to make the creation of new neural networks and their integration with the different targets and preprocessing techniques simple and straightforward. To achieve this, the output layer is dynamically adapted depending on the used target. The regression targets have one output node, while the classification targets have two output nodes for the binary target or three output nodes for the best strategy target.

Three models are then created for further evaluation. All of them are LSTM-based neural networks, with one of them being a mixture of LSTM and CNN. The networks all have dropout after the LSTM layer to prevent overfitting. After the LSTM and dropout layer there is a batch normalization layer, which normalizes the data within batches to speed

up training [110]. After these layers, there is one dense (or fully connected) layer followed by an output layer.

A *time window* (or *sequence length*) of 200 was chosen. The inputs are fed in batches of 64. The activation function of the LSTM is the *tanh* function, as shown in Section 2.3.5. For the output layer a linear (for regression targets) or a softmax (for classifcation targets) activation function is used. The architecture of these models is as follows and can be seen in Figure 3.2.

- The first neural network model used is a rather basic one. It consists of one LSTM layer with 128 units, followed by a dropout layer and a batch normalization layer. Afterwards there is a dense layer with 32 nodes and a rectifier activation function as well as an output layer with one, two or three nodes, depending on the target. The second and the third model expand on this architecture.

- The second neural network consists of an LSTM layer with 128 units, with dropout and batch normalization repeated three times. This is followed by a dense layer with 32 nodes and a rectifier activation function and the output layer.

- The third model adds a one-dimensional convolution layer with kernel size 3 and a rectifier activation function. This is followed by a one-dimensional max pooling layer with a pooling size of 4. After this, the model consists of an LSTM layer with 128 units, dropout and batch normalization. Finally, there is a dense layer with 64 units and a *leaky* rectifier activation function ($f(x) = \max(x, 0.001x)$) and the output layer.

There is no definitive best neural network architecture for a problem, nor are there final rules for constructing one. The models are usually created intuitively and by experimenting. We created the first model to be a minimalist LSTM network. Based on this model, we created several others, each altering one aspect, such as the number of layers, adding convolution, changing the number of nodes, etc. We did some brief testing on these models and selected two of them (model 2 and 3) for further evaluation.

These models are trained using the supplied, preprocessed data on a target. As a cost function, the mean squared error (MSE) is used in regression and the categorical crossentropy for classification. The training happens through backpropagation using an optimization algorithm called Adam [111]. This method keeps an adaptive learning rate for every weight and is an improvement over conventional gradient descent.

Only a certain amount of the data is used for training, in our case 95%. The rest is used for validation and simulation. The neural networks are trained for 10 epochs or more. The idea is to save the weights of the model after every epoch. This is useful for reconstructing the models at any training stage later on and to evaluate the performance of every processing step, using the measure defined in the next section.

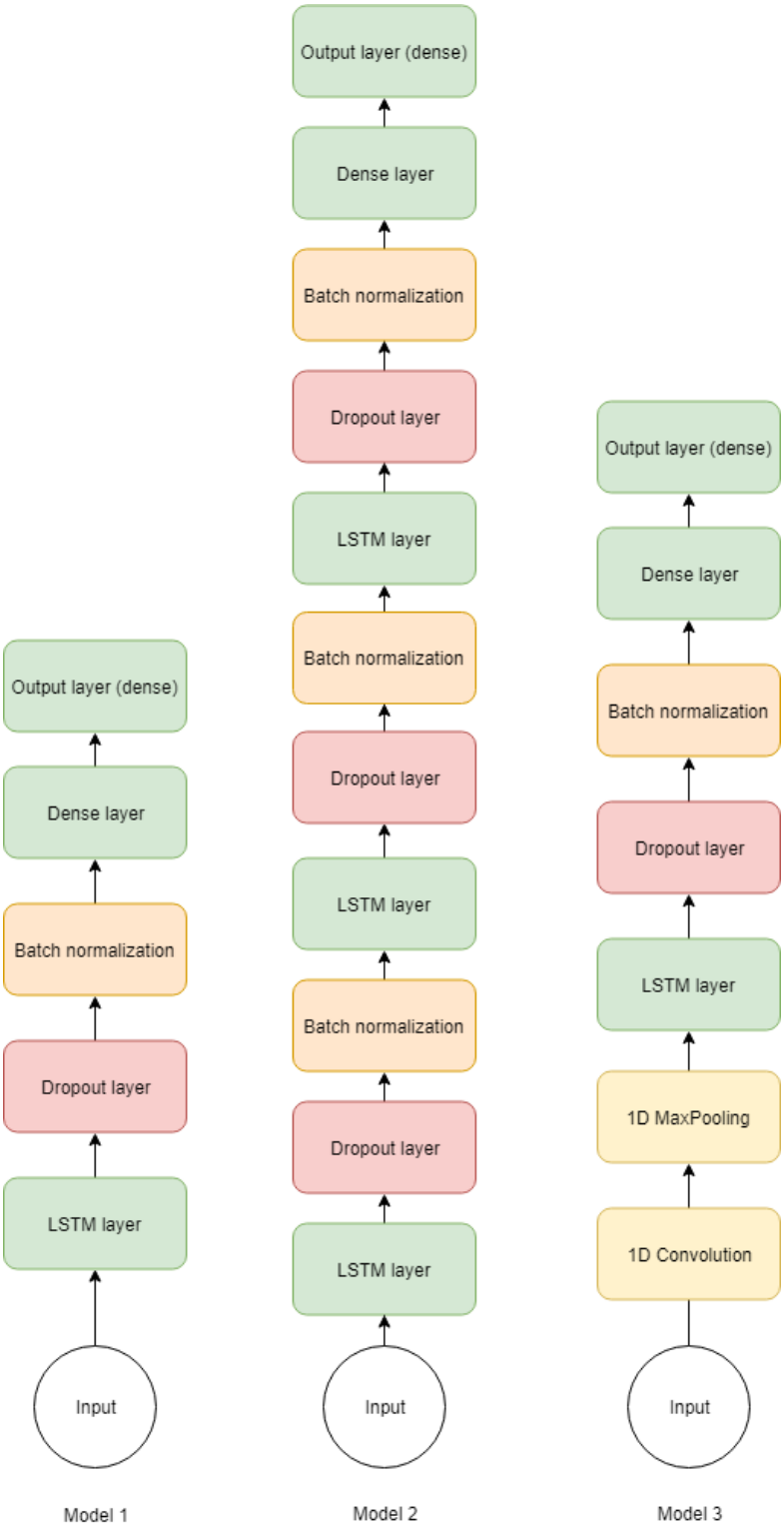

Figure 3.2: Architecture of the three different models used.

3.6 Performance measurement and testing

In Section 2.4 we already gave an overview on how to measure the performance of a neural network. The cost function is used for backpropagation and is a measure for neural networks. This cost function can vary and is usually different for classification and regression problems. In our case, we have targets for both classification and regression. In regression we use the MSE as cost function, in classification we use the categorical crossentropy. They are calculated as follows:

- Mean squared error (MSE):

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left((y^{(i)} - \hat{y}^{(i)})^2\right) \tag{3.5}$$

  where $y^{(i)}$ is the value of the label of the i^{th} input sample and $\hat{y}^{(i)}$ the predicted output value.

- Categorical crossentropy:

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\hat{y}^{(i)}\right) \tag{3.6}$$

  where $y^{(i)}$ is the label of the i^{th} input sample, a vector of size C and C is the number of different categories. $y^{(i)}$ consist of only 0s except for the value at the location of the category that it actually is, where the value is 1. $\hat{y}^{(i)}$ denotes the predicted output vector of size C, containing the probabilities that the output is of the respective category.

The MSE is the average squared difference between the prediction and the actual label, while categorical crossentropy is a measure of how much the predicted probabilities for each class and the actual class differ. In the first case the output $y^{(i)}$ and the prediction $\hat{y}^{(i)}$ are scalar values, in the second case they are vectors. Using MSE for classification problems or categorical crossentropy for regression makes only little sense. Another performance measure that works only for classification is *accuracy*, where the percentage of the correctly predicted cases is calculated. So, even though these are common measures of performance for neural networks on their own and are useful for training, we want to introduce another metric to evaluate and compare our neural networks despite having different targets.

This measure works as follows. For every data point in the validation data, the trained model is used to predict its target. Afterwards, a trading signal (buy, hold or sell) is generated from this prediction. The simulation starts with a value of 100 US dollars and will simulate trading based on the trading signals at each time step. For every trade that

is executed, a fee is deducted from the simulation value. At the end, the simulation will have a value which is then compared to the actual price change of the cryptocurrency. Despite being able to compare the performance of our various models with different targets, this metric has the added benefit of seeing the real life viability for this form of automated trading. Evaluating a model intended for trading purposes by profit was also suggested by White [105].

The following two different outcomes are desirable when evaluating a model in such a simulation, ideally both occurring at the same time.

- **Outperforming the price development.**
- **Achieving a positive simulation outcome, i.e. more than 100 US dollars.**

Achieving at least one of these outcomes at all times is desirable. However, the simulations are carried out over periods of time where the Ethereum price moves upwards, downwards or stays the same. It might be easier to outperform the price when the price drops immensely and it might be easier to achieve a positive simulation outcome when the price increases significantly.

Since our work is a simulation, assumptions and simplifications have to be made. The data records we used are in one-minute intervals, so there are no values in-between, which is obviously inaccurate. Trades can be carried out at any time and there is a price movement within one minute. In the simulation, every trade is carried out using the currently latest closing price. Furthermore every trade order is filled immediately at that price, i.e. it cannot happen that an order is not filled. This is another inaccuracy, as there can be trades that are filled after some delay, not at all or that buys (sells) are only filled at a higher (lower) price than the last market price. Finally, the simulation does not take into account any impact its own trades might have on the market.

CHAPTER 4

Implementation

In this chapter we present the technical details on how the design of Chapter 3 was implemented in code. In Section 4.1 we briefly go over the requirements for the technologies used in our software, which suitable technologies are available for the task and which technology stack we chose based on these requirements. Section 4.2 focuses on the program architecture. Section 4.3 goes over the computational effort and the hardware setup used to cope with it.

4.1 Selection of technologies

We have described the models of artificial neural networks in Section 2.3. As they consist mostly of matrix operations, implementing simple neural networks in a naive way may appear straightforward. The complexity of more advanced neural networks such as LSTM networks or improved backpropagation makes implementing them significantly harder, especially if the computation should be done in an efficient way on graphics cards. Thankfully, there are numerous different frameworks for deep learning which implement different concepts very efficiently and make the construction of neural networks simple. In the next sections we will briefly go over our requirements for these frameworks and other technologies, compare them and list the ones we chose for implementing our tool.

4.1.1 Requirements

The most important requirements we have for our deep learning framework are, that it has implementations for the concepts we intend to use. These concepts are LSTM, convolution and pooling (CNN), dropout and, of course, standard, fully connected neural network layers with different activation functions as well as efficient backpropagation algorithms. These implementations have to be as efficient as possible, in order to deal with our dataset. The framework should be easy to use and have a large developer community

behind it, for support reasons, e.g. for finding helpful comments on a framework-specific problem online. While implementing the different normalization approaches ourselves is definitely possible, existing and efficient implementations are helpful. All other employed technologies should be easy to integrate with this framework.

