TU WIEN Informatics

Cryptocurrencies: Deep-Learning for Sentiment & Market Analysis

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Muhm, BSc.
Matrikelnummer 01326486

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

Wien, 14. September 2021

Thomas Muhm	Dr. Horst Eidenberger

TU WIEN Informatics

Cryptocurrencies: Deep-Learning for Sentiment & Market Analysis

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Thomas Muhm, BSc.
Registration Number 01326486

to the Faculty of Informatics

at the TU Wien

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

Vienna, 14th September, 2021

Thomas Muhm	Dr. Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Thomas Muhm, BSc.

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, 14. September 2021

Thomas Muhm

Danksagung

Ich möchte mein herzliches Dankeschön an Herrn Ao.Univ.Prof. Dr. Horst Eidenberger aussprechen für die Möglichkeit an diesem überaus spannenden und einzigartigen Thema zu forschen sowie für seine exzellente Betreuung während der Ausarbeitung der Arbeit. Ich schätze es ihm hoch an, dass er stets für zeitnahes und wertvolles Feedback verfügbar war und mir hinsichtlich einer flexiblen Zeitplanung entgegenkommen ist.

Des Weiteren möchte ich meinen großartigen Freunden danken, welche mich zahlreich motiviert haben und mir den Mut zugesprochen haben, trotz zwischenzeitlicher Schwierigkeiten nicht aufzugeben. Ins besonders möchte ich David J., Lukas A., Alex M., Manu K. und Conny F. sowie meiner Schwester Chrisi und meinen Bruder David für die stetige Ermutigung und Unterstützung während der Diplomarbeit danken. Meinen Freunden und Arbeitskollegen Robin G. und Manu G. möchte ich ebenfalls meinen herzlichen Dank aussprechen für die vielen spannenden Diskussionen zum Thema der Arbeit. Zu guter Letzt möchte ich meinen Eltern dafür danken, dass sie immer an mich geglaubt und mir unentwegt positiv zugesprochen haben.

Acknowledgements

I would like to express my sincere gratitude to Ao.Univ.Prof. Dr. Horst Eidenberger for the opportunity to research on this exceptionally exciting and unique topic as well as for his excellent supervision during the writing process of the thesis. I highly appreciate that he was always available for timely and valuable feedback and that he showed flexibility regarding the timeframe of this work.

Furthermore, I would like to thank my great friends who motivated me in numerous ways and gave me the courage not to give up despite temporary difficulties. In specific, I would like to thank David J., Lukas A., Alex M., Manu K. and Conny F. as well as my sister Chrisi and my brother David for their constant encouragement and support during the thesis. I would also like to express my heartfelt thanks to my friends and work colleagues Robin G. and Manu G. for the many exciting discussions on the topic of the thesis. Lastly, I would like to thank my parents for always believing in me and providing me with constant optimism.

Kurzfassung

Der Fortschritt der Blockchain-Technologie sowie ihre weltweite Anerkennung innerhalb der letzten Jahre ermöglichte den Aufstieg digitaler Kryptowährungen. Diese Währungen zeichnen sich durch einzigartige Eigenschaften wie Pseudonymität, niedrige Handelsgebühren und geringe Einstiegshürden aus, was sie als Anlagemöglichkeiten zunehmend interessant macht. Darüber hinaus gelten diese Währungen als sehr volatil und nach Ansicht von Forschern trifft die Hypothese des effizienten Marktes noch nicht zu, wodurch sie als ideales Ziel für automatisiertes Trading gesehen werden.

Zeitgleich haben tiefe neuronale Netze sowie neuartige Architekturen neuronaler Netze vielversprechende Forschungsergebnisse in den Bereichen der Zeitreihenvorhersagen und der Sentiment-Analysen geliefert. Forschung bezüglich einer Kombination von Deep-Learning, technischen Indikatoren und Sentiment Analyse zur Vorhersage von Kryptomärkten ist jedoch immer noch mangelhaft. Das Ziel dieser Arbeit ist es daher, diese Forschungslücke zu untersuchen und Antworten auf komplexe, damit einhergehende Fragen zu liefern.

Um dieses Ziel zu erreichen, haben wir mehrere tiefe neuronale Netzwerke zum Generieren von Handelsstrategien für die Kryptowährung Bitcoin entwickelt und bewertet. Insbesondere haben wir uns auf die Optimierung der Struktur und der Hyperparameter der neuronalen Netze konzentriert, anpassbare Zielwerte für verschiedene Risikobereiche erforscht, alternative Input-Quellen getestet und eine Simulations-Engine für die generierten Handelsstrategien entwickelt.

Die Ergebnisse unserer Experimente bestätigen die Hypothese, dass Kryptowährungen enorme Möglichkeiten für profitables, automatisiertes Trading eröffnen. Unsere Experimente zeigen, dass KI-basierter Handel die Profitabilität im Vergleich zu einer Buy-and-Hold-Strategie deutlich verbessern und gleichzeitig das damit einhergehende Risiko reduzieren kann.

Abstract

The advances of blockchain technology and its global recognition over the last years enabled the rise of digital currencies, also known as cryptocurrencies. These currencies are characterized by unique properties like pseudonymity, low trading fees, and minimal barriers of entry which made them increasingly interesting as investment opportunities. Additionally, these currencies are considered highly volatile and the efficient market hypothesis does currently not hold true according to researchers, making them the ideal target for automated analysis and trading.

At the same time, deep neural networks and novel neural-network architectures have been producing promising research results for time-series predictions and sentiment-analyses. However, research on the combination of deep-learning, technical indicators, and financial sentiment analysis in the field of cryptocurrency market predictions is still scarce. The aim of this thesis is to explore this research gap and provide answers to the complex questions associated with it.

To reach that goal we developed and evaluated multiple deep neural networks to generate trading strategies for the Bitcoin cryptocurrency. Specifically, we focused on the optimization of the structure and hyperparameters of the neural networks, explored the space of risk adjustable target values, tested alternative input sources, and developed a simulation engine for the generated trading strategies.

The findings of our experiments confirm the hypothesis that cryptocurrencies open vast opportunities for profitable automated trading. Our experiments show that AI-based trading can significantly improve profitability compared to a buy-and-hold strategy while simultaneously reducing the risk associated with it.

Contents

CHAPTER 1

Introduction

This chapter provides a brief introduction to the research context, the motivation for this thesis, and the problem statement. Furthermore, it highlights the proposed research questions as well as the research objective in section 1.2. The last part outlines the applied research methodology and its rational as well as describes the structure of the thesis.

1.1 Motivation and Problem Statement

In 2008, a person under the pseudonym Satoshi Nakamoto released a whitepaper introducing a new form of currency called Bitcoin [1]. This event marked the beginning of a new era of currencies, so-called cryptocurrencies, with unique features and areas for application. All cryptocurrencies share the distinctive advantage of the underlying blockchain technology, which solved critical issues that digital currencies had up to that point. Today, thousands of different cryptocurrencies exist [2], many of which are just small modifications of the Bitcoin protocol while others bring new ideas such as smart-contracts (e.g Ethereum [3]) or privacy (e.g. Monero [4]) into the crypto-ecosystem. Most of these currencies promote a strong value statement as they promise to be secure, fast, unregulable, and independent from any government [5].

Much like company stocks, these cryptocurrencies can be traded on so called crypto-exchanges, which are comparable to stock-brokers. However, the crypto-market shares much more characteristics with the penny-stock-market than the stock-market. Main arguments for this include the exceptionally high volatility of cryptocurrencies and the fact that many of them, especially smaller currencies, can be manipulated by single market participants that move sizeable amounts of funds [6].

Viewing cryptocurrencies under the light of investment opportunities, one can question the otherwise dominant efficient market hypothesis (EHM), which states that every available piece of information is reflected in the prices on the stock market. According to the hypothesis, it is not possible to predict the future of the market better than a random coin toss does [7]. Given the above arguments and the overall novelty of the crypto-market, we believe that this hypothesis does not yet hold true for most cryptocurrencies. This gives us the opportunity to perform market predictions that are better than random.

For that purpose, deep neural networks can be seen as the most suitable methodology. In recent years, they received a lot of attention while deep-learning for time-series predictions has yielded promising results [8]. However, research on the combination of deep-learning and technical indicators is much scarcer. We expect this combination to possibly lead to significantly better results than the mere direct use of price and volume data. This hypothesis is strengthened by Aumayr [9]. Aumayr implemented multiple deep neural network models which predict the price movement of cryptocurrencies by using price data as input to these models. He concludes that the predictions can be further improved by optimizing the feature extraction step. Feature extraction shares many similarities to technical indicators, as both methods modify the input layer to achieve more accurate results.

This leads to the question which determining factors, if any, influence the market movements of cryptocurrencies next to price and volume data. Our assumption is that public opinion might be able to significantly affect the prices of cryptocurrencies and therefore deserves special attention. The underlying cryptocurrency community is noticeably active on various microblogging platforms such as twitter, stocktwits, and reddit, which results in a rapid flow of information among the majority of market participants [10] [11]. Consequently, We reason that these communities are able to affect the prices of cryptocurrencies through microblogging messages, especially considering above assumptions about volatility, novelty, and low market caps.

In conclusion, we believe that the combination of automated news analysis, price analysis, and technical indicators is able to significantly outperform competitive approaches to predict the market. This hypothesis was tested by implementing an advanced deep learning system for market predictions and sentiment analysis for financial microblogging data. Additionally, a simulation engine for the evaluation of different risk levels associated with the entered trading positions was developed.

1.2 Aim of the Work

The problem statement described in section 1.1 yields multiple interesting areas for research and angles to approach them. To evaluate the possibility to outperform the market, this thesis pays special attention on the topics of deep learning for cryptocurrency price predictions, sentiment analysis for financial microblogging data, as well as methods for the risk reduction of trading strategies.

Specifically, the following questions are researched, evaluated, and discussed:

1. Is it possible to improve the results of a price prediction system through an optimization of the internal structure and the parameters of its neural network? Analyzed through the case of *Aumayr's* price prediction system.
2. Which target values can be selected for a price prediction neural network to allow for risk adjustments?
3. Can the utilization of different combinations of technical indicators improve the results of the price prediction?
4. Does microblogging sentiment data have a positive impact on price predictions?
5. Can options like short trading, stop loss, or a limited trading position lower the risk of an automated trading strategy?

These research questions are relevant and important for a number of reasons.

First, the quest for a way to optimize the neural network structure and its parameters in an automated way is a hot and highly active research topic. The emergence of novel, structured, and well-documented findings is crucial to advance the current body of knowledge based on the extraordinary recency of the topic.

Second, the research questions focus on the specifics of optimizing an existing neural network for the prediction of cryptocurrency markets. Artificial intelligence in the field of finance and cryptocurrencies can be characterized by a high level of research scarcity, as the majority of high quality experiments are performed for personal gain. The resulting lack of publicly available explorations reveals the importance to produce accessible findings.

Furthermore, our research focuses, among other things, on input features for market prediction models. One of the research questions centers around technical indicators, which are used by various types of traders and represent one of the main toolkits for the typical crypto trader. Our approach, however, combines multiple indicators to identify and shed light on hidden predictability for crypto markets. Hence, we believe this combination approach provides valuable insights for a novel research sphere.

Lastly, our research questions reveal a whole set of approaches to potentially increase returns and reduce risks from financial investing in crypto markets. If adapted, these findings can provide critical knowledge for private and institutional investors to adapt latest technologies and thereby profit from lower risks and greater gains. In this sense, we believe our findings to suggest novel ways to increase the upside of investing in a broader spectrum, resulting in an increase of public wealth.

A semi-automated system consisting of four major parts was drafted and developed in order to answer the proposed research questions and research the hypotheses.

The first part of the system uses deep learning technology to make predictions on the cryptocurrency market. In this step, we constructed a software to create risk-based target values for the neural networks. Subsequently, we developed an optimization engine for the internal structure and the parameters for the neural networks.

The second part of the system automatically analyzes the sentiment of different microblogging platforms to identify implications for the crypto market. This part provides a download tool for multiple social networks as well as a feature to cleanup and pre-process the downloaded messages. Next, it assigns the sentiment values positive, negative, or neutral to the messages. The assignment happens through a specialized deep neural network for the classification of financial microblogging data.

The next part focuses on the training and evaluation of the neural networks. For this purpose, we utilized the following input features:

- crypto market data
- combinations of 62 different technical indicators
- microblogging sentiment data

The final part of the system provides a simulation and backtesting engine for the prediction results of the neural networks. This engine allows for further adjustments of the risk tolerance by allowing the configuration of short trading, stop loss, and the regulation of the position size. Moreover, the engine provides statistical values about the resulting trading strategy.

1.3 Methodological Approach

The core part of this thesis involves the construction and the evaluation of an advanced deep learning prediction system for the crypto market. The system supports the sentiment analysis process of microblogging data and is able to use different kinds of time-series data during the market prediction process.

First, in order to gain a comprehensive understanding about the research context, an in-depth analysis of state-of-the-art literature is carried out. Main topics of interest include research on the blockchain technology and cryptocurrencies, general aspects of trading, trading strategies and different kinds of technical indicators, time-series prediction with neural networks, and deep learning for sentiment analysis.

Building on the most recent knowledge, we developed a software tool featuring deep learning methodology to draw novel conclusions on market predictions on cryptocurrencies. For that purpose, we identified and downloaded historic trading data for bitcoin. Next, we constructed a target generation engine which allows for risk adjustments and carried out an optimization process for the internal structure and parameters of the neural networks.

This operation included the identification and selection of 62 different technical indicators. Ultimately, we finalized the software by engineering the input selection, cleanup, and transformation process.

In a next step, we enhanced the system by adding a sentiment analysis process. For this process, we selected relevant microblogging sources based on their intensity of use inside the crypto community. Then, we implemented a software to download the data to provide the system with messages of the selected sources. Finally, we used these messages for the evaluation of multiple neural networks for the sentiment analysis process.

Finally, we evaluated our proposed research questions. To achieve this, we implemented a system to simulate trades according to the neural network predictions. Furthermore, we tested options for short trading, stop loss, position size regulation, and risk adjustment.

1.4 Structure of the Work

The approach of this thesis is to pay attention to readability and therefore provide the information in the most logical sequence, chronologically ordered, and divided in four Chapters as follows:

Chapter 2 (State of the Art)
State of the art provides an overview of the most relevant literature for the specific research context. The most important findings and insights are highlighted and used throughout the thesis. In particular, the chapter includes an introduction to the blockchain technology, cryptocurrencies and their features, as well as the special case of bitcoin. Furthermore, it provides an overview of the stock and cryptocurrency market, the general aspects and terminology of trading, and a detailed description of technical indicators. Additionally, the chapter includes an in-depth introduction to different neural networks and their inner structure.

Chapter 3 (Design)
This chapter outlines the methodological approach which was used in this thesis. On the one hand, an in-depth description of the inner parts of the developed system is provided. Furthermore, it is precisely stated how the different input data was obtained and how it was cleaned, preprocessed, and transformed to be useable by the neural networks. Next, the chapter describes the process for the optimization of the structure and hyperparameters of the neural networks. In addition, it gives a detailed report of the developed sentiment analysis processes. The last part provides an elaboration about the different performance metrics which were used to find the best model.

Chapter 4 (Implementation)
The implementation chapter presents the different languages, libraries, and tools which were used to develop the described system. Furthermore, it provides a detailed description of the implemented system, which includes a presentation and discussion of the project's most important code parts. The last part of this chapter incorporates the utilized hardware and the required computation times as well as describes the applied code optimization procedures.

Chapter 5 (Results and Evaluation)
Chapter five provides a comprehensive explanation of the process used to evaluate the whole system. This includes the optimization procedure for the neural networks, the selection process of the technical indicator combinations, as well as the evaluation of the sentiment analysis. Furthermore, it presents the results of the trained neural networks and assesses the research questions.

Chapter 6 (Summary)
The last chapter summarizes this thesis's key findings and presents a concise conclusion for the carried out experiments. Additionally, it includes a section with limitations as well as identified opportunities for future research.

CHAPTER 2

State of the Art

This chapter provides a comprehensive analysis of state-of-the-art literature concerning the various technologies and methods used throughout the thesis. Section 2.1 presents an overview of the blockchain technology including a general introduction to cryptocurrencies and a deep-dive into the bitcoin specifically, since this currency was used for the price predictions in this thesis. Section 2.2 highlights the context of trading markets with descriptions regarding the general trading terminology, different relevant trading markets and various analytical methods for trading strategies. Section 2.3 provides a contextual understanding of artificial neural networks. It outlines the various aspects of a neural network and presents a detailed introduction to recurrent, convolutional, and transformer neural network architectures. In the last part of this chapter, we present further papers with a similar research context or methodology.

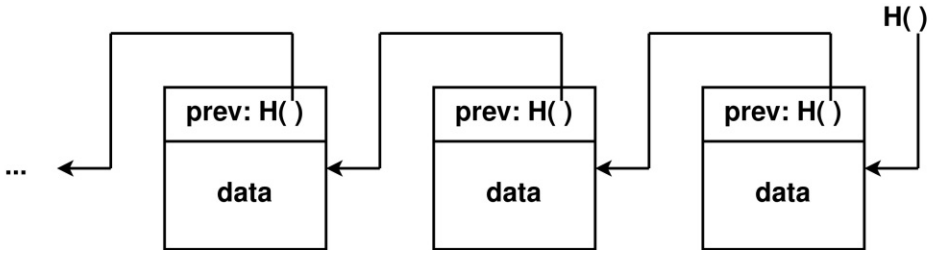
2.1 Blockchain & Cryptocurrencies

2.1.1 Blockchain Technologies

A blockchain is a digital database or digital ledger where information is grouped together and stored inside so called blocks. New content can only be appended within a new block at the end of the database. Therefore, as the name suggests, a blockchain can also be described as an append-only list of blocks or chain of blocks [12] (p. 38). These blocks are linked together via cryptography. This guarantees that the content of each block cannot be altered without invalidating all the blocks after the altered data. In detail, each block contains the cryptographic hash of the previous block which is automatically validated every time the chain is altered. The hash is calculated by a cryptographic hash function which is a one way mapping for data of arbitrary size to a fixed size output, the hash. Changing any part of the input will result in a completely changed hash value [12] (p. 28). Image 2.1 illustrates the structure of a simple blockchain consisting of three blocks.

The blockchain technology results in a specified amount of benefits including decentralization and protection from manipulation. Applications derived from these advantages range from decentralised financial services, which is currently the primary use, to smart contracts, and more [13].

Figure 2.1: Simple blockchain illustration, *Bitcoin and Beyond* [12] (p. 38)



The concept for the first blockchain was invented by the pseudonym Satoshi Nakamoto who intended to build a decentralized and secure currency that does not necessitate trust after the big financial crash caused by the banking industry in 2007-2008 [1][14]. This concept was built on a decentralized peer to peer network where anyone can run a node and join or leave the network at any time [15] (p. 2089). Each node checks the validity of the blocks on its own, resulting in the redundancy of trust in other nodes. The main problem for such a technology was the method of creating new blocks. If anyone can create new blocks without any effort, it becomes impossible to find a consensus between the nodes in the network. Therefore, one of the big breakthroughs of Nakamoto's invention was the design of a decentralized consensus system based on the Proof of Work (PoW) algorithm [15] (p. 2086).