4.1.2 Comparison of available technologies

At the time of writing this thesis, most deep learning frameworks use Python as a programming language. We will list some popular ones and briefly go over the advantages and disadvantages for each one [112][113]. Note, however, that this is a very subjective matter, where opinions differ significantly. There exist also many more frameworks than the ones listed below, which we did not consider for space reasons.

- **TensorFlow** [114]:

  This framework was developed by Google and released under open-source license in 2015. It is the most popular deep learning framework (see Figure 4.1) with a large community as well as companies behind it. It supports all relevant kinds of neural network components, training algorithms and so on. In TensorFlow, developers can build computational graphs in Python, which are then efficiently executed as native code using CUDA for GPU computations.

- **PyTorch** [115]:

  This open-source framework, backed by Facebook, is the successor of the library Torch. There are also large companies that use it and the community behind it is growing, but nowhere near as large as for TensorFlow.

- **Caffe** [116]:

  Caffe was developed at Berkley University. Its successor Caffe2 was integrated in PyTorch in 2018.

- **Theano** [117]:

  As one of the earlier Python based open-source deep learning frameworks, Theano was popular following the years after its initial release in 2007. It is however not in active development anymore, which is why we have decided against using it.

- **MXNET** [118]:

  This Apache open-source project is allegedly scalable and performant, but not very popular in the deep learning community.

- **Microsoft Cognitive Toolkit (CNTK)** [119]:

  This is an open-source framework by Microsoft. It supposedly has good performance and efficiency, but a smaller community behind it.

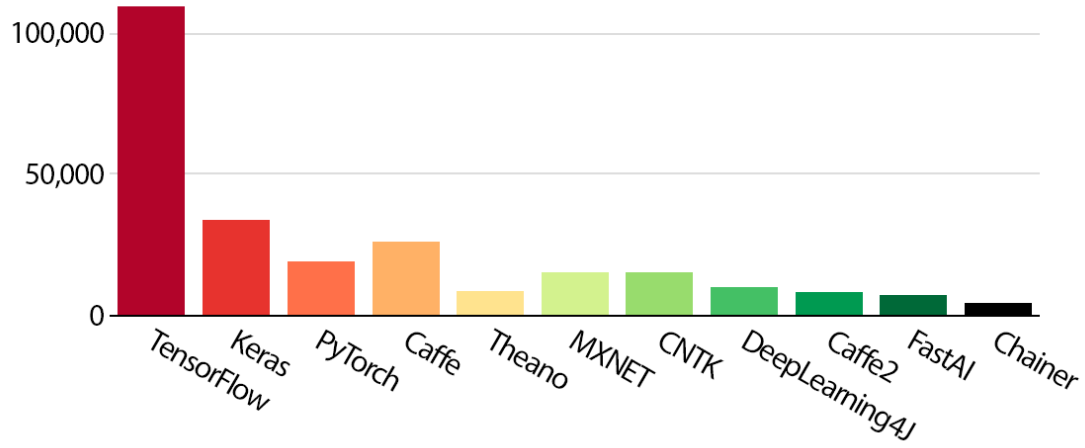

Figure 4.1: Popularity of deep learning frameworks measured in stars given to each project on GitHub; Jeff Hale, *Deep Learning Framework Power Scores 2018* [112].

- **Deeplearning4j (DL4J)** [120]:

  This framework is based on the programming language Java and runs in the Java virtual machine (JVM). It supports different neural network types and can interact with TensorFlow and Keras. However, the community behind it is not very large.

- **Keras** [121]:

  Keras is a Python library that uses other deep learning frameworks as backend. Choices for backend include either TensorFlow, Theano or the Microsoft Cognitive Toolkit. With Keras, neural networks can be constructed in a simple and user-friendly fashion and it helps to reduce the lines of code.

As most of these frameworks offer implementations for every neural network component we need and all claim to be fast and efficient, we settled for one that is user-friendly and has a high popularity for community support. The popularity of the different frameworks measured in GitHub stars is shown in Figure 4.1 [112].

4.1.3 Used technologies

Based on our requirements, we ended up choosing Keras with TensorFlow as backend for our deep learning framework. Therefore Python is the programming language of our choice and we selected the rest of the technologies to fit.

For preprocessing we use *scikit-learn* [82], which is a Python package for machine learning. For other tasks we chose appropriate Python packages. The following technologies were used to build the prognosis tool:

- **Python 3.6.6** as programming language.
- **TensorFlow 1.12 GPU** (with Keras) to build and train the neural networks, using CUDA 9 and cuDNN 7 for hardware acceleration.
- **scikit-learn**, a python package for data preprocessing.
- **pandas**, a python package for data representation and handling [122].
- Several other python packages such as **numpy**, **joblib**, **requests**, **websockets** and **asyncio** for tasks such as fetching the historical data from Binance's public API.

4.2 Program architecture

Our tool was designed with modularity in mind, to make the evaluation of various models and preprocessing techniques simple. The different tasks the tool has to carry out and the general workflow are as follows. First the data is fetched from the API and stored. This data is then preprocessed and a neural network is trained on it. Afterwards, a simulation is carried out to evaluate its performance. The target, model and preprocessing techniques are selected beforehand in a run configuration along with some other parameters. Via an automation, many different parameter combinations can be tested automatically. We outlined this workflow in Figure 4.2 and will describe the steps in detail over the next few sections.

4.2.1 Data fetching

This module of the tool is used for fetching the historical data. It uses the Python library *requests* [123] to send http requests to the Binance public API [108]. The user provides a symbol, for example "BTCUSDT" – the price of Bitcoin (BTC) in US dollars (or rather a cryptocurrency tied one-to-one to the dollar called Tether), and the module will fetch historical data for that symbol.

By default, the module will fetch this data for a time range between the current date and the earliest possible date of the Binance API, which is August 2017, in one-minute intervals. These parameters can be tweaked easily to fetch different data. The data that is gathered for every minute contains many different fields: *open time*, *open*, *high*, *low*, *close*, *volume*, *close time*, *quote asset volume*, *number of trades*, *taker buy base asset volume* and *taker buy quote asset volume*. As Binance limits the number of data points that can be fetched per request to 1,000, several requests are made and the data is then combined locally.

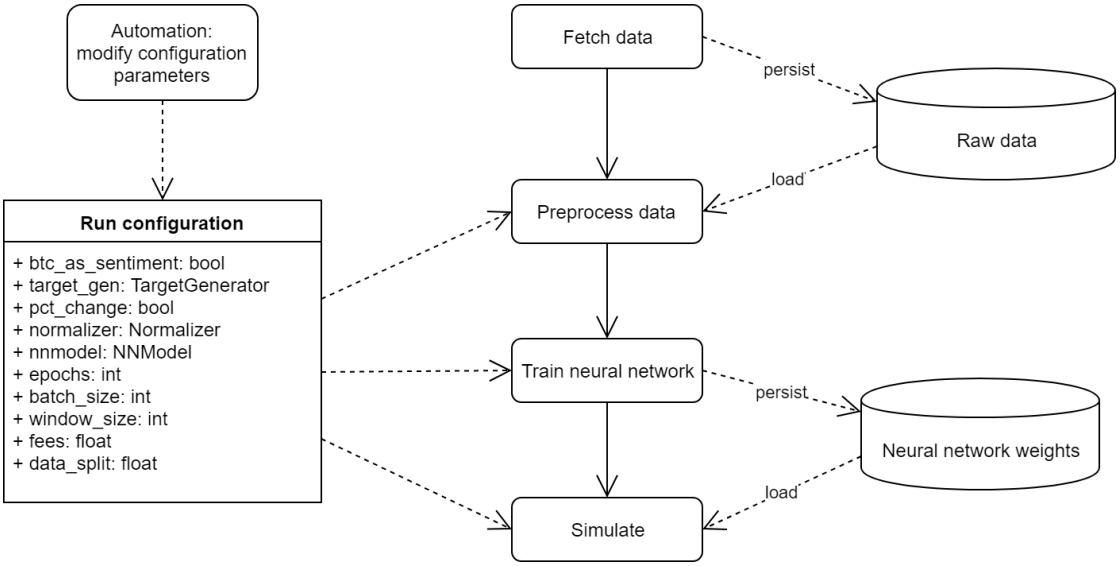

Figure 4.2: Visualization of the workflow of the implemented software.

The requests are made in a ways so that the API limit for the maximum amount of requests per second is not exceeded. The fetched and combined data is sanity checked for holes or other irregularities. For instance, we check if the data points are really exactly one minute apart, and fix them if necessary. The data is saved as a *comma separated value* (csv) file. With the one-minute intervals and the time range we used, these files are roughly 60 MB of size. Please note that we did discard irrelevant fields to save space. An irrelevant field is for instance the second *time* field, because the two time values will always be one minute apart.

This process of fetching the whole history of one symbol is not computationally expensive and takes only a few minutes to complete. This module of the software can be executed completely independently from the rest of the program, since the data is persistently saved and can be accessed later on.

4.2.2 Data handling

This module loads the saved csv file fetched by the previous module and is responsible for three steps. It extracts features from the data, generates a target and normalizes the data. The features are extracted by dropping the fields that are deemed unnecessary and, optionally, by adding the Bitcoin closing price, leaving us with the features which were selected in Section 3.2.

As there are different targets, there are different target generators. To make the generation of targets as modular as possible, we used the inheritance mechanism of object oriented programming. We defined an abstract superclass with the necessary methods for creating the target. These methods are implemented in the various subclasses, one for each

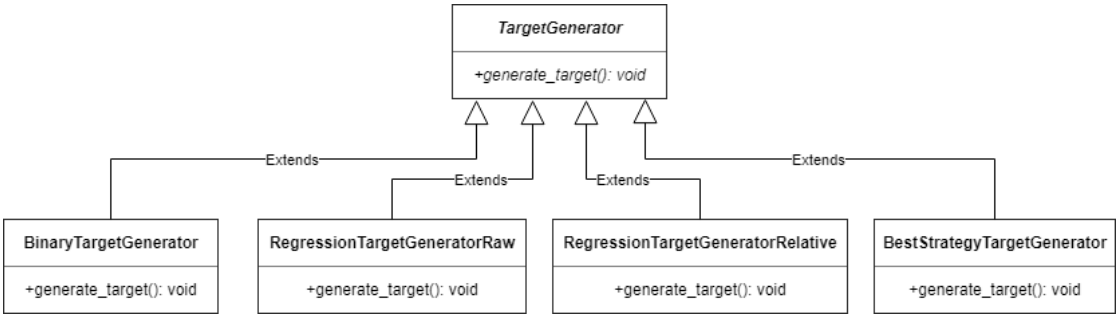

Figure 4.3: Class diagram of the target generation inheritance.

target. With this approach we can use any target generator interchangeably instead of the superclass and constructing new target generators becomes easy. For creating a new target, that target generation mechanism merely has to be implemented as a subclass of that superclass and is ready to be used. The architecture is illustrated in Figure 4.3. We described the different targets in Section 3.3.