PoW is a cryptographic zero knowledge proof that can be used by one node to prove to the other nodes that a specified number of computational power was used [16]. This proof can then be verified by other nodes with minimal effort. More specifically, the PoW used in the blockchain technology is a brute force search process for the hash value of the next block that falls within specified target range, also known as the difficulty [15] (p. 2086). During the search process, a single value within the new block is incremented to change the hash output. The difficulty to find a new block in a blockchain is automatically adjusted by the blockchain itself based on the computational power that is currently used for finding blocks [12] (p. 70). This adjustment ensures that the average time to find new blocks always remains at the same level. The nodes which participate in this process are called miners and the process for finding a block is regarded as mining.

2.1.2 Cryptocurrencies

Cryptocurrencies – representing the primary use case of the blockchain technology – are a digital, decentralized medium of exchange which does not require a middleman, is not controllable by any government, and does not require the trust of any third party [15] (pp. 2085-2086). Furthermore, the transactions for cryptocurrencies are fast, secure, and

only require a small transaction fee to be processed [12] (pp. 79-84). These beneficial properties are achieved through the underlying blockchain technology explained above. More specifically, the blockchain of a specific currency functions as a digital ledger to store the full transaction history of every user. This results in the public visibility of every historic transaction of every user. Only a few cryptocurrencies focus on complete anonymity and therefore developed methods to hide the transaction history while still providing a cryptographic proof for the existence of every transaction. Furthermore, the transparency of cryptocurrencies in publicly visible ledgers can be bypassed with so called Mixer services. Such a service takes coins from multiple addresses as an input and obscures their trail by sending them to new addresses within a single transaction [17].

The pseudonymity of cryptocurrencies is achieved by using an address system which does not link any information to its real users [12] (pp. 173-174). This system uses an address to send coins to or receive coins from another address. New addresses can be created on demand without effort and any user can have an unlimited amount of them.

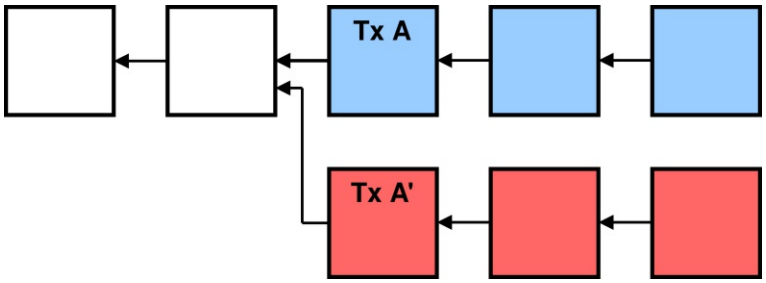
Cryptocurrencies use asymmetric cryptography for the transaction process, which means that two keys for each address exist [15] (pp. 2092-2093). The first one is a private key which is required to sign a transaction. By signing a transaction, a user can prove ownership of an address and is therefore able to spend the coins of this specific address. The second key, also called public key, is known by the entire network and is used to verify the validity of a signed transaction.

One of the major problems for digital currencies before the invention of cryptocurrencies was the double spending problem [15] (pp. 2093-2096). Double spending describes the hazard that an owner of a digital currency is able to spend the same funds multiple times without anyone noticing. This problem is solved on multiple levels within the blockchain. Firstly, since all transactions on a blockchain are cryptographically secured and publicly verifiable, it is not possible to spend the same funds multiple times within one valid chain [15] (p. 2093). However, the possibility remains that a chain splits into multiple chains where each new one contains a transaction that sends the same coins to a different address. Such a chain split is called fork, which is illustrated in figure 2.2 (p. 167) [12]. In order to solve the problem of chain splits, nodes always use the longest valid chain for the current state of the blockchain. The only remaining risk is a so called 51% attack [15] (p. 2094). It would require a malicious user to create blocks faster than all other miners, thus mobilizing computational resources of 51% of the network, to create a successful double spend attack. The difficulty for such an attack increases exponentially with each block that is added after the malicious transaction. These added blocks are therefore called confirmations for the transaction. For the bitcoin network, as an example, it can be said that the attack is infeasible if there are three or more confirmations for a transaction.

To create demand to expand a cryptocurrency and to prevent miners from acting maliciously they are most commonly incentivized by being rewarded with coins from the underlying cryptocurrency as well as the transaction fees for every successfully mined block [12] (p. 67). This block reward is often reduced over time until the maximum

supply of the currency is mined. Once that point is reached, miners are only paid by the transaction fees of other users.

Figure 2.2: Illustration of a blockchain fork, [12] (p. 167)



2.1.3 Bitcoin

Bitcoin was invented by the pseudonym Satoshi Nakamoto in 2008 and represents the first cryptocurrency as mentioned in the introduction [1]. Like most other cryptocurrencies, it is an open-source project. As a consequence, many descendant cryptocurrencies are based on its codebase. Bitcoin is often called a deflationary currency because its maximum supply is capped at 21 million coins where every coin lost or destroyed shrinks the maximum supply indefinitely [12] (p. 67). The current reward for miners is 6.25 bitcoins per block. This reward value started at 50 bitcoins when the currency was introduced and is halved approximately every four years or more specifically every 210,000 blocks.

Bitcoin attracted various kinds of personalities. In the first years, the typical investor was characterized as someone curious about the technology itself or a person that required an untraceable digital currency [18]. The second wave of buyers was attracted after the first big price movements of a few hundred percent and mainly consisted of speculative investors [19]. In the current phase, bitcoin is increasingly viewed as a store of value, which explains the growing number of institutional investors [20][21][22]. In many cases – especially within countries with a high inflation rate like Venezuela, Turkey, Chile, or Russia – people are shielding their wealth by storing their assets as bitcoin to preserve them from devaluation [23][24].

The first commercial bitcoin transaction was the payment of two pizzas in 2010 for 10,000 bitcoins which is nowadays considered a historic event [25]. Since that day the price of bitcoin reached a new all-time high approximately every four years with the most recent one of 57,000$ for a single coin on 22.02.2021 [26]. It is important to note, however, that the price movements are not only positive. In December 2017, for example, the price of bitcoin had reached 20,000$ followed by a crash down to 3,500$ in the following years, representing an 83% decline. In fact, the volatility is so high that a price movement of more than 10% in a single day is considered normal. Consequently, bitcoin depicts a very promising asset for short- and mid-term traders as well as trading bots.

2.2 Trading

The first part of this section introduces the stock market and the cryptocurrency market. The following part describes relevant trading terminology to foster understanding and clarity throughout the thesis. The last part of this section presents different kinds of analytical methods for trading strategies, and an in depth description of the 62 technical indicators.

2.2.1 Trading Markets

Stock Market

The most important marketplace for trading is the stock market, also known as the stock exchange. This refers to a place where financial activities such as buying, selling, and the issuance of publicly held companies takes place [27] (pp. 3-4). In this context a share, also called a stock, represents partial ownership of an underlying company. Compared to other markets the stock market is a highly regulated & secure environment [28]. The general trading activity requires stockbrokers who act as middlemen. Such brokers can be banks or businesses with the sole purpose of stock trading. The stock market itself is a major driver for the economic growth of modern society, for example through its function as an institution for capital allocation as shown by Beck & Levine in 2004 [29].

The price of a stock can be visualized as a non-linear time series chart and is often described by the random walk theory [30] (pp. 19-21). This theory states that the price changes act in a random way and can not be predicted. The efficient market hypothesis claims that same statement by arguing that every available information is already reflected in the current price of any stock respectively [31] (pp. 41-49). Consequently, according to the hypothesis, future market developments are considered in today's price, making predictions beyond that impossible. Despite the general knowledge of this information the stock market is attracting more new traders and investors every year, many of which are confident to beat the market.

Cryptocurrency Market

The cryptocurrency market is the counterpart to the stock market in the world of cryptocurrencies. There are three different modes of exchange for these currencies. The most basic one is through cryptocurrency brokers, which are companies that allow users to buy and sell cryptocurrencies with fiat amount with a simple and easy to understand user interface. Centralized cryptocurrencies exchanges (CEX), as the next option, are used by more advanced users, manual traders, and automated trading bots. These exchanges offer the whole functionality of trading platforms including Order Books, Margin-Trading, APIs, etc. CEXs also offer the possibility to exchange cryptocurrencies with other cryptocurrencies. Examples for such cryptocurrency exchanges are Binance [32], Coinbase [33], Kraken [34], and Bitpanda [35]. The last mode of exchange is processed through decentralized cryptocurrency exchanges (DEX), which are entirely decentralized algorithms and coded into the blockchain itself [36].

The crypto market is growing at an enormous speed both in number of available currencies and in terms of the total market cap. As of April 2021, there are more than 3000 different cryptocurrencies [37], each having its own trading price. The combined market for cryptocurrencies broke the market cap of USD 2 trillion on April 10th, 2021. This value, however, is not distributed equally between the currencies. Bitcoin for example represents by far the biggest cryptocurrency and grabs about 40 to 70 percent of the whole market cap depending on the current market cycle. Figure 2.3 displays the development of the total crypto market cap in USD between 2016 and 2021 with a logarithmic scale and marks the key breakpoints of 100 billion and 2 trillion.

Figure 2.3: Total cryptocurrency market cap (logarithmic scale), tradingview.com [38]

One of the key differentiating factors of the cryptocurrency market compared to the stock market is the much lower level of regulations and the existence of gray areas [39]. Trading with fiat money, for example, is highly regulated compared to trading with stablecoins, which represent a cryptocurrency that is pegged to a fiat currency [40] (pp. 65-66). This leads to the phenomenon that most of the cryptocurrency exchanges offer a wide variety of trading pairs with stablecoins but only a few with fiat money. Another consequence of the low level of regulations is the size of margin offered in exchanges – a margin of e.g. 100x is no rarity for the crypto market [41]. All these big margin trades are another contributing factor to the high market volatility in the crypto market.

Another characteristic of crypto markets is the existence of different kinds of market manipulations resulting from the lack of regulations [42] (p. 212). So called whales, for instance, describe market participants with enormous financial capital that can set off significant market movements by buying or selling at specific chart patterns (2.7) to trigger stop losses (2.2.2) and margin liquidations [42] (p. 235). Another example are pump and dump groups, which are coordinated groups of people which collectively buy a currency at a predefined time and immediately sell it afterwards. In pump and dump groups, only the insiders are profiting off the trades while most other members are set to lose money [42] (pp. 215-216). The third manipulation is caused by leading crypto influencers. Elon Musk, for example, is able to move the bitcoin market by a two-digit percentage value with just a single tweet [43].

In contrast to these mentioned downsides, the crypto market offers some unique advantages. Trading cryptocurrencies, for example, is possible 24 hours a day, 7 days a week [44]. Furthermore, resulting from the highly emotional and particularly volatile character of crypto trading, are lots of market imperfections and trading opportunities. Moreover, it is possible to trade with complete anonymity on decentralised exchanges. Lastly, all involved transactions like buying, selling, or transferring assets are associated with low fees compared to traditional markets [44].

Summing up the most important characteristics, it can be said that the efficient market hypothesis does not seem to hold true for the case of cryptocurrencies which makes the underlying market an ideal target for prediction engines. This statement is strengthened by the author Kyriazis in a paper which elaborates on investment opportunities in the crypto market [45].

2.2.2 Trading Terminology

This section outlines relevant terminology to facilitate understanding and clarity about financial trading in regard to the research context.

Trading Strategy

The term trading strategy describes a concrete plan for entering and exiting a trading position (2.2.2) with the goal of making profit. This plan should uphold the principles of being verifiable, quantifiable, consistent, and objective [46] (p. 46). Most amateur traders in the markets do not follow predefined trading strategies and are known to make spontaneous adjustment to their strategies based on emotions. This has been identified by research as the main reason why the average day trader fails to beat the market and consequently loses money [47] [48].

Trading Position

After an investor executes a trade to either long or short the market (2.2.2), he or she is in an active trading position [49]. This position is characterized by a position size (2.2.2), a direction, as well as an exit strategy. An active trading position grows in value when the market is moving according to the investors' expectations. Upwards movements, in this sense, increase the value of long trades while a declining market increases the value of short trades. Exit strategies differ and can be defined as a fixed target price or a dynamic value based on the price movements.

Position Size

The position size is the invested amount for an entered trading position and can be stated in absolute numbers or relative share in regard to the investment portfolio [50]. As an example, both an absolute $100 and a relative 2% are adequate ways to define the size of a position. Once the size of a single position equals 100% of the trading account it is

called ‘all in‘, which represents a trading position that is often used for buy-and-hold strategies (2.2.2). In the specific case of margin trading, it is possible to have a position greater than 100%. Margin trading allows the trader to borrow additional funds by using his whole trading account or a single trading position as collateral. Generally, the higher the position size the more risk the trader takes for a single trading position. Consequently, the quest of identifying a good position size is highly important. There are multiple ways to approach this problem, one of which is called the Kelly criterion and is also known as the scientific gambling method [51].

Stop Loss

A stop loss predefines an exit point for a losing trading position. It consists of two parameters, the activation price and the exit price [31] (p. 263-265). The activation price represents the trigger point for opening a new order that closes the trading position at the defined exit point. The very similar trailing stop loss shares the same idea but defines the activation and exit points as percentage values [31] (p. 265-267). Therefore, in the case of a trailing stop loss, these points follow a profitable trading position while remaining unchanged for a losing trading position. A trailing stop loss of 5%, for example, will close the trading position if it is down 5% from its highest profit point. More volatile (2.2.2) assets require stop loss values with a greater difference to the current price due to bigger market movements. Generally, it can be said that stop loss strategies are more useful for assets with lower volatility since the loss of the trading position will be smaller when the stop value is reached.

Buy-and-Hold Strategy

The buy-and-hold strategy describes an investment plan that focuses on the long term [52]. An investor following this strategy buys an asset and, as the name suggest, holds it without an exit strategy or sell target in mind. This passive form of investment, mostly executed through buying and holding a diversified stock portfolio, recorded higher returns and lower risks over the past 15 years when compared with other popular investment strategies like managed investment funds [53].

Long & Short

The previously mentioned long and short specification of a trading position represents the expectation of an investor about the underlying assets development (p. 607-608) [31]. A long position is a bet on a positive market movement, while a short position speculates on the market moving downwards. Theoretically, long positions have an unlimited upside potential since the market value of the underlying asset can increase indefinitely. The potential loss, however, is limited since the asset price can not fall below zero. In other words, the value of a long position can only drop by a maximum of 100% – which represents a total loss – but can potentially increase way beyond plus 100%. Logically, short positions are the opposite: they have an upside that is limited to 100%

when the underlying assets market price drops to zero while their loss potential is, again only theoretically, indefinite.

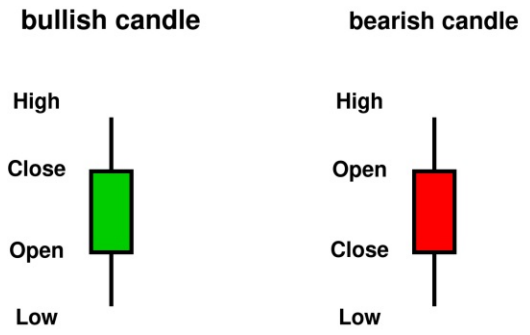
Bulls & Bears

In the context of financial investing, traders that enter long positions and thereby move the price upwards are known as bulls [54]. In contrast, investors that sell their long positions or enter short positions and consequently drive the price downwards are called bears [55]. An upwards moving market is also referred to as a bull market while a declining market is referred to as bear market. The stock market is therefore often illustrated by the battle between bulls and bears.

OHLC Candle

OHLC (open-high-low-close) candles are used in so called Japanese Candlestick Charts, which are the most prominent technique to display the historic price movements of an asset for a specific time unit like minute, hour, day, etc [31] (pp. 211-213). In such a chart, a single OHLC candle consists of a vertical line which marks the highest and lowest price during the selected time unit and a body illustrating its opening and closing price. The body can furthermore be green for a bullish candle or red for a bearish one. A bullish candle has a closing price above the opening price and a bearish candle vice versa. Figure 2.4 illustrates the two types of candles and shows where the open, high, low and close values can be found. The OHLC data format is often extended by the trading volume (V) of the candle to construct the OHLCV data format.

Figure 2.4: Japanese Candle Stick (OHLC), [31] (p. 212)



Market Trends

The market trend describes the pattern of market movements over a longer period of time for one specific asset, an industry, or the economy as a whole [30] (pp. 49-54). These patterns arise since the market is exposed to a certain level of volatility and is therefore moving in zigzag or wave patterns with obvious peaks and troughs and not in straight lines. Positive market patterns, in this sense, are typically called an upwards or bullish

trend while the opposite is referred to as a downwards or bearish trend. In the absence of a pattern due to counterbalancing movements or a lack of movements a sideways market trend is apparent. Figure 2.5 illustrates simplified examples of the described types of trends. The concept of identifying these trends is essential for market analysis as most corresponding analytical models build upon it.

Figure 2.5: Illustration of different market trends, [30] (p. 50)

Market Volatility

The term market volatility describes the degree of variation for the trading price over time. It is a statistical measurement that is typically calculated using variance and standard deviation and is often used as a means to measure risk [31] (pp. 585-586). A highly volatile asset, in this sense, is considered a risky investment since its price undergoes more significant changes and is therefore more likely to suddenly move contrary to an investor's expectations. However, a high volatility also provides more trading opportunities for experienced investors. On the one hand, if the long-term trend is identified correctly, skilled investors manage to record high gains by ignoring the short-term volatility. On the other hand, the presence of constantly changing directions with accompanying highs and lows makes room for short-term traders to prove their skill or try their luck.

Support & Resistance Levels

The concept of support and resistance levels is used to explain unique characteristics of certain price levels in technical analysis. The terms describe a specific price of an asset that acts as a barrier and prevents further price movements in a certain direction [30] (pp. 55-57). More precisely, a support level is a price range that prevents the asset from moving further downwards due to a concentration of demand. A resistance level, by contrast, is a price range with a strong sell pressure that prevents the asset from moving further upwards. Support and resistance areas are often price ranges where significant volume was traded in the past and can be identified with different technical methods.

Performance Metrics

Performance metrics are statistical methods used to analyze and evaluate the performance of trading strategies. In this section, the most relevant ones are outlined and elaborated.

- **Return on Investment (ROI)**
  The return on investment, short ROI, measures the profit or loss associated with a certain investment over a specified time span [31] (p. 564). Its simplicity and comparability make it a popular indicator to analyze the performance of a trading strategy. ROI can be calculated with the following simple formula:

$$ROI = \frac{Current\ Value\ of\ Investment - Initial\ Investment}{Initial\ Investment} \tag{2.1}$$

- **Sharpe Ratio**
  The sharpe ratio is one of the most popular performance measurements for portfolios and trading strategies [31] (p. 551). The metric was developed by and named after the Nobel Prize Winner in Economic Sciences William F. Sharpe and is used to calculate the risk to reward ratio of an investment [56]. The sharpe ratio is defined as:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p} \tag{2.2}$$

  where:
  R_p is the return of the investment
  R_f denotes the risk free rate
  σ_p is the standard deviation of the portfolio's daily returns.

  The risk-free rate in this formula is the return an investor would expect when investing in an alternating asset that does not carry any risk. Although there is not a single correct value for the risk-free rate, it is most common to apply the interest rate of the us treasury bond or the return of a diversified low-risk ETF.

  The standard deviation, as explained above, measures the volatility of an asset, and is therefore used as the risk factor. It can be calculated with the following formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)}{n-1}} \tag{2.3}$$

  where:
  n is number of data points
  μ is the mean value for the data points
  x_i is the value of the i-th data point

- **Sortino Ratio**
  The sortino ratio is an enhanced version of the sharpe ratio and was introduced to

overcome the main flaw associated with it [31] (p. 551). While the sharpe ratio uses both the up- and downside volatility to account for the risk, the sortino ratio only considers the downside volatility. Therefore only downside movements have a negative effect on the ratio. The derived formula to calculate the sortino ratio is:

$$Sortino\ Ratio = \frac{R_p - R_f}{\sigma_n} \tag{2.4}$$

where:
σ_n is the standard deviation of the portfolio's negative returns.