The first step of the normalization explained in Section 3.4 is implemented as a simple function calculating the percentage change of the value. The second part of the normalization (or scaling) was built similarly to the target generation. Here we also use inheritance to define which methods we need for normalizing in an abstract superclass. These methods are implemented in every subclass. Analogous to target generation, this makes adding new ways of normalization simple. The architecture of this component is shown in Figure 4.4.

As seen in the figure, we require two different normalization methods. The first is for scaling the training set, the second for scaling the validation set. This is necessary, because we need to scale the validation data exactly like the training data. Imagine a feature of the training data that has a certain range and is scaled to be between 0 and 1. Now the maximum value of that feature in the training data corresponds to 1 and the minimum to 0. This feature might have a different range in the validation data, and the maximum (minimum) might be bigger (smaller) than in the training data. Yet, to avoid biased results, it has to be scaled exactly to the same interval as the training data, so that the maximum (minimum) of the validation data might be scaled to a value larger than 1 (smaller than 0). The configuration of the normalization of the training set is therefore saved and reused when normalizing the validation set.

4.2.3 Neural networks, training and simulation

To make the creation of new neural networks simple, we yet again used a similar inheritance architecture as with the target generation and normalization. The neural networks themselves are constructed using Keras' Sequential model API. This is a very convenient way to construct neural networks, as the layers can be added one by one along with their parameters, such as nodes, activation functions and so on. Depending on the

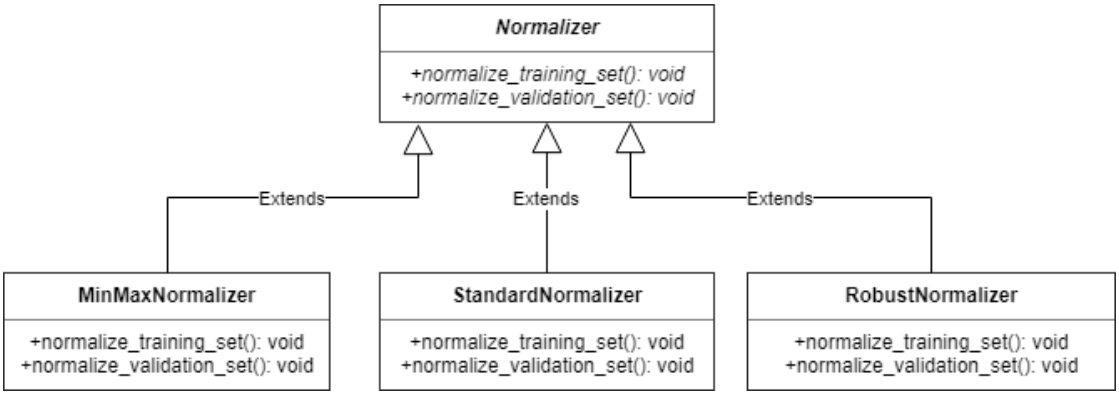

Figure 4.4: Class diagram of the normalization inheritance.

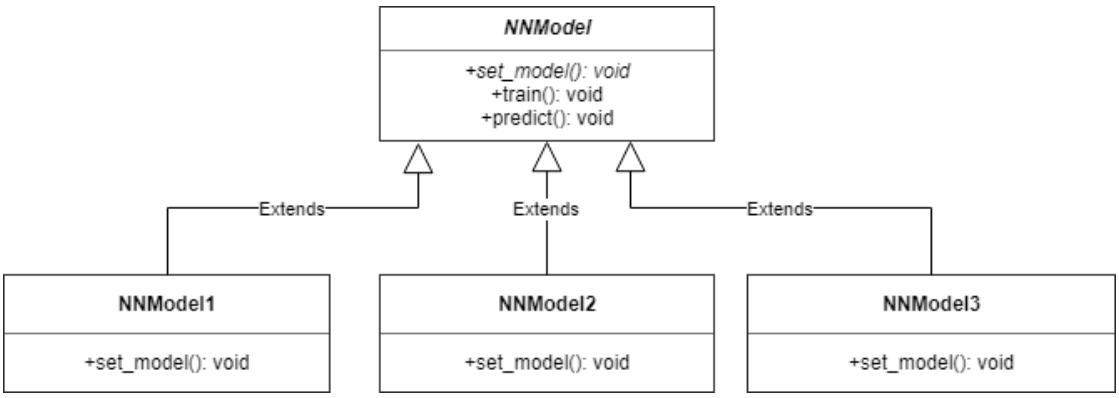

Figure 4.5: Class diagram of the neural network inheritance.

target, we need one output node (regression) or two to three (classification). Therefore, the output layer along with its activation function is created dynamically, according to the target that was defined earlier.

The superclass provides methods for training and prediction, which are used on the model that is set automatically according to the subclass. The *train* method uses the supplied data to train the neural network, whereas the *predict* method is used to predict outputs with a trained network, e.g. for simulation. This architecture is shown in Figure 4.5, the structure of the neural networks was explained in Section 2.3.

Now that we have modular target generation, preprocessing and neural networks, we need a way to combine them. Before starting the training with this software, the user defines a run configuration. In this configuration, the user simply selects the target generator, preprocessing combination and neural network of his choice along with other parameters such as the number of epochs, the batch size, etc.

Afterwards the user can start the main component of the tool with the parameter “train” in order to start the training process. The preprocessed data is fed to the neural network,

which trains for the provided number of epochs. The weights of the trained models are saved after every epoch so that they can be reused for further training or for simulation.

Running the same component with the parameter "simulate" results in starting a simulation. For this, the same run configuration as with the training process is used. An epoch number can be provided to use a specific training state of the neural network, otherwise the latest model is used. The weights are loaded and the simulation starts. The simulation is performed in a function, where according to the prediction of the neural network, the corresponding trading action is selected. This is also done dynamically according to the target and does not have to be set by the user. The simulation starts on the validation data, which was not used for testing, with a value of 100 US dollars. The simulation goes over every sample and carries out the trading decision. Buying or selling is done at the price currently gone over by the simulation and a fee set in the configuration is deducted. Finally the simulation is visualized in a graph.

4.2.4 Test automation

In order to test several different configurations in a convenient way, we required test automation. In this part of the software, a range of values for all the different parameters in a run configuration can be defined, and all possible combinations are tested automatically. The software trains and tests the model for each configuration and saves the results for every epoch along with the run configuration settings. Furthermore, we defined an API, that allows a socket connection with another program. Through this API, the software can be supplied with (real) data and sends back a trading signal, which the other program can process further.

4.3 Hardware and computational effort

The computational effort for machine learning and specifically for neural networks is high. The increase of computational power is one of the reasons why neural networks have become popular over the last years. Still, for models trained on a large amount of training samples, the effort remains high.

Neural network calculations are mostly matrix operations, which is why *graphics processing units* (GPUs) are usually far more efficient than CPUs. As of now, most frameworks for deep learning support nVidia's *CUDA* API. The interface OpenCL, which for instance AMD's GPUs use, is usually not supported. This is the reason why mainly nVidia GPUs are used for deep learning.

Carrying out the calculation on the GPU and utilizing CUDA enables us to use the faster LSTM implementation *CuDNNLSTM*, based on nVidia's *cuDNN* library. This decreased the training time from around 1 hour per epoch to around 5 minutes per epoch for the simplest neural network (model 1) and our number of training samples.

The evaluation done in this thesis was performed on a single consumer workstation. To cope with the computational effort, the following hardware setup was used.

- **GPUs:** 2 × nVidia GeForce GTX 1080
- **CPU:** AMD Ryzen Threadripper 1900X, 8-Cores with 3.8 GHz each
- **RAM:** 32GB DDR4 RAM

A dual GPU setup was chosen, so that two models can be trained in parallel at the same time. Despite the use of CuDNNLSTM and this hardware setup, which was rather high end for a consumer workstation at the time of writing this thesis, along with other measures taken to reduce the workload (see Section 5.3) the experimentation required a duration of around 2-3 weeks computing non-stop, 24 hours a day.

CHAPTER 5

Evaluation

In this chapter we discuss the performed experiments and their results. The goal of the evaluation is to see if neural networks can predict profitable trading signals for Ethereum. We test different preprocessing methods, neural network model and target combinations in order to compare their performance among each other as well as to state-of-the-art solutions. Section 5.1 describes the setup of the experimentation. Section 5.2 introduces three different sectors where tests are carried out. In Section 5.3 we go over the possible combinations and the approach for reducing the large search space. Section 5.4 shows how the best preprocessing combination was found and chosen, while Section 5.5 goes over the performance of the different models. Section 5.6 lists and discusses the main results, while Section 5.7 shows the results when training over more epochs. To see all results in detail along with an explanation for the used abbreviations in tables, we refer the reader to the appendix.

5.1 Setup

We carry all experimentation out on historical data. To briefly recapitulate, this historical data consist of the price of Ethereum (ETH) in US dollar (USD) and its trading volume on the cryptocurrency exchange *Binance*. The data ranges from August 2017 to December 2018 and is in one-minute intervals. This results in about 705 000 data records. In the experiments, we consider the following features.

- o: Opening price of interval
- h: Highest price of interval
- l: Lowest price of interval
- c: Closing price of interval

- v: Trading volume of interval
- Optional: Bitcoin (BTC) closing price as "sentiment" value

This data are used for training and simulation. We will see how the simulation based on the predictions of all the trained models performs in relation to the actual price development. The number of trades a model carries out in the simulation is also closely examined. Any model which has not more than one trade will be regarded as a failure. If all epochs of a preprocessing and model combination are such failures, the whole combination is disregarded. We use the measured performance to find the best preprocessing model combinations or at least the ones that learn quickly while still providing good results.

5.2 Sectors

The models were trained on 95% of the data, while the rest of the data was used for validation and simulation. The training was carried out in 10 epochs each, the trained model after every epoch was saved. Every saved model was then used to simulate trading on the validation data, i.e. the remaining 5% of the data. The model provides a trading decision for every minute of the test data. It is assumed that every trading decision is carried out without delay at market price, i.e. the current closing price. For every trade, a fee of 0.075% is deducted from the simulation value, which starts at 100 USD.

The models might perform differently depending on the market situation of the testing data, i.e. its performance differs in a period where the ETH price is rising rapidly (*bullish*) compared to when the price is dropping (*bearish*) or stays more or less the same (*stagnating*). To make the results more conclusive, the experiments were carried out three times. Of the historic price data, three different 5% regions or *sectors* are selected. The price progression of ETH in USD is shown in Figure 5.1 along with these sectors. All of the three different 5% data sectors were used as validation data, while the remaining 95% of the data was used for training. In sector one the price stays more or less the same (-8%). In the second sector, the price drops to about 45% and in the third the price increases to 186%. In every sector the same combinations are separately trained and tested.

5.3 Combinations and reducing the search space

All possible experiment configurations that we evaluate include the combinations of all four different targets listed in Section 3.3, all six preprocessing combinations listed in Section 3.4, the three different neural network models listed in Section 3.5 and two possibilities for either including or leaving out the BTC closing price as "sentiment" value.