- **Maximum Drawdown**
  A maximum drawdown (MDD) measures the decline of an investment, trading account, or fund from its peak to its trough during a specific period [31] (p. 563). It is analyzed for risk measurement and usually quoted as the percentage change between the peak and the subsequent trough. For example, a trading account of $10,000 that drops to a low of $9,000 before increasing again witnessed a 10% drawdown. The MDD can be calculated with the following formula:

$$MDD = \frac{Trough\ Value - Peak\ Value}{Peak\ Value} \tag{2.5}$$

All these listed performance metrics are important to evaluate and test different kinds of trading strategies which are discussed in the next section.

2.2.3 Analytical Methods for Trading Strategies

This section highlights the three prevalent methods used to predict the future price level of an asset or market. Each of these methods is named after the respective data that is analyzed.

Technical Analysis

Technical analysis is the most commonly used technique for short and mid-term price prediction by non-professional traders. This type of analysis focuses on different analytical methods of historic price and volume movements. It is split into multiple approaches where the most popular ones include the Eliot Wave Theory, the Fibonacci retracements, Chart patterns, and Technical Indicators [30].

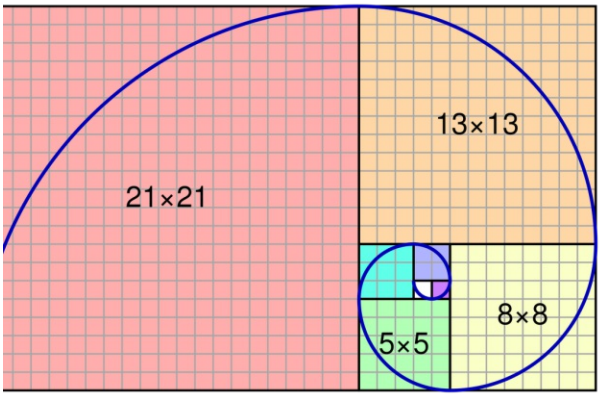
The Eliot Wave Principle is a technical analysis method that is based on the psychology of investing. It was invented by Ralph Nelson Elliott and first published in the book *The*

Wave Principle in the year 1938 [57]. Elliott believes that the market moves in trend- and correction waves which are caused by the changing psychology of investors.

Fibonacci retracements are based on the Fibonacci numbers, which represent a famous mathematical sequence and can be found in a variety of everyday areas of life and nature like flower petals, tree branches, shells, and even human faces. Fibonacci retracements use this sequence to identify strong support and resistance areas in the chart to predict future movements [30] (pp. 493-497). Furthermore, they are applied in the Eliot wave theory to forecast the size of upcoming waves based on current and historic ones. The Fibonacci sequence can be illustrated with the circle pattern displayed in figure 2.6 and calculated with the following formula:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad (\text{for } n > 1) \tag{2.6}$$

Figure 2.6: Fibonacci sequence pattern, smithsonianmag.com [58]

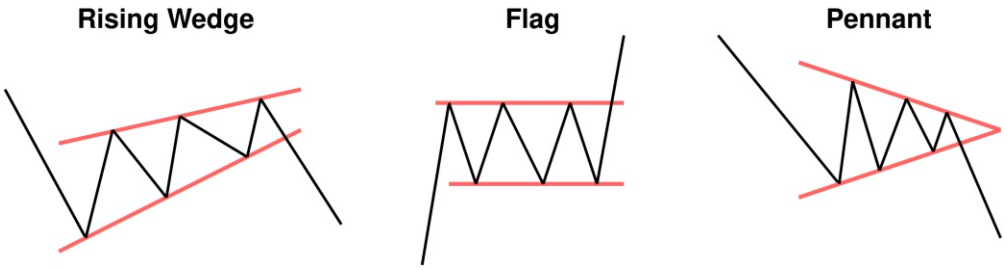


The next approach, known as chart patterns, describes naturally occurring and repeating patterns that can be found in the price charts of many different assets [31] (pp. 302-305). Chart patterns are identified by connecting relevant historic price points and drawing so called trendlines. A certain sequence of these trendlines then reveals the chart patterns that illustrate the bigger picture. The most common occurring patterns are pennants, flags, and wedges. An example of them can be found in figure 2.7.

These 3 techniques require a lot of experience and are mostly exercised by directly working with the price chart of an asset. However, there are multiple ways to interpret and draw trendlines and different patterns. Therefore, the same price chart does not have one universal message but can be translated differently by investors, which makes these techniques difficult to automate.

Technical indicators on the other hand are clearly defined mathematical calculations which are applied to the historic trading data [59]. Manual traders use the results of

Figure 2.7: Illustration of popular chart patterns, tradingview.com [38]



these calculations as an additional visual input for their trading decisions by displaying them within their trading program. Since these indicators have a clearly defined output for every input value, they can easily be integrated into automated trading strategies. There are numerous different technical indicators which can mostly be grouped into 5 categories: overlap studies, momentum, volume, volatility, and statistical [60]. The most important indicators are presented below.

Technical Indicators: Overlap Studies

The first group of indicators, called the overlap studies, uses the same scale as the price data and is mostly analyzed by being directly compared to the price of an asset [60]. The group is commonly split into the two subgroups trend indicators and mean reversion indicators. Trend indicators are used to confirm and predict the current price trend, while mean reversion indicators are used to predict trend changes. Manual traders usually print the indicators directly on the price chart for an easy visual comparison. When implemented into an automated trading system the difference between the assets price and the indicators result is used as input data to derive trading strategies.

Overlap studies mostly describe different kinds of Moving Averages (MA), which are among the most famous, versatile, and widely used technical indicators [31] (pp. 275-285). Simply put, all these indicators are just calculating an average over the selected number of past price ticks. This number of price ticks is an optional and variable parameter setting for the indicators with a default value of 30. By visualizing the results of these moving averages, a smoothed and lagging price is presented. The number of input ticks can be increased to further smoothen and lag the price. Figure 2.8 displays two different settings for a simple moving average to illustrate the different smooth levels and the price lags. Moving averages come in many versions, varying in how they affect the smoothness of the result by assigning a different weight to more recent price ticks. A list of these variations includes Simple MA (**SMA**), Weighted MA (**WMA**), Exponential MA (**EMA**), Double Exponential MA (**DEMA**), Triple Exponential MA (**TEMA**), and Triangular Moving Average (**TRIMA**), to name the most relevant ones [31] (pp. 275-285). The purpose of moving averages is to identify and follow more long- term trends. Therefore, they are also viewed as a curving trendline.

The different moving averages can be calculated using the following formulas:

$$SMA_x = \sum_{i=x-n+1}^{x} p_i * \frac{1}{n} \tag{2.7}$$

$$EMA_x = p_x * \frac{2}{n+1} + EMA_{x-1} * (1 - \frac{2}{n+1}) \tag{2.8}$$

$$WMA_x = \sum_{i=x-n+1}^{x} (p_i * i) * \frac{n*(n+1)}{2} \tag{2.9}$$

$$DEMA_x = 2 * EMA_x - EMA(EMA_x) \tag{2.10}$$

$$TRIMA_x = SMA(SMA_x) \tag{2.11}$$

where:
p_x is the price of the current tick
n is the period length

Figure 2.8: Comparison of SMA 20 and SMA 100, tradingview.com [38]

Bollinger Bands, developed in the 1980s by John Bollinger, is a forecasting method utilizing SMA [61]. The approach places two bands around a moving average to encapsulate the price as illustrated in figure 2.9. While there are variations, the default configuration places the bands' two standard deviations above and below the moving average. The standard deviation, in this setting, describes how dispersed the prices are around an average value. Using two standard deviations ensures that 95% of the prices

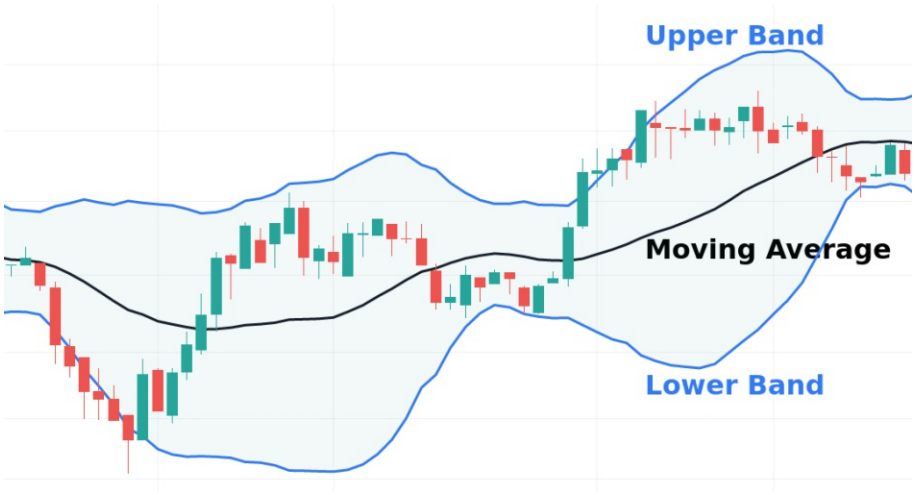
are within the Bollinger Bands. When a price then reaches the top of the Bollinger bands or increases beyond that, it is seen as overbought, meaning the assets inherent value is lower than its market value. The opposite is true when the price decreases below the lower band, suggesting an asset is oversold. Therefore, this indicator is useful to identify overreactions of the market. Interestingly, the behavior of Bollinger Bands is affected by the market's underlying volatility. With an increased volatility, the gap between the bands grows, allowing bigger price movements before an oversold/overbought signal can be identified. A lower volatility results in contracting bands respectively. Furthermore, Bollinger Bands can be used for trend trading with the upper and lower band set as the price target. The formula to calculate the upper and lower band is the following:

$$Upper\ Band_x = SMA_x + m * \sigma_n \tag{2.12}$$

$$Lower\ Band_x = SMA_x - m * \sigma_n \tag{2.13}$$

where:
m is the number of standard deviations
σ is the standard deviation over the last n ticks

Figure 2.9: Illustration of Bollinger Bands, tradingview.com [38]



Other more uncommon indicators in the group of overlap studies include the **T3** Moving Average, Kaufman Adaptive Moving Average (**KAMA**), MESA Adaptive Moving Average (**MAMA**), the parabolic **SAR** indicator as well as the Hilbert Transform Instantaneous Trendline (**HT Trendline**) [31] (p. 285) [30] (pp. 381-384) [60].

Technical Indicators: Momentum

The second group of indicators is called momentum indicators [31] (pp. 430-432). These indicators measure the momentum of an underlying asset, in this case the rate of change for the price or volume data. They are useful to detect trends, identify overbought or undersold extremes in the markets, and reveal strengths and weaknesses in the data. Momentum indicators use different value ranges as the price. Therefore, a separate chart is needed for visualization as they cannot be displayed directly on the price chart. Some of the most important momentum indicators are listed below:

MOM is the most basic momentum indicator and was among the first who were invented [30] (pp. 228-229). It simply calculates the price change between the current price and a historic price point. The formula for MOM is defined as:

$$MOM_x = price - price_{x-n} \tag{2.14}$$

Similar to the MOM indicator, **ROC** was one of the first indicators invented to identify an assets momentum [30] (p. 234). ROC – the abbreviation for Rate of Change – hereby measures the speed of which the price changes for a specified period of time. Mathematically, this can be described as the percentage change between the current price and a historic price point. The formula to calculate the Rate of Change is:

$$ROC_x = \left(\frac{price_0}{price_x} - 1\right) * 100 \tag{2.15}$$

CCI, short for the Commodity Channel Index by Donald R. Lambert, assesses the trend direction and trend strength of an assets price [30] (pp. 237-239). Investors apply the CCI to identify an overbought or oversold area by analyzing the historical values of the indicator. Specifically, it is used to measure the difference between the latest closing price and the moving average over a specified period of time. Afterwards, the result is normalized by a division through the mean deviation. The formula to calculate the indicator is:

$$CCI_x = \frac{TP_x - SMA(TP_x)}{0.015 * mean\ deviation} \tag{2.16}$$

$$TP_x = \frac{high + low + close}{3} \tag{2.17}$$

The relative strength index (**RSI**) is one of the most popular technical indicators [30] (pp. 239-246). It was developed by J. Welles Wilder Jr and first presented in the book *New Concepts in Technical Trading Systems* [62]. As other momentum indicators, it is used to measure the magnitude of historic price changes to identify overbought and oversold price ranges. Beyond that, it solves two problem associated with most other indicators: Firstly, earlier indicators showed too much sensitivity to sharp price changes, resulting in exaggerated reactions. The relative strength index balances such reactions by applying smoothing methods. Secondly, for comparison purposes, a constant range is needed. To approach this issue, a constant vertical range of 0 to 100 is created by the formula for RSI:

$$RSI_x = 100 - \frac{100}{1 + RS_x} \tag{2.18}$$

$$RS_x = \frac{average\ of\ previous\ n\ ticks\ up}{average\ of\ previous\ n\ ticks\ down} \tag{2.19}$$

Another famous indicator is the **MACD**, short for moving average convergence divergence [30] (pp. 252-254). MACD was developed by Gerald Appel and shows the relationship between a fast and a slow exponential moving average. Trading signals are mostly triggered by a crossover between the two EMAs, where the speed of the crossover depicts an important factor used to check if the bullish or bearish movements are varying in intensity. To calculate the MACD indicator the following formula is applied:

$$MACD_x = fast\ EMA_x - slow\ EMA_x \tag{2.20}$$

The Stochastic K%D indicator (**STOCH**) is based on the observation that in times of positive price movements, the closing price tends to be closer to the upper end of the current price trend and during a price decrease the closing price tends to be closer to the lower end of the current price trend [30] (pp. 246-249). The indicator is based on two lines, %K and %D, where %D is mainly used for trading signals. The following formula is used for its calculation:

$$\%K_x = 100 - \left[\frac{close_x - LOW_n}{HIGH_n - LOW_n}\right] \tag{2.21}$$

$$\%D_x = MA_m(\%K) \tag{2.22}$$

where:
LOW_n is the lowest price over the last n ticks

$HIGH_n$ is the highest price over the last n ticks
MA_n is a moving average over the period m

The Percentage Price Oscillator, short **PPO**, shares many similarities to the MACD [63]. In contrast, however, the type of the respective Moving Average can be changed. Furthermore, the relationship of the two moving averages is calculated as percentage with the formula:

$$PPO_x = \frac{fast\ MA_x - slow\ MA_x}{slow\ MA_x} * 100 \tag{2.23}$$

The Balance of Power indicator, abbreviated **BOP**, is an oscillator used to measure the strength of buying and selling pressure [64]. Introduced by Igor Levshin in the *Technical Analysis of Stocks & Commodities* magazine, issue August 2001 [65], this indicator compares the power of buyers to push prices to higher extremes to the power of sellers to move prices to lower extremes. An indicator in the positive range indicates that the bulls are in charge, while an indicator in the negative range implies the opposite. A result near zero indicates a balance between the two and can signal a trend reversal. The underlying formula for the Balance of Power indicator is:

$$BOP_x = SMA\left(\frac{close_x - open_x}{high_x - low_x}\right) \tag{2.24}$$

Next to the listed and explained most relevant momentum indicators, there exists a wide range of less relevant indicators. Examples are ADX, ADXR, APO, AROON, AROONOSC, CMO, DX, MFI, MINUS DI, MINUS DM, PLUS DI, PLUS DM, Larry Williams %R, STOCH F, STOCH RSI, TRIX, and ULTOSC. For the sake of completion, this thesis refers to the documentation of the talib library for further information [60].

Technical Indicators: Volume

The third group of indicators, namely the volume indicators, have volume as their main input source. Like all other indicators, they are used to make predictions about the future of an asset. However, instead of the price or the trend, they help to forecast the volume. The most relevant examples of volume indicators are listed below.

The Accumulation/Distribution Indicator (**AD OSC**) is based on volume and price data to check whether an asset is being accumulated or distributed [31] (p. 419). More specific, the indicator identifies divergences between the volume flow and the price trend. If, for example, the price of an asset increases while its accumulation volume does not,

the higher price is not supported. Consequently, the indicator will fall and suggest the likelihood of a price decline. The following formulas are used to calculate the A/D indicator:

$$Money\ Flow\ Multiplier_x = \frac{(close_x - low_x) - (high_x - close_x)}{high_x - low_x} \tag{2.25}$$

$$Money\ Flow\ Volume_x = Money\ Flow\ Multiplier_x * volume_x \tag{2.26}$$

$$AD_x = AD_{x-1} + Money\ Flow\ Volume_x \tag{2.27}$$

The On Balance Volume indicator, short **OBV**, is a simple yet particularly popular technical indicator used to predict the volume of an asset [30] (pp. 165-166). It was developed by Joseph Granville and published in the book *Granville's New Key to Stock Market Profits* in 1963 [66]. Granville believed that the volume is one of the main factors influencing market movements. Utilizing OBV, investors calculate the momentum of the volume and usually display the result side by side with the asset price. Generally, the indicator moves in the same direction as the price. If the trends misalign, however, the OBV trend is used to suggest a future trend change for the price. This is due to the fact that, according to Granville, buying or selling pressure is usually detected first in the volume and only later in the price. To calculate the OBV, the following formula can be used:

$$OBV_x = OBV_{x-1} + \begin{cases} volume_x, & \text{if } close_x > close_{x-1} \\ 0, & \text{if } close = close_{x-1} \\ -volume_x, & \text{if } close_x < close_{x-1} \end{cases} \tag{2.28}$$

Technical Indicators: Volatility

The fourth group of technical indicators is called volatility indicators. As the name suggests, these indicators are used to measure the price volatility of an asset. Volatility is considered a niche indicator, resulting in a lower number of applicable examples. As the most relevant ones, the true range indicator (**TR**) and the average true range indicator (**ATR**), invented by J. Welles Wilder Jr., are presented in the book *New Concepts in Technical Trading Systems* [62]. The true range indicator is utilized to calculate the trading range of the last tick, which is defined as the range between the highest and lowest trading price within a specified period. Respectively, the ATR is applied for the average trading range over the last n periods.

The formulas are defined as:

$$TR_x = MAX[(high_x - low_x), ABS(high_x - close_{x-1}), ABS(low_x - close_{x-1})] \quad (2.29)$$

$$ATR_x = \left(\frac{1}{n}\right) \sum_{i=x-n}^{x} TR_i \quad (2.30)$$

Technical Indicators: Statistical Functions

The last group of indicators, known as statistical indicators, contains statistical functions like Beta Coefficient (BETA), Pearson's Correlation Coefficient, Linear Regression, Linear Regression Angle, Linear Regression Intercept, Linear Regression Slope, Standard Deviation, and Variance. However, these indicators are not very popular and rarely used by the typical trader. They can be used for more advanced trading strategies but also require a deep understanding of the underlying statistical functions [60].