Hence, this search space is of considerable size and every training run is computationally expensive. Training every combination on all three sectors, ten epochs each would be equal to 4320 runs. To make this workload manageable the experiments were structured

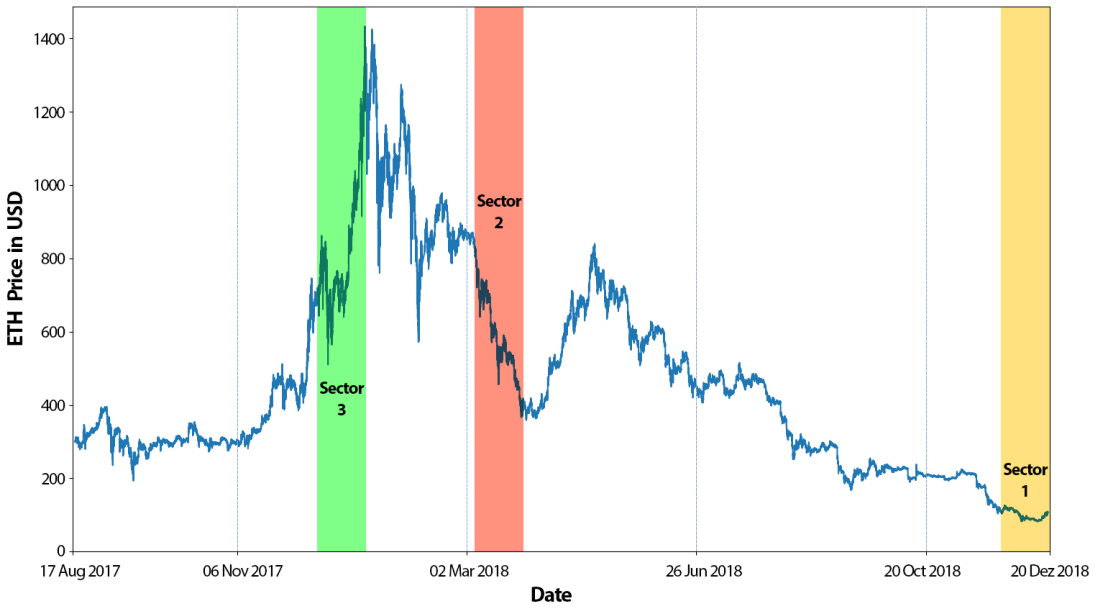

Figure 5.1: ETH price in USD on Binance [108], sectors 1-3 (right to left).

as follows. The full set of combinations was only tested with the first model in sector one, in order to find the most successful ones. Combinations that failed according to the definition above were disregarded for the following experiments to reduce the vast search space to the more promising combinations. The most successful combinations for each target were then tested on the other neural network models. All results can be seen in the appendix.

As we already discussed, a simulation was performed for every preprocessing model combination after every epoch. One of these simulations can be seen in Figure 5.2. The simulation value and the ETH price progression after that period of time are measured and their difference is calculated at the end. This difference is written in the tables that are presented in the following sections as *perf*, while the end value would be written as *value*.

5.4 Evaluation of targets and preprocessing

The structure of the neural network models is presented in Section 3.5 and Figure 3.2. To sum it up, the first model consists of one LSTM layer, followed by a dropout layer, a normalization layer, a dense layer with 32 nodes and an output layer with one, two or three nodes, depending on the target.

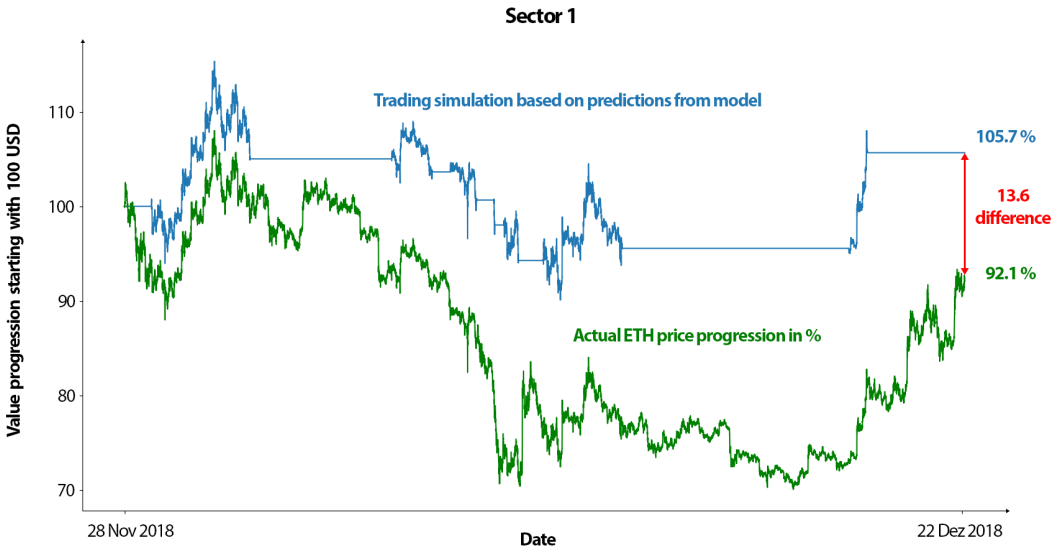

Figure 5.2: A single simulation for sector 1: The difference between the value of the simulation and the actual price progression of ETH is used to measure the performance of the models, here depicted in red.

We tested this model most thoroughly and used it to narrow down on the preprocessing combinations that performed best within 10 epochs. Every combination was tested for the first sector, which is equal to 480 test runs. The following combinations failed. The regression target on the relative price change was unsuccessful in every combination, as all trained models traded not even once in the simulation, with one exception where it traded twice in two epochs. Using the Min-Max normalization in combination with the binary target also resulted in failures. Similarly, any combination using the Min-Max normalization in combination with converting the raw data to the percentage change of its immediate predecessor failed. All other combinations with the Min-Max normalization produced far worse results than the other normalization techniques. Adding the Bitcoin price to the input as sentiment value did not increase the performance in most cases but rather seemed to have a negative impact. This is most likely either due to the fact that when using the Bitcoin price, we introduce an additional input feature without having more training samples or that the Bitcoin price is simply not influential.

The most promising target was by far the best strategy target. In the time frame of the first sector, the Ethereum price sank to roughly 92% of its initial value or by around -8%. Most of the combinations outperformed that -8% with some being even positive in most of the epochs, despite performing several trades and deducting a fee of 0.075% each time. Those results are shown in Table 5.1. A more detailed explanation of the terminology used within the tables can be found in Section 7.1.

Table 5.1: Results using the best strategy target for model 1 in sector 1: # stands for the number of trades carried out in the simulation, while *perf* shows how the simulation performed in comparison to the ETH price (shown in Figure 5.2). The ETH price sank to 92.056% in that time, so if a model increased its value to, for instance, 106.107%, a *perf* of 14.051 is noted down as the performance difference between the simulation value of 106.107 and the ETH performance of 92.056.

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	7.944	0	7.944	125	16.242	104	-3.224
	1	-0.359	1	10.505	77	-1.848	515	-34.359
	11	-3.807	18	9.126	7	1.849	597	-42.444
	0	7.944	291	-17.673	52	2.399	200	8.835
	4	-1.717	35	14.051	219	9.270	187	-4.387
	366	-15.093	1	-1.977	142	16.820	287	-8.434
	68	-1.063	19	-5.115	2	8.242	49	-4.608
	35	-4.771	119	-8.989	192	-4.590	6	6.254
	1	6.353	15	-2.634	264	-14.881	183	-19.547
	0	7.944	5	-3.488	48	6.208	51	-7.012
Robust	1	1.489	0	7.944	30	5.537	51	-7.560
	53	-7.747	0	7.944	2	6.733	40	4.261
	0	7.944	0	7.944	21	-8.937	297	-14.154
	649	-37.981	0	7.944	26	1.447	105	-12.694
	437	-31.837	1	-1.546	3	-2.434	304	-11.942
	2	15.074	1	1.278	15	-1.909	97	-12.233
	0	7.944	3	31.192	31	-7.166	62	-3.158
	35	6.898	1	6.353	0	7.944	38	4.430
	5	1.448	291	-18.217	20	6.113	3	-0.426
	7	2.174	101	-9.353	21	15.357	0	7.944

In all further experiments, only combinations consisting of the best strategy target as well as either the Standard or the Robust Normalizer were considered, resulting in eight different preprocessing combinations. This reduces the required number of test runs to just 640.

5.5 Evaluation of the neural network models

Even though we tested not using the percentage change of the preceding value for every model in every sector, this turned out to consistently produce unsatisfactory results, and will not be shown here. Table 5.2 shows the results for the remaining four preprocessing combinations for every model and every sector, averaged over all epochs. Note that we now show the average value the simulation achieved, not the difference to the actual ETH price. We go over these results in the next three subsections.

5.5.1 Model 1

For the remaining four combinations, the model had a similar average performance to the price in sector 1, being slightly below it. In the second sector the price sunk to around 45%. In this market, the model outperformed the price on average, even if just slightly. In the third sector the price rose to approximately 186%. Here the model not only underperformed the price, but was negative to the point of losing money in an otherwise very bullish market.

5.5.2 Model 2

The second neural network model is similar to the first one but with more layers. It consists of 3 LSTM layers, each followed by dropout and normalization layers. Then follows a dense layer which is followed by an output layer. This model performed worst out of the three models, performing on average worse than the price in every sector. It had a very large number of trades. While in sector 1 there were at least a handful of epochs that outperformed the price movement, namely when not using the BTC closing price, it performed badly in sectors 2 and 3 to the point of almost losing all money despite sector 3 being very bullish. The reason for this bad performance might be due to the large structure of the network and the in comparison relatively small training set (around 670,000 samples).

5.5.3 Model 3

The third neural network model has a similar topology to the first one but has additional convolution and max pooling layers on the input side. Out of all models, this one performed by far the best. On average, it outperformed the price in sector 1 and 2. In sector 3, it did worse than the price on average, but still remained positive.

Table 5.2: Number of trades and performance the models achieved on average in each sector with regard to the initial value of 100 USD. *Avg. value* stands for the amount of USD held after simulating trading over each individual sector.

	Best strategy target, %											
	no sentiment						sentiment					
	avg. #	avg. val.	a. #	a. v.	a. #	a. v.	a. #	a. v.	a. #	a. v.	a. #	a. v.
	Model 1											
	Sector 1		Sector 2		Sector 3		Sector 1		Sector 2		Sector 3	
Standard	112.8	96.03	152.8	58.46	1954.7	50.33	217.9	81.16	521.2	49.49	1523.3	71.64
Robust	16.9	94.32	342.5	45.88	2286.4	31.65	99.7	87.50	332.8	60.06	2237.7	37.95
	Model 2											
	Sector 1		Sector 2		Sector 3		Sector 1		Sector 2		Sector 3	
Standard	157.1	87.23	2200.4	11.12	4145.4	15.09	659	58.74	4408	4.56	6374	6.79
Robust	180.7	88.03	2459.9	11.40	4273.5	8.22	641.3	56.16	4260.8	5.93	6494.8	5.05
	Model 3											
	Sector 1		Sector 2		Sector 3		Sector 1		Sector 2		Sector 3	
Standard	12.4	103.08	10.2	81.67	68.8	125.13	11.2	93.05	15.5	79.39	72.6	122.52
Robust	7.1	102.34	2	76.18	24.1	121.45	8	99.95	11	72.31	56.3	141.22

Table 5.3: Number of trades and simulation value the models achieved on average over all sectors out of the initial value of 100 USD.