Fundamental Analysis

Next to technical analysis, fundamental analysis represents another method used to forecast the future price level of an asset or market. This type of analysis focuses on the economic forces of supply and demand and examines all relevant factors affecting the price to determine the intrinsic value of an asset [30] (pp. 5-6). The intrinsic value, in this sense, is what any asset like a company stock is actually worth according to its fundamentals. An asset with a market price above its intrinsic value is considered overpriced, while a market price below the intrinsic value depicts an underpriced asset. Fundamental information to assess and analyze an intrinsic value includes earning reports, information about the competition of a company, a company's management, and basically any other available data about a company. Logically, according to the efficient market hypothesis, the current market price already includes all available information and should therefore always correspond to the intrinsic value. Consequently, challenging the efficient market hypothesis with fundamental analysis can only be possible by an investor who utilizes secret information or has superior interpretation skills to deduce a deviation between an asset's intrinsic value and its market value. A glance into the past suggests the existence of such deviations, as some of the most dramatic bull and bear runs in history began with no or little fundamental changes.

For cryptocurrencies, fundamental information includes the current coin supply, the hash rate for proof of stake coins, the distribution of the coins e.g. the presence of big whales, the number of used addresses, the number of transactions, the transaction flow for big transactions to and from exchanges, and basically any data that is directly or indirectly

linked to the cryptocurrency. Overall, fundamental analysis is a highly manual, creative, and interpretive method that is nearly impossible to automate.

Sentiment Analysis

The last forecasting method, called sentiment analysis and also known as opinion mining, is the process of extracting the emotions, mood, and sentiment of a given text to deduce its impact [31] (pp. 89-94). One big advantage of this approach is the potential degree of automation, since the process can be utilized in combination with different natural language processing (NLP) techniques [67]. In the field of stock prediction, it is used to analyze texts which are related to the specific asset in question. Some example text sources for stock predictions are earning results, reports, news articles about the company or the market it is operating in, and social media feeds like Facebook, Twitter, Reddit, or Stocktwits. For cryptocurrencies, the sources are scarcer and mostly include news articles and social media feeds.

There are different techniques which can be used for sentiment analysis. The simplest one is the bag of words model, which uses a sentiment dictionary that assigns a sentiment value to each word [68]. To calculate the sentiment of a given text with the bag of words model, each word is replaced by its corresponding sentiment value from the dictionary. Afterwards, the sum is calculated over all words to deduce a final sentiment value for the text. This model has many flaws since it does not understand a text's meaning, its context, and any complex phrases. Therefore, more sophisticated techniques were developed which use machine learning and deep neural networks for more accurate predictions. Over the last three years, significant progress was achieved within the NLP sector by utilizing a new deep neural network architecture called transformer. A detailed introduction to this architecture can be found in section 2.3.7. In summary, sentiment analysis has become more important due to technical advancements that help derive more accurate meaning from enormous amounts of text.

2.3 Artificial Neural Networks (ANN)

This section starts off with a description of the term machine learning, followed by an introduction into the basics of neural networks. Afterwards, specific methods to improve and adjust the learning process of neural networks are outlined. The last part of this section introduces multiple neural network architectures and their respective advantages. The last part of this section will introduce multiple neural network architectures and their respective advantages.

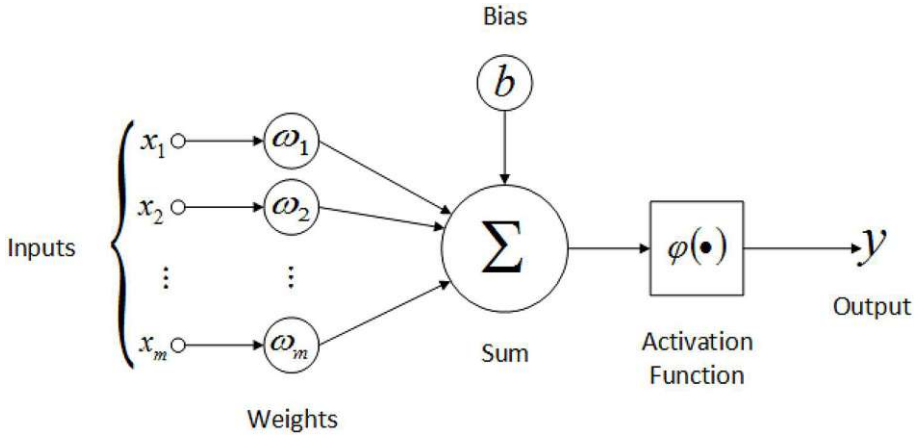
2.3.1 Machine Learning & Artificial Neural Networks

'Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.' – Arthur Samuel, 1959 [69]

Machine Learning, commonly abbreviated ML, belongs to the field of artificial intelligence of computer science and is a method to build algorithms without explicitly programming them [70]. In the field of machine learning so called models are built, which are then fed information about an existing problem. These models then automatically identify patterns to solve the previously defined problem during the so-called training process. These trained models can afterwards apply the learned patterns to new data to predict possible solutions. Or, simply put, these models learn from input data and then solve complex problems based on the experience they gathered. Therefore, machine learning is specifically beneficial in all use cases that have enormous amounts of data to analyze. Consequently, it is applied to various fields across many industries, today's examples include self-driving cars, genomics analysis, speech recognition, natural language processing, fraud detection, and many more [70].

Artificial neural networks are a subset of the machine learning models and are based around the idea of the human brain [71] (pp. 277-281). A neural network is a system of neurons which interact with each other to construct complex mathematical formulas. More specifically, each neuron in the system may have several incoming connections from other neurons, input data from the outside world as well as a bias value. To calculate the output of a neuron, a weighted sum over all inputs is calculated first, while the weights and the bias values are adjusted during the learning process. Afterwards, an activation function is applied to obtain the final output for the neuron. An illustration of this calculation is provided in figure 2.10. A more detailed introduction to the different activation functions follows in chapter 2.3.3.

Figure 2.10: Illustration for neuron calculation, towardsdatascience.com [72]



Furthermore, neural networks are composed of multiple layers of nodes [71] (pp. 286-289). The first layer is called the input layer and receives input data from the outside world. The last layer, known as the output layer, returns the prediction of a trained model. All layers in between these two are called hidden layers, the number of which is flexible and depends on the complexity of the neural network. Any network with more than one hidden layer is referenced as a deep neural network.

2.3.2 Neural Network Introduction

In this section, fundamental elements of neural networks are explained building on the information of 2.3. Starting off with the input data for neural networks, also called training data or features, which is one of the most important parts of the neural network [73] (pp. 19-21). This data contains all the existing knowledge concerning the problem the network is intended to solve. To achieve maximum efficiency during the training process, the input data must be cleaned and preprocessed first [71] (pp. 66-68) [74]. Typical cleanup steps include the removal of outliers and duplicate values. Preprocessing, on the other hand, usually covers shuffling the input data to reduce biases, scaling the input data to enhance learning, as well as feature selection, elimination, and expansion.

Feature selection and feature elimination are processes to eliminate bad and unnecessary features from the input data set [71] (pp. 26-28). This procedure can significantly reduce the input space and subsequently the required computational power as well as the time required for the training process. Some approaches, for example, use statistical methods to analyze the relationship between existing features to identify and remove unnecessary features. Some neural networks also include specific layers to reduce the dimensionality of input data and thus indirectly perform a feature selection process. For example, a CNN layer (2.3.7) might be used with this intention. Feature expansion, on the other hand, requires methods to transform and combine existing features with the objective of finding new ones with a superior predictive ability. A technical indicator, for example, is a feature obtained by using existing features like price and input data. [71]

The importance of the feature engineering process for a machine learning project, including all actions to identify the right set of features as well as to appropriately preprocess the input data, is summarized eloquently in the common saying *'Garbage in, garbage out'* [71] (pp. 27).

Next to the management of input data, the training process of a neural network represents its next crucial part. In this sense, machine learning systems can be classified according to the extensity and the type of supervision that occurs during this process. There are four major categories, namely supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [71] (pp. 8-15). Supervised learning describes a method where the training data includes not just the problem but also its solution in form of a label or target value. With unsupervised learning, by contrast, algorithms do not require any information about the desired output. This approach is used to discover unknown patterns within the provided data. Semi-supervised learning is a combination of the two prior methods and only requires a small amount of labeled input data. The last approach, reinforcement learning, shows more fundamental differences to the algorithms above. It works with a reward system and tries to maximize the achieved reward during the learning process [75] (p. 66). Games are a classic example, as they are often solved with reinforced learning and have a strictly defined set of rules. For the specific case of this thesis, a neural network solution with a supervised training model was created. Therefore, more detailed corresponding research for only this approach is provided to not

cross the scope of this thesis.

The training process for supervised neural network models, also known as learning process, adjusts the weights of each neuron with the goal to reduce the prediction error. This prediction error is calculated by the loss- or cost function that describes the degree of deviation the achieved results have from the optimal results [76]. Logically, a prediction error of zero would be a perfect model without any deviation from the optimal results. The prediction error is propagated from the back to the front of the network by the backpropagation algorithm [77]. Furthermore, the weights are adjusted during the backpropagation process by the optimizer function. A more detailed explanation of different optimizer functions can be found in 2.3.4.

The following sections go into more detail regarding the diverse concepts of neural networks. Section 2.3.3 presents different kinds of activation functions and outlines their advantages as well as disadvantages. Next, the section 2.3.4 describes different loss and optimizer functions. Moreover, in the sections 2.13 and 2.3.6, methods to avoid overfitting and underfitting are described as well as a process to optimize the hyperparameters of a neural network.