	Best strategy target, %			
	no sentiment		sentiment	
	avg. #	avg. value	avg. #	avg. value
	Model 1			
Standard	2220.3	28.252	2262.4	28.775
Robust	2645.8	13.699	2670.2	19.947
	Model 2			
Standard	6502.9	1.463	11441	0.182
Robust	6914.1	0.825	11396.9	0.168
	Model 3			
Standard	91.4	105.344	99.3	90.513
Robust	33.2	94.685	75.3	102.070

5.6 Performance

We finally regard all three sectors of the ETH price. Recall that the price in the first sector sinks to 92%, in the second it drops to 45% and in the third it rises to 186%. If one were to hold ETH in all three sectors, it would fall to about 76%. All three sectors add up to about 73 days. Please note that the order of the sectors does obviously not matter.

If we take the preprocessing and model combinations and take their average performance over all epochs, in all sectors, we can see how the preprocessing and model combinations would have performed when applying them over all three sectors. These results are shown in Table 5.3. We can quickly see, that model 3 is by far the best. This can be explained by the model's convolutional and pooling layers, which compute features and reduce the input dimensionality [124]. It outperforms the 76% of holding ETH over all three sectors easily, even being positive overall in two configurations on average. The number of trades is held at a rather low level in this model, so a different fee would not have as much impact as on models 1 and 2. While the best of the three models was rather clear, the best preprocessing combination is not. There are two that performed rather well. Interestingly enough, the Standard normalizer performed better without sentiment data, while the Robust normalizer performed better with sentiment data.

5.7 Training more epochs

To make the computation of these results feasible, the number of epochs used in every configuration was limited to 10. A major concern was that the performance increases significantly only after significantly more training epochs. To test the performance with regard to more epochs, the two most promising preprocessing techniques and model combinations were used and trained for 150 epochs on all three sectors.

The performance, however, did not increase significantly. Sometimes the models would hit snags and have zero trades after some epochs, but this behavior tends to be compensated after one or more epochs, most likely due to the dropout layers which prevent overfitting.

5.8 Comparison to other methods

We already compared our approach with the buy and hold strategy in the last sections. Model 1 and 2 performed worse than the buy and hold strategy, while model 3 performed better, with an average difference of almost 30% of the starting value for the best preprocessing combination.

Compared to the methods presented in Section 2.5, we used a more practical approach in this thesis. Unlike in those papers, where mostly the accuracy or the MSE was used to measure the performance, we used backtesting, i.e. the profit which was achieved when simulating trading with the trained networks (as explained in Section 3.6). The accuracy for the best strategy target, the classification target that was tested most thoroughly is not entirely accurate. This is due to the fact, that if we hold Ethereum, the signals *buy* and *hold* amount to the same trading outcome (*hold*), since we cannot buy more Ethereum when our portfolio already consists of 100% Ethereum. Similarly, when holding US dollar the signals *sell* and *hold* amount to the same trading outcome (*hold*). So the accuracy of the predicted signals is not the same as the accuracy of the rightly carried out trading actions.

Therefore we have two different accuracies for our models. We managed to get around 38.35% accuracy for the predicted trading signals, which is around 5% higher than random chance out of three choices. However, for the corrected accuracy, which represents the amount of rightfully chosen trading actions, we managed to achieve around 61.57%. This is also slightly higher than the 60.80% we managed to get when using the binary classification target. The corrected accuracy is higher in most cases than for most papers presented in Section 2.5, but we doubt that this comparison has substantial significance.

Note that using a trading fee in combination with high frequency trading had a big impact on the simulation outcomes. So while models 1 and 2 did not perform well at all, they still had comparable accuracies, only slightly below the third model. The problem is that with fees a large number of trades has a big negative impact on the outcome. Simulations without trading fees would result in large profits even with the simplest models and the binary classification target. In particular, during one run, model 2 managed to generate

a profit of around 254% with around 10,500 trades in the validation period using binary classification. When applying fees, the simulation value got close to zero.

The paper *Bitcoin Price Prediction Using Ensembles of Neural Networks* [107] did backtest their model and achieved a performance increase of about 85% over 50 days, compared to a BTC price increase of a little over 75%, outperforming buy and hold by roughly 10%. With our best combination we outperformed buy and hold over all three sectors (about 73 days) by 30%.

CHAPTER 6

Summary

6.1 Conclusions

The aim of this thesis was to see if predicting trading signals of cryptocurrencies with modern neural networks is possible and can result in a profitable trading strategy. We collected 705,000 one-minute samples of historic price data of the cryptocurrency Ethereum, ranging from August 2017 to December 2018. We evaluated three different neural networks, model 1 and 2 being LSTM networks and model 3 a hybrid of CNN and LSTM. We used the open, high, low, close and volume of Ethereum along with optionally the closing price of Bitcoin as features. We implemented 6 preprocessing combinations and 4 different targets, 2 for regression and 2 for classification.

We then set out to test these different configurations. The training is dependent on the target, using backpropagation with the Adam optimizer on either the MSE for regression and categorical crossentropy for classification as loss functions. We used a 95% to 5% training to validation split. The performance of the neural networks was measured by backtesting the models on the validation data. In other words, we used the predicted outcomes of the networks to generate trading decisions for every minute of the validation data, and carried this decision out. Starting at 100 US dollar we then looked at the value the simulation had in the end and compared it to the price development of Ethereum during that time, i.e. the buy and hold strategy. For every simulated trade a fee of 0.075% was deducted. The simulated trades were always carried out immediately and did not influence the market. We carried the simulation out independently over three different price sectors, one where the Ethereum price was rising, one where it was falling and one where is was staying roughly the same.

In the evaluation part, we found the target that tries to predict the best trading decision for each point in time to have the best performance and pinpointed the two best performing preprocessing combinations. These were transforming the data to the percentage change

of its predecessor and then applying either the standard or the robust normalizer. Utilizing the Bitcoin closing price as additional feature produced inconclusive results, performing slightly better on average when used in combination with the robust scaler and slightly worse when used in combination with the standard normalizer, as compared to not using the Bitcoin closing price at all.

Out of the three models, model 3 had the best performance. In the best configuration and averaged over all epochs, the first one had around 28$ of the initial 100 US dollars left after the simulation with an average of about 2220 trades over all three sectors. This result is worse than the roughly 76$ achieved when using buy and hold over the same period of time. The second model performed even worse, with only about 1.5$ of the initial 100 US dollars left. The third model achieved a little over 105$ in that time in the best configuration, with an average of 91.4 trades. Model 3 outperformed buy and hold buy almost 30%. The reason for the good performance of model 3 is most likely because of its convolution and pooling layers in the beginning, which compute basic features from the data and reduce their dimensionality. This may also hint towards the features that we used being in need of improvement.

The results point towards an answer, that, theoretically, by using neural networks to predict trading signals for Ethereum in high frequency one-minute intervals, arbitrage is possible to some degree and a slight profit or at least outperforming a buy and hold strategy can be achieved, even when trading fees are present. It is still unclear if this approach has real-world application and if by removing the simplifications assumed in the simulation, a profit is still possible, e.g. when the trades are not carried out instantly at the current market price or when trades actually have influence on the market.

6.2 Future work

Aside from testing the real-world application of the models presented in this thesis, there are numerous other ways of extending this work. This can be done by either improving the data or the model.

For improving the data, the most obvious step might be to use more data for training, either from years further in the past or newer ones as new trading data becomes available over time. Beside that, the data of other cryptocurrencies or even stocks could be used for training the model. The model could also be tested on different cryptocurrencies or stocks, maybe to get insights into the synergy between these assets.

Other than the quantity of the training samples, the features can be improved as well. An approach is to use other technical data, such as technical indicators. To boil down the number of features to the most important ones, an approach such as the principle component analysis might be useful [81].

Other than technical data, it is also interesting to see if fundamental data, such as current hashrate or transaction fee of the cryptocurrency has an impact on the prediction. For

this, a dataset with a granularity similar to the technical data would have to be acquired, in order to avoid major interpolation.

Using sentiment data for price forecasting is a very promising approach that has been experimented with to some degree recently. In particular, combining real sentiment data of news, Twitter posts and so on with the technical data in a prediction seems very promising to us. Tools for automated and well performing sentiment analysis are freely available, but aggregating the appropriate raw data is both expensive and tedious.

Finally, improving the model is another pathway. Since the best performing model was a relatively simple CNN LSTM hybrid neural network, it would make sense to experiment with more complex or different convolutions for the architecture of the neural network. New RNN constructs such as gated recurrent units (GRUs) can be investigated as well [125].

Also completely new approaches are possible, for example investigating if there are arbitrage possibilities in the price differences between different cryptocurrency exchanges. Trying all these different improvements and how they impact on the performance of cryptocurrency price predictions with neural networks is certainly worthwhile being pursued in the future.

CHAPTER 7

Appendix

In this chapter a detailed explanation of the data tables and the abbreviations used is given in Section 7.1. All results are listed in Section 7.2.

7.1 Explanation of the abbreviations used in the tables

All calculated results shown in tabular form are structured as follows. Their title refers to one of the different targets presented in Section 3.3 and their various preprocessing combinations. “MinMax”, “Standard” and “Robust” refer to the different data normalization techniques presented in Section 3.4. “no %” and “%” refer to whether or not the input data was changed to be the percentage values of their immediate predecessors. “No sentiment” and “sentiment” refer to whether or not the Bitcoin price has been added as additional feature representing an supposed inherent sentiment value for the cryptocurrency market.

Every combination is evaluated over 10 epochs, listed vertically from first to last, each with the number of trades (“#”) in the simulation of the model of that epoch and the performance (“perf”) difference to how the Ethereum price actually performed in percentage points (buy and hold strategy), a positive value meaning it performed better than the price, a negative value meaning it performed worse. A simulation on a preprocessing model combination at any epoch always starts with 100 USD. If it increases its value to 105% while the ETH price in that time falls to 92%, the performance of this preprocessing model combination at that specific epoch is marked down as 13% in the table, i.e. the difference between the simulation value of 105% and the actual ETH price development of 92%. Tables 5.2 and 5.3 show the average value (“avg value”) instead of performance, which represents the value the simulation achieved.

The simulation is explained in 3.6. The number of trades and the performance are each color coded to be able to compare the different results at a glance. The colors for the

number of trades range from red (no, very few trades) over white (medium amount of trades) to blue (many trades). The colors for the performance range from red (bad performance) over yellow (mediocre performance) to green (good performance).

7.2 All results

The Tables 7.1 to 7.4 are the raw results of all combinations tested with model 1 on the first testing sector as explained in Section 5.2. Tables 7.5 and 7.6 show the results for Sector 2 and 3 for the first model. Tables 7.7 to 7.9 show the results for the second model. Tables 7.10 to 7.12 show the results for the third model.

Please recall that for in sector 1, the ETH price sank to roughly 92%, in sector 2 it sank to roughly 45% and in sector 3 it increased to about 186%.

Table 7.1: Results model 1, sector 1, binary target

	Binary target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
MinMax	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
Standard	5156	-90.382	0	7.922	0	7.922	13142	-92.073
	40	6.341	0	7.922	0	7.922	14746	-92.076
	40	6.341	10	6.467	0	7.922	4084	-87.654
	504	-23.927	2	7.799	0	7.922	9700	-92.020
	594	-28.135	2	7.799	0	7.922	13168	-92.073
	334	-14.323	2	7.799	0	7.922	13432	-92.074
	446	-20.244	2	7.799	0	7.922	9470	-92.010
	318	-12.586	2	7.799	0	7.922	13986	-92.075
	0	7.922	2	7.799	0	7.922	14368	-92.076
	0	7.922	2	7.799	0	7.922	12367	-92.070
Robust	0	7.922	0	7.922	2	7.783	800	-39.264
	132	6.018	0	7.922	0	7.922	13178	-92.072
	0	7.922	0	7.922	0	7.922	5144	-90.030
	286	-7.335	0	7.922	0	7.922	9493	-91.993
	96	4.339	0	7.922	0	7.922	122	-3.212
	70	7.236	0	7.922	0	7.922	0	7.922
	0	7.922	0	7.922	0	7.922	0	7.922
	78	3.409	0	7.922	0	7.922	0	7.922
	6	7.154	0	7.922	0	7.922	10483	-92.039
	396	-17.726	0	7.922	0	7.922	12	7.651

Table 7.2: Results model 1, sector 1, Regression of raw price

	Regression of raw price							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
MinMax	1523	-60.000	15	-1.041	1	-0.069	1	-0.069
	1	-0.069	645	-36.215	1	-0.069	1	-0.069
	13	-0.316	349	-21.626	1	-0.069	1	-0.069
	51	-4.667	145	-8.829	1	-0.069	1	-0.069
	3	-0.142	91	-10.262	1	-0.069	1	-0.069
	13	-1.408	9	0.555	133	-7.978	1	-0.069
	18	7.137	129	-12.723	59	-4.364	1	-0.069
	2027	-68.343	133	-12.576	1	-0.069	1	-0.069
	15558	-92.055	171	-14.193	1	-0.069	1	-0.069
	3	-0.255	2401	-76.184	1	-0.069	1	-0.069
Standard	115	3.761	2466	-76.363	5	-0.007	1	-0.069
	2594	-79.330	1935	-66.494	113	-7.441	1	-0.069
	962	-40.770	57	-4.160	529	-26.665	1018	-48.042
	3975	-87.150	25	-3.943	477	-29.547	317	-19.858
	2001	-70.584	13759	-92.053	89	-7.979	77	-2.767
	371	-22.346	1871	-68.567	179	-14.267	437	-30.698
	2095	-72.302	2284	-75.235	41	-5.567	193	-12.201
	9205	-91.993	1905	-63.313	43	-3.104	169	-14.685
	5430	-90.453	51	-4.524	177	-12.272	289	-15.807
	6173	-91.106	3782	-86.037	147	-10.684	217	-13.804
Robust	1522	-60.778	4012	-87.852	1	-0.069	1	-0.069
	6269	-91.143	6046	-91.039	127	-10.777	121	-5.517
	9920	-91.997	3	-0.002	249	-16.148	53	-2.235
	4076	-87.511	3557	-85.066	529	-32.279	171	-11.329
	6799	-91.413	1327	-56.449	293	-14.847	313	-23.933
	227	-14.657	4944	-89.982	401	-26.174	93	-7.221
	253	-19.639	6475	-91.198	979	-48.060	23	-1.590
	67	-3.474	6047	-90.945	867	-46.752	223	-15.112
	421	-21.707	6187	-91.106	205	-10.441	859	-44.236
	721	-38.410	5291	-90.178	393	-25.808	943	-47.667

Table 7.3: Results model 1, sector 1, regression of relative price change

	Regression of relative price change							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
MinMax	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
Standard	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	2	7.359	0	7.944	0	7.944
	0	7.944	2	7.359	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
Robust	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944

Table 7.4: Results model 1, sector 1, best strategy target

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
MinMax	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
	0	7.944	0	7.944	0	7.944	0	7.944
Standard	0	7.944	0	7.944	125	16.242	104	-3.224
	1	-0.359	1	10.505	77	-1.848	515	-34.359
	11	-3.807	18	9.126	7	1.849	597	-42.444
	0	7.944	291	-17.673	52	2.399	200	8.835
	4	-1.717	35	14.051	219	9.270	187	-4.387
	366	-15.093	1	-1.977	142	16.820	287	-8.434
	68	-1.063	19	-5.115	2	8.242	49	-4.608
	35	-4.771	119	-8.989	192	-4.590	6	6.254
	1	6.353	15	-2.634	264	-14.881	183	-19.547
	0	7.944	5	-3.488	48	6.208	51	-7.012
Robust	1	1.489	0	7.944	30	5.537	51	-7.560
	53	-7.747	0	7.944	2	6.733	40	4.261
	0	7.944	0	7.944	21	-8.937	297	-14.154
	649	-37.981	0	7.944	26	1.447	105	-12.694
	437	-31.837	1	-1.546	3	-2.434	304	-11.942
	2	15.074	1	1.278	15	-1.909	97	-12.233
	0	7.944	3	31.192	31	-7.166	62	-3.158
	35	6.898	1	6.353	0	7.944	38	4.430
	5	1.448	291	-18.217	20	6.113	3	-0.426
	7	2.174	101	-9.353	21	15.357	0	7.944

Table 7.5: Results model 1, sector 2, best strategy target

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	1	-23.9669	0	-86.1997	2975	-176.218	3855	-176.676
	0	-86.1997	1	28.71177	2662	-170.203	5761	-178.961
	0	-86.1997	1	-70.3851	2393	-172.781	605	-87.1982
	0	-86.1997	0	-86.1997	1811	-150.922	1211	-145.708
	0	-86.1997	1	21.85831	3443	-175.884	432	-119.89
	0	-86.1997	0	-86.1997	1639	-156.64	940	-131.793
	0	-86.1997	1	-31.6115	911	-50.7969	421	-60.2185
	0	-86.1997	0	-86.1997	1831	-149.101	355	-52.693
	0	-86.1997	0	-86.1997	273	7.515481	1079	-111.448
	0	-86.1997	1	-0.13965	1609	-163.678	574	-81.0387
Robust	0	-86.1997	0	-86.1997	3012	-175.966	2431	-171.122
	0	-86.1997	1	5.88836	3205	-178.676	1389	-149.961
	0	-86.1997	0	-86.1997	834	-115.272	6475	-185.133
	1	-31.7222	1	-31.6115	4121	-183.96	2293	-158.482
	0	-86.1997	7	-69.531	1049	-95.3993	1619	-150.616
	0	-86.1997	0	-86.1997	1923	-143.535	2901	-171.727
	1	-21.425	3	-37.575	3079	-178.257	1464	-143.483
	0	-86.1997	1	-25.9652	2633	-175.906	743	-82.2659
	0	-86.1997	1	-3.38606	1969	-156.137	1527	-134.802
	0	-86.1997	0	-86.1997	1039	-142.353	1535	-134.882

Table 7.6: Results model 1, sector 3, best strategy target

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	55.46503	1	5.990694	226	29.72739	3023	-39.9065
	0	55.46503	1	5.654752	14	59.46176	20	35.65444
	0	55.46503	1	5.654752	475	-16.1299	259	0.070142
	0	55.46503	1	0.748808	200	-2.95901	155	24.6324
	1	5.654752	0	55.46503	154	9.730234	579	-8.74588
	1	5.654752	0	55.46503	68	17.34398	391	-10.1024
	0	55.46503	1	5.654752	122	0.505085	667	-4.63053
	1	5.654752	0	55.46503	90	44.64219	1	6.122253
	1	24.56928	1	2.350335	63	1.004341	21	5.563886
	0	55.46503	0	55.46503	116	-4.10092	96	40.88692
Robust	1	0.748808	0	55.46503	309	-7.8659	1461	-18.5029
	0	55.46503	0	55.46503	276	27.33527	552	14.41385
	0	55.46503	2	47.89453	285	-5.91141	58	1.788062
	0	55.46503	3	20.04673	443	-10.1863	376	19.06228
	1	5.654752	0	55.46503	285	-2.19976	138	33.87669
	0	55.46503	0	55.46503	69	6.709905	228	9.409531
	0	55.46503	1	38.20864	1140	-18.3361	129	-4.3501
	0	55.46503	0	55.46503	401	-11.1927	306	1.103708
	0	55.46503	2	49.99408	216	28.40892	80	43.01943
	0	55.46503	4	57.70896	1	6.71241	0	55.46503

Table 7.7: Results model 2, sector 1

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	7.943965	0	7.943965	7	4.743626	340	-23.3685
	0	7.943965	0	7.943965	113	-11.0325	590	-32.0667
	0	7.943965	0	7.943965	105	-2.68914	1066	-56.8299
	0	7.943965	1	26.28372	199	-12.6691	644	-33.5905
	0	7.943965	0	7.943965	170	-14.1567	674	-35.6117
	0	7.943965	0	7.943965	109	-1.2896	462	-34.6351
	0	7.943965	0	7.943965	137	-14.9505	896	-41.8617
	1	-1.54572	0	7.943965	251	2.803899	724	-23.3688
	0	7.943965	0	7.943965	350	-5.59311	780	-37.9355
	0	7.943965	1	33.45582	130	6.591804	414	-13.903
Robust	0	7.943965	0	7.943965	5	7.547606	799	-48.4551
	1	3.65066	0	7.943965	28	-0.54713	543	-28.6695
	0	7.943965	1	24.10098	175	-14.1055	378	-26.9185
	0	7.943965	0	7.943965	116	18.11212	396	-15.8417
	0	7.943965	0	7.943965	57	34.40861	480	-31.4443
	0	7.943965	1	28.94304	247	-17.7734	872	-43.317
	0	7.943965	104	-13.9573	258	-20.1849	562	-22.2523
	0	7.943965	0	7.943965	125	-4.40446	714	-49.0859
	0	7.943965	0	7.943965	291	-8.70126	487	-38.1173
	0	7.943965	0	7.943965	505	-34.5854	1182	-54.9078

Table 7.8: Results model 2, sector 2

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	1	-22.2945	0	-86.1997	3101	-176.917	6183	-184.649
	0	-86.1997	0	-86.1997	2891	-156.963	5503	-180.838
	0	-86.1997	0	-86.1997	3933	-166.11	4735	-168.507
	1	-55.9805	1	-20.8302	4587	-181.36	5937	-178.151
	1	-12.1954	0	-86.1997	4793	-179.648	5817	-174.236
	2	-19.8685	1	-31.9803	1823	-129.573	7949	-183.864
	0	-86.1997	1	34.0604	5531	-183.173	6257	-178.699
	0	-86.1997	0	-86.1997	5307	-181.441	6349	-178.043
	3	-3.89032	0	-86.1997	4009	-172.007	7835	-184.429
	0	-86.1997	0	-86.1997	5479	-183.917	7175	-182.68
Robust	0	-86.1997	1	-85.7253	3081	-172.376	4691	-174.994
	0	-86.1997	0	-86.1997	3293	-175.741	6053	-180.887
	0	-86.1997	0	-86.1997	4471	-178.175	5307	-176.247
	0	-86.1997	0	-86.1997	4973	-181.675	6353	-180.306
	0	-86.1997	0	-86.1997	3064	-173.409	6693	-182.356
	2	-80.8056	0	-86.1997	4955	-181.933	5801	-180.223
	0	-86.1997	0	-86.1997	5588	-184.541	8195	-184.623
	0	-86.1997	0	-86.1997	4545	-179.149	7411	-184.396
	1	-12.327	0	-86.1997	4213	-175.61	6891	-183.542
	0	-86.1997	0	-86.1997	4552	-177.229	7553	-183.955

Table 7.9: Results model 2, sector 3

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	55.46503	0	55.46503	1291	-26.8899	3935	-40.8802
	0	55.46503	0	55.46503	3079	-38.3648	2781	-33.1635
	1	23.40863	0	55.46503	1625	-27.379	5143	-42.4744
	0	55.46503	0	55.46503	1791	-32.9294	3885	-40.1615
	0	55.46503	0	55.46503	2635	-35.0925	2291	-33.0975
	0	55.46503	0	55.46503	1207	-29.0114	4579	-40.9801
	0	55.46503	0	55.46503	2303	-34.2571	5475	-42.5435
	0	55.46503	0	55.46503	3223	-40.3344	4919	-41.7572
	0	55.46503	0	55.46503	2877	-39.1331	5095	-41.8673
	0	55.46503	3	3.301289	1973	-30.7888	5977	-42.8508
Robust	1	5.990694	0	55.46503	597	-18.2156	1081	-16.5401
	0	55.46503	0	55.46503	1725	-30.0449	3603	-39.16
	1	7.66645	1	3.302933	1715	-25.7405	3549	-38.8326
	0	55.46503	0	55.46503	2846	-37.2918	5155	-42.3613
	0	55.46503	2	59.23671	1581	-28.4975	4473	-42.0444
	0	55.46503	0	55.46503	3245	-40.1134	2749	-36.0768
	0	55.46503	0	55.46503	4799	-42.8776	5831	-43.1126
	0	55.46503	0	55.46503	1973	-31.2148	5833	-43.2503
	0	55.46503	0	55.46503	3027	-38.4126	5579	-42.7224
	0	55.46503	0	55.46503	3091	-38.9434	4755	-41.9417

Table 7.10: Results model 3, sector 1

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	7.943965	43	12.84162	1	22.05774	6	-4.08673
	1	10.59307	5	3.23844	52	-12.6971	10	6.655583
	1	15.43965	0	7.943965	2	7.202061	6	0.053346
	0	7.943965	90	-14.1975	15	10.61994	22	-6.12146
	0	7.943965	0	7.943965	0	7.943965	2	8.264972
	0	7.943965	0	7.943965	4	13.49548	11	-6.69358
	4	10.3311	54	24.7059	6	10.10831	27	-15.0731
	0	7.943965	57	-5.1436	13	14.20253	8	14.04817
	11	1.325433	4	-1.40295	21	32.46483	6	5.25883
	0	7.943965	7	-0.91303	10	4.813451	14	7.607154
Robust	11	17.33621	0	7.943965	0	7.943965	0	7.943965
	0	7.943965	1	10.53439	1	28.7259	0	7.943965
	0	7.943965	0	7.943965	0	7.943965	1	-2.10517
	2	18.66987	0	7.943965	2	10.14556	0	7.943965
	0	7.943965	0	7.943965	4	12.04453	14	-8.30039
	0	7.943965	0	7.943965	33	26.15489	17	-6.99546
	20	2.694609	0	7.943965	12	5.962739	2	12.97737
	1	10.4075	0	7.943965	8	-4.15613	17	3.457938
	2	15.38734	0	7.943965	2	4.745512	24	39.02022
	3	4.65916	0	7.943965	9	3.327787	5	17.06613

Table 7.11: Results model 3, sector 2

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	1	-11.917	0	-86.1997	51	-21.4345	8	-76.8536
	1	-12.1662	0	-86.1997	53	-99.7333	44	-95.2181
	1	-12.3124	1	63.51479	69	-103.52	32	-52.4732
	2	-39.5187	0	-86.1997	62	-26.8025	89	-33.3965
	1	-12.3562	1	10.18331	27	-52.5553	13	-83.7154
	1	-12.3562	2	-68.6567	55	-110.418	91	-16.0112
	1	-12.3124	1	10.58842	131	-93.3676	68	-80.0716
	0	-86.1997	0	-86.1997	61	-42.4359	157	-84.3902
	1	-12.049	4	-16.8232	107	-32.4058	110	-56.7641
	1	-12.0344	0	-86.1997	72	-28.0166	114	-57.8619
Robust	1	11.82769	0	-86.1997	0	-86.1997	35	-14.3538
	2	-66.0412	0	-86.1997	39	-53.1145	7	-78.3723
	1	-12.3416	0	-86.1997	2	-80.7031	158	-57.8333
	2	-40.2657	0	-86.1997	32	-57.5857	13	-39.0026
	1	-12.2393	0	-86.1997	48	-46.9833	27	-67.4118
	4	-87.1662	1	-51.004	22	-71.855	18	-77.1894
	2	-38.9598	0	-86.1997	24	-74.0648	31	-97.0772
	1	-12.4292	0	-86.1997	11	-47.4045	153	-63.0653
	4	-34.392	0	-86.1997	28	-112.174	91	93.07215
	1	10.02899	0	-86.1997	35	-17.4236	30	-48.5364

Table 7.12: Results model 3, sector 3

	Best strategy target							
	no %				%			
	no sentiment		sentiment		no sentiment		sentiment	
	#	perf	#	perf	#	perf	#	perf
Standard	0	55.46503	0	55.46503	2	54.83899	1	12.39042
	0	55.46503	0	55.46503	0	55.46503	3	52.65932
	0	55.46503	0	55.46503	4	52.19609	0	55.46503
	0	55.46503	0	55.46503	14	47.94197	16	34.01333
	0	55.46503	1	26.14154	3	3.870916	1	17.54383
	0	55.46503	0	55.46503	18	41.31945	21	19.73147
	1	55.57556	0	55.46503	0	55.46503	10	60.21071
	1	6.122253	0	55.46503	27	10.1614	11	25.15281
	1	48.74664	0	55.46503	16	30.17637	54	47.28276
	1	6.524528	0	55.46503	18	19.95401	38	24.13746
Robust	0	55.46503	0	55.46503	1	6.262702	36	1.271962
	0	55.46503	0	55.46503	4	39.1737	1	36.85962
	1	7.004229	0	55.46503	1	26.81223	9	9.694553
	1	8.849288	1	6.353795	0	55.46503	44	38.61729
	0	55.46503	1	19.58999	0	55.46503	0	55.46503
	0	55.46503	0	55.46503	2	31.43952	3	25.77843
	0	55.46503	0	55.46503	3	16.24477	6	44.45318
	0	55.46503	0	55.46503	6	45.35184	5	15.40606
	4	41.25875	1	12.70553	2	34.97005	6	-5.25301
	1	6.262702	0	55.46503	1	5.264698	0	55.46503

List of Figures

List of Tables

Bibliography

[1] Web of Science, also known as ISI Web of Knowledge, Number of publications containing "deep learning" or "blockchain", see `http://www.webofknowledge.com/`, [online] accessed Feb. 4, 2019.

[2] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, May 2013.

[3] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

[4] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[5] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[6] X. Zhang and M. Lapata, "Chinese poetry generation with recurrent neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 670–680, 2014.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, (New York, NY, USA), pp. 303–314, ACM, 2018.

[8] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[9] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. `https://doi.org/10.5210/fm.v2i9.548`.

[10] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[11] G. S. Atsalakis and K. P. Valavanis, “Surveying stock market forecasting techniques – part ii: Soft computing methods,” *Expert Systems with Applications*, vol. 36, no. 3, Part 2, pp. 5932–5941, 2009.

[12] J. E. Jarrett and E. Kyper, “Arima modeling with intervention to forecast and analyze chinese stock prices,” *International Journal of Engineering Business Management*, vol. 3, no. 3, pp. 53–58, 2011.

[13] W. Brock, J. Lakonishok, and B. LeBaron, “Simple technical trading rules and the stochastic properties of stock returns,” *The Journal of finance*, vol. 47, no. 5, pp. 1731–1764, 1992.

[14] E. Chong, C. Han, and F. C. Park, “Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies,” *Expert Systems with Applications*, vol. 83, pp. 187 – 205, 2017.

[15] Snapshot of cryptocurrency market capitalization and 24h trading volume according to coinmarketcap.com, Jan. 2019. archived, available at `https://archive.fo/YAOHc`.

[16] P. J. Brockwell, R. A. Davis, and M. V. Calder, *Introduction to time series and forecasting*, vol. 2. Springer, 2002.

[17] J. D. Hamilton, *Time series analysis*, vol. 2. Princeton university press Princeton, NJ, 1994.

[18] Z. Tang, C. de Almeida, and P. A. Fishwick, “Time series forecasting using neural networks vs. box- jenkins methodology,” *SIMULATION*, vol. 57, no. 5, pp. 303–310, 1991.

[19] F. A. Gers, D. Eck, and J. Schmidhuber, “Applying lstm to time series predictable through time-window approaches,” in *Neural Nets WIRN Vietri-01* (R. Tagliaferri and M. Marinaro, eds.), (London), pp. 193–200, Springer London, 2002.

[20] L. Bachelier, *Théorie de la spéculation*. Gauthier-Villars, 1900.

[21] N. Wiener, “Differential-space,” *Journal of Mathematics and Physics*, vol. 2, no. 1-4, pp. 131–174, 1923.

[22] S. R. Dunbar, “Stochastic processes and advanced mathematical finance: The definition of brownian motion and the wiener process,” *Department of Mathematics, University of Nebraska-Lincoln, USA*, 2016.

[23] M. Nilsson, *First order hidden markov model: Theory and implementation issues*. 2005.

[24] B. G. Malkiel and E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," *The Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970.

[25] B. G. Malkiel, "Efficient market hypothesis," in *Finance*, pp. 127–134, Springer, 1989.

[26] E. F. Fama, "Random walks in stock market prices," *Financial Analysts Journal*, vol. 21, no. 5, pp. 55–59, 1965.

[27] B. G. Malkiel and K. McCue, *A random walk down Wall Street*. Norton New York, 1985.

[28] R. H. Thaler, "Anomalies: the january effect," *Journal of Economic Perspectives*, vol. 1, no. 1, pp. 197–201, 1987.

[29] R. K. Y. Low and E. Tan, "The role of analyst forecasts in the momentum effect," *International Review of Financial Analysis*, vol. 48, pp. 67–84, 2016.

[30] R. Ball, "The earnings-price anomaly," *Journal of Accounting and Economics*, vol. 15, no. 2-3, pp. 319–345, 1992.

[31] R. A. Haugen and J. Lakonishok, *The incredible January effect: The stock market's unsolved mystery*. Business One Irwin, 1988.

[32] T. C. Lin, "The new investor," *UCLA L. Rev.*, vol. 60, p. 678, 2012.

[33] M. Glantz and R. Kissell, *Multi-asset risk modeling: techniques for a global economy in an electronic and algorithmic trading era*. Academic Press, 2013.

[34] R. Sant and M. A. Zaman, "Market reaction to business week 'inside wall street' column: A self-fulfilling prophecy," *Journal of Banking & Finance*, vol. 20, no. 4, pp. 617 – 643, 1996.

[35] C. D. Kirkpatrick II and J. A. Dahlquist, *Technical analysis: the complete resource for financial market technicians*. FT press, 2010.

[36] J. S. Abarbanell and B. J. Bushee, "Fundamental analysis, future earnings, and stock prices," *Journal of Accounting Research*, vol. 35, no. 1, pp. 1–24, 1997.

[37] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[38] R. G. Hagstrom, *The Warren Buffett Way*. John Wiley & Sons, 2013.

[39] A. W. Lo, H. Mamaysky, and J. Wang, "Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation," *The journal of finance*, vol. 55, no. 4, pp. 1705–1765, 2000.

[40] R. N. Elliott, D. C. Douglas, M. W. Sherwood, D. Laidlaw, and P. Sweet, *The wave principle*. 1938.

[41] A. J. Frost and R. R. Prechter, *Elliott wave principle: key to market behavior*. Elliott Wave International, 2005.

[42] J. J. Murphy, *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[43] C.-H. Park and S. H. Irwin, "What do we know about the profitability of technical analysis?," *Journal of Economic Surveys*, vol. 21, no. 4, pp. 786–826, 2007.

[44] R. T. Farias Nazário, J. L. e Silva, V. A. Sobreiro, and H. Kimura, "A literature review of technical analysis on stock markets," *The Quarterly Review of Economics and Finance*, vol. 66, pp. 115–126, 2017.

[45] S. Schulmeister, "Profitability of technical stock trading: Has it moved from daily to intraday data?," *Review of Financial Economics*, vol. 18, no. 4, pp. 190–201, 2009.

[46] A. W. Lo, "The adaptive markets hypothesis: Market efficiency from an evolutionary perspective," 2004.

[47] A. Gorgulho, R. Neves, and N. Horta, "Applying a ga kernel on optimizing technical analysis rules for stock picking and portfolio composition," *Expert Systems with Applications*, 2011.

[48] Y. Wang, "Stock price direction prediction by directly using prices data: an empirical study on the kospi and hsi," *International Journal of Business Intelligence and Data Mining*, vol. 9, no. 2, pp. 145–160, 2014.

[49] Y. B. Wijaya, S. Kom, and T. A. Napitupulu, "Stock price prediction: Comparison of arima and artificial neural network methods - an indonesia stock's case," in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 176–179, Dec. 2010.

[50] S. Lauren and S. D. Harlili, "Stock trend prediction using simple moving average supported by news classification," in *2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA)*, pp. 135–139, Aug. 2014.

[51] J. Nadkarni and R. Ferreira Neves, "Combining neuroevolution and principal component analysis to trade in the financial markets," *Expert Systems with Applications*, vol. 103, pp. 184–195, 2018.

[52] S. C. Nayak, B. B. Misra, and H. S. Behera, "Index prediction with neuro-genetic hybrid network: A comparative analysis of performance," in *2012 International Conference on Computing, Communication and Applications*, pp. 1–6, Feb. 2012.

[53] J. L. Ticknor, "A bayesian regularized artificial neural network for stock market forecasting," *Expert Systems with Applications*, vol. 40, no. 14, pp. 5501–5506, 2013.

[54] A. M. Rather, A. Agarwal, and V. Sastry, "Recurrent neural network and a hybrid model for prediction of stock returns," *Expert Systems with Applications*, vol. 42, no. 6, pp. 3234–3241, 2015.

[55] A. U. Khan, T. Bandopadhyaya, and S. Sharma, "Genetic algorithm based backpropagation neural network performs better than backpropagation neural network in stock rates prediction," *Journal of Computer Science and Network Security*, vol. 8, no. 7, pp. 162–166, 2008.

[56] Y. Takemoto and S. Knight, "Mt. gox files for bankruptcy, hit with lawsuit." Reuters, Feb. 2014. archived, available at `https://archive.fo/i1RjJ`.

[57] Snapshot of Bitcoin price development according to coinmarketcap.com, Feb. 2019. archived, available at `https://archive.fo/FxkOO`.

[58] Snapshot of number of different cryptocurrencies according to coin.market, Feb. 2019. archived, available at `https://archive.fo/KYNmS`.

[59] N. Van Saberhagen, "Cryptonote v 2.0," Oct. 2013. archived, available at `https://archive.fo/IjP2Q`.

[60] S. Popov, "The tangle," Apr. 2016. v. 0.6, archived, available at `https://archive.fo/kDlRf`.

[61] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing*, pp. 163–180, Springer, 2015.

[62] Z. M. Seward, "This is how much a bloomberg terminal costs." Quartz, Feb. 2019. archived, available at `https://archive.fo/KII15`.

[63] C. Russo, "Crypto exchanges are raking in billions of dollars." Bloomberg, Mar. 2018. estimated revenue of cryptocurrency exchanges, archived, available at `https://archive.fo/udxFG`.

[64] L. Kristoufek, "What are the main drivers of the bitcoin price? evidence from wavelet coherence analysis," *PLOS ONE*, vol. 10, pp. 1–15, Apr. 2015.

[65] A. Greaves and B. Au, "Using the bitcoin transaction graph to predict the price of bitcoin," 2015.

[66] J. Kaminski, "Nowcasting the bitcoin market with twitter signals," *arXiv preprint arXiv:1406.7577*, 2014.

[67] M. Matta, I. Lunesu, and M. Marchesi, "Bitcoin spread prediction using social and web search media," in *UMAP Workshops*, 2015.

[68] I. Georgoula, D. Pournarakis, C. Bilanakos, D. Sotiropoulos, and G. M. Giaglis, "Using time-series and sentiment analysis to detect the determinants of bitcoin prices," 2015.

[69] I. Madan, S. Saluja, and A. Zhao, "Automated Bitcoin Trading via Machine Learning Algorithms," vol. 20, 2015.

[70] S. McNally, J. Roche, and S. Caton, "Predicting the price of bitcoin using machine learning," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 339–343, March 2018.

[71] A. Radityo, Q. Munajat, and I. Budi, "Prediction of bitcoin exchange rate to american dollar using artificial neural network methods," in *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 433–438, Oct. 2017.

[72] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[73] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer: New York, 2006.

[74] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[75] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.

[76] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[77] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[78] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. " O'Reilly Media, Inc.", 2018.

[79] A. Dingli and K. S. Fournier, "Financial time series forecasting - a machine learning approach," *Machine Learning and Applications: An International Journal*, vol. 4, no. 1/2/3, pp. 11–27, 2017.

[80] I. Jolliffe, *Principal Component Analysis*. Springer Berlin Heidelberg, 2011.

[81] X. Zhong and D. Enke, "Forecasting daily stock market return using dimensionality reduction," *Expert Systems with Applications*, vol. 67, pp. 126–139, 2017.

[82] Scikit-learn, `https://scikit-learn.org/stable/`, [online] accessed Mar. 11, 2019.

[83] K. Kuźniar and M. Zając, "Some methods of pre-processing input data for neural networks," *Computer Assisted Methods in Engineering and Science*, vol. 22, no. 2, pp. 141–151, 2017.

[84] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[85] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[86] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[87] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[88] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[89] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, vol. 385. 01 2012.

[90] C. Olah, "Understanding lstm networks," Aug. 2015. archived, available at `https://archive.fo/OMcur`.

[91] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," *CoRR*, vol. abs/1801.01078, 2018.

[92] P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[93] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[94] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," 04 1991.

[95] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[96] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[97] S. Yan, "Understanding lstm and its diagrams," Mar. 2016. archived, available at `https://archive.fo/ynlbP`.

[98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[99] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[100] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[101] Z. Zhang, "Derivation of backpropagation in convolutional neural network (cnn)," *University of Tennessee, Knoxville, TN*, 2016.

[102] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[103] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[104] W. Dubitzky, M. Granzow, and D. P. Berrar, *Fundamentals of data mining in genomics and proteomics*. Springer Science & Business Media, 2007.

[105] H. White, "Economic prediction using neural networks: the case of ibm daily stock returns," in *IEEE 1988 International Conference on Neural Networks*, pp. 451–458 vol.2, July 1988.

[106] L. Alessandretti, A. ElBahrawy, L. M. Aiello, and A. Baronchelli, "Anticipating cryptocurrency prices using machine learning," *arXiv e-prints*, p. arXiv:1805.08550, May 2018.

[107] E. Sin and L. Wang, "Bitcoin price prediction using ensembles of neural networks," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pp. 666–671, July 2017.

[108] Binance official public API, `https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md`, [online] accessed Feb. 26, 2019.

[109] The Block Explorer, `https://www.blockchain.com/explorer`, [online] accessed Feb. 26, 2019.

[110] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[111] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[112] J. Hale, "Deep learning framework power scores 2018." towardsdatascience.com, Sept. 2018. archived, available at `https://archive.fo/vSpeV`.

[113] M. Opala, "Deep learning frameworks comparison – tensorflow, pytorch, keras, mxnet, the microsoft cognitive toolkit, caffe, deeplearning4j, chainer." netguru.com, Sept. 2018. archived, available at https://archive.fo/TmSGH.

[114] TensorFlow, https://www.tensorflow.org/, [online] accessed Mar. 5, 2019.

[115] PyTorch, https://pytorch.org/, [online] accessed Mar. 5, 2019.

[116] Caffe, http://caffe.berkeleyvision.org/, [online] accessed Mar. 5, 2019.

[117] Theano, http://www.deeplearning.net/software/theano/, [online] accessed Mar. 5, 2019.

[118] MXNET, https://mxnet.apache.org/, [online] accessed Mar. 5, 2019.

[119] Microsoft Cognitive Toolkit (CNTK), https://www.microsoft.com/en-us/cognitive-toolkit/, [online] accessed Mar. 5, 2019.

[120] Deeplearning4J, https://deeplearning4j.org/, [online] accessed Mar. 5, 2019.

[121] Keras, https://keras.io/, [online] accessed Mar. 5, 2019.

[122] Pandas, https://pandas.pydata.org/, [online] accessed Mar. 19, 2019.

[123] Requests, http://docs.python-requests.org/en/master/, [online] accessed Mar. 19, 2019.

[124] M. Khoshdeli, R. Cong, and B. Parvin, "Detection of nuclei in h&e stained sections using convolutional neural networks," 02 2017.

[125] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.