2.3.3 Activation Functions

Activation functions represent a critical part of neural networks and are mathematically expressed with the Greek letter ϕ [71] (pp. 326-332). As described in the previous section, they calculate the output of a neuron by using the weighted sum of the incoming connections as their input. Furthermore, they are used to shrink the output space of the neurons. Overall, there are numerous different activation functions with properties that vary in their complexity. The most commonly used ones are presented in the following list:

- **Linear or Identity**
  The linear or identity activation function is the most basic one and simply returns the input value directly as the output value [78]. Nowadays, it is often replaced by more sophisticated activation functions, but it is still relevant in the output layer for regression problems. The identity activation function is defined as:

$$\phi(x) = x \tag{2.31}$$

- **Step**
  The next activation function, called the step activation function, is a simple threshold function [78]. More specifically, it switches the output from 0 to 1 once the input exceeds the threshold of 0.5. It is useful for binary classification problems and can be calculated with the following formula:

$$\phi(x) = \begin{cases} 1, & \text{if } x \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{2.32}$$

- **Sigmoid**
  The sigmoid function is a s-shaped activation function that transforms the inputs to values between 0 and 1 [79]. Therefore, it is usually used when a positive output value is required. However, the sigmoid function has barely an effect on the predictions for very high or very low inputs since these values are always transformed to an output near 0 or 1. Ultimately, this leads to a neural network that is unable to learn any further, which is known as the vanishing gradient problem [80]. The sigmoid function is defined by:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.33}$$

- **Hyperbolic Tangent**
  The hyperbolic tangent is yet another s-shaped activation function [79]. In contrast to the sigmoid function, however, it transforms the inputs to a range between -1 and 1. Therefore, it holds a slight advantage over the sigmoid function since it is able to also learn from negative inputs. Apart from that it suffers from the vanishing gradient problem as well. To calculate the hyperbolic tangent, the following formula is used:

$$\phi(x) = TANH(x) \tag{2.34}$$

- **Rectified Linear Unit (ReLU)**
  Rectified linear units, short ReLUs, were developed by Teh & Hinton [81] in 2000 and quickly became the most common activation functions. Contrary to the sigmoid or hyperbolic tangent functions, the rectified linear unit does not saturate to -1, 0, or 1, which solves the vanishing grading problem and thereby results in a superior performance. Accordingly, RelUs and their modified versions are the activation function recommended for most use cases by current research. The following formula defines the rectified linear unit:

$$\phi(x) = MAX(0, x) \tag{2.35}$$

  All activation functions below represent modified versions of the ReLU.

- **Leaky Rectified Linear Unit (Leaky ReLU)**
  The leaky rectified linear unit is a modification that, unlike the original, also allows for small negative values [79]. To calculate the leaky ReLU, the following formula is used:

$$\phi(x) = \begin{cases} 0.01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.36}$$

- **Exponential Linear Unit (ELU)**
  The second modification of the standard ReLU is called exponential linear unit, or ELU for short [82]. It constitutes a generalization of the ReLU that is based on a parameterized exponential function. Therewith, it progresses from small negative values to positive ones. The following formula defines the exponential linear unit:

$$\phi(x) = \begin{cases} \alpha(e^x - 1), & \text{if } x \leq 0 \\ x, & \text{otherwise} \end{cases} \tag{2.37}$$

  with $\alpha > 0$

- **Scaled Exponential Linear Unit (SELU)**
  The scaled exponential linear unit represents a modified version of the ELU, where both α and λ are fixed parameters which are derived from the input data [83]. A calculation of the SELU requires the following formula:

$$\phi(x) = \lambda \begin{cases} \alpha(e^x - 1), & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.38}$$

  For standard scaled inputs, which is a mean of 0 and a standard deviation of 1, the values are $\alpha = 1.6732$, $\lambda = 1.0507$

- **Softmax**
  The softmax activation function calculates a normalized probability distribution for its inputs [79]. It is mainly used within the output layer of multinomial classification- or multiclass problems and is the only activation function applicable to these kinds of problems. To calculate the softmax function, the following formula is used:

$$\phi(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J}} \text{ for } i = 1, ..., J \tag{2.39}$$

2.3.4 Loss Functions, Optimizers & Learning Rate

This section provides an overview of the loss- and optimizer functions as well as the learning rate. Furthermore, the most relevant functions are outlined and defined with their corresponding formulas.

As mentioned in 2.3.2, a loss function is required to calculate the current error of the neural network, which is expressed as the degree of deviation between the achieved and the optimal results. The loss function is selected based on the type of problem that must be solved. For the case of regression problems, the most used loss function is called the mean square error (MSE) [84]. It represents an estimator function which measures the average squared difference between the current and the desired solutions. For binary classification problems, the preferred loss function is the binary crossentropy (BCE) [71]

(pp. 150-151). For multinomial classification problems, accordingly, different types of multiclass crossentropy functions are used. Entropy is defined as a measure of uncertainty for a given probability distribution. In detail, the crossentropy functions compare each predicted logarithmic probability to their actual class output, which can be either 0 or 1. Afterwards, it calculates the loss value based on the deviation between these values.

Optimizer functions, as introduced in 2.3.2, are necessary to recalculate the weights during the learning process. By default, they build upon the gradient descent approach, which is used to identify local minima [77]. However, there are various different and improved methods for the optimization process.

One parameter of the optimizer function deserves special attention. The so-called learning rate (LR) is a specification that determines how much the weights are changed during each iteration of the training process [71] (pp. 119-123). A smaller LR leads to smaller adjustments, resulting in a longer training time until the optimal solution is identified. A higher LR, by contrast, brings about bigger weight changes and thereby decreases the training time. This, however, leads to less sensitivity and might overshoot and miss the optimal solution. Consequently, it is highly important to identify a good learning rate, which made it a heavily discussed research field. Common approaches include the testing of different LRs with methods like hyperparameter optimization 2.3.6, or the usage of advanced optimizer functions which are able to adjust the LR automatically during the learning process. An illustration of a slow, a fast, and a decent learning rate in combination with the gradient descent approach is displayed in figure 2.11.

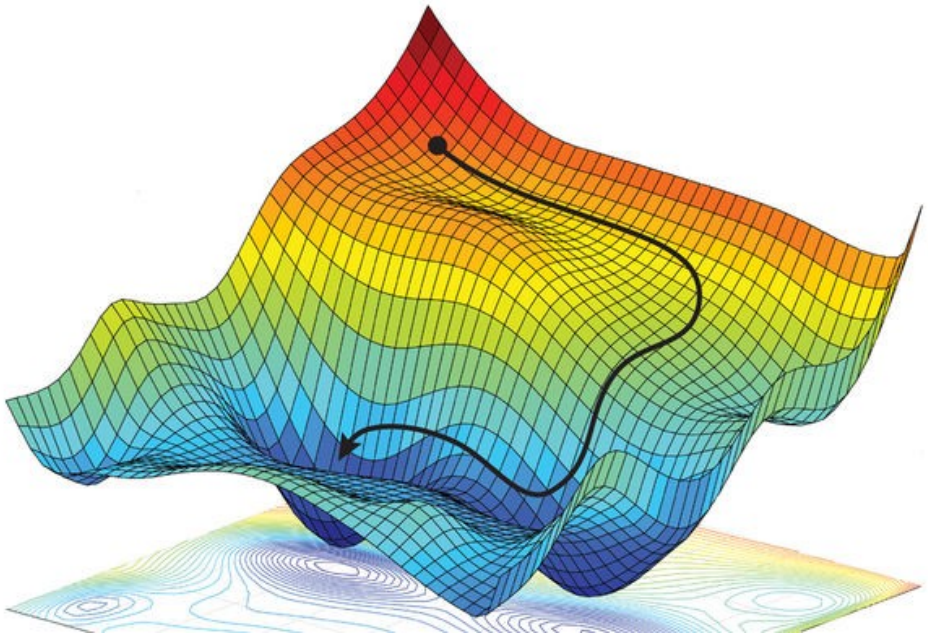
Figure 2.11: Illustration of different learning rates, morioh.com [85]

Overall, optimizer functions and their corresponding learning rates represent a critical part of artificial neural networks. The list of optimizer functions is long and research for superior functions is always ongoing. The currently most common ones, including a short description of their properties, are listed below:

- **Gradient Descent**
  Gradient Descent is the standard optimization algorithm used for machine learning problems [71] (pp. 119-123). In general it calculates how the weights should be altered so that the function can move towards a local minimum. It is an iterative

algorithm which calculates the gradient for the current settings of the model. During each iteration the weights are adjusted so that the loss moves in the direction of the steepest descent which is defined by the negative of the gradient. The speed of this descent is defined by the learning rate described above. Figure 2.12 illustrates this iterative approach for a 3-dimensional problem. Furthermore, all of the following algorithms built upon the idea of gradient descent.

Figure 2.12: Illustration of gradient descent approach, acoldbrew.medium.com [86]



- **Stochastic Gradient Descent (SGD)**
  Stochastic gradient descent, abbreviated SGD, is a slightly improved form of the gradient descent algorithm achieved by updating the weights of the model more frequently [77]. The default gradient descent method performs an update after the entire dataset has been fed to the model. By contrast, the SGD algorithm updates the weights after each training sample.

- **Adaptive Gradient Algorithm (AdaGrad)**
  The adaptive gradient algorithm, short AdaGrad, was published by Duchi et al. in 2011 [87] and owes its name to the ability to automatically adapt its learning rate. As previously mentioned, a constant learning rate is suboptimal for efficiently finding the ideal output. The adaptive gradient algorithm solves this problem by decreasing the learning rate for each parameter during the training process.

- **AdaDelta**
  AdaDelta is an extension of the AdaGrad algorithm, which constantly decreases the learning rate in a monotonical way [88]. This can result in a slow learning time or even prevent ever reaching the optimal result when the learning rate decays too quickly. Adadelta, on the other hand, uses the previous gradients of a fixed window size. Consequently, the learning rate will only decrease when approaching a local minimum.

- **Root Mean Square Propagation (RMSprop)**
  Just like Adadelta, the root mean square propagation, RMSprop, was developed to overcome Adagrad's problem of radically diminishing learning rates [89]. Despite being designed independently from Adadelta, the algorithm shares many similarities with its simultaneously developed counterpart.

- **Adaptive Moment Estimation (Adam)**
  The adaptive moment estimation, short Adam, is currently among the most popular optimizers [90]. It combines the advantages of SGD, RMSprop and momentum strategies, as it uses momentum to compute an adaptive learning rate for each parameter. Additionally, Adam uses the squared gradients for the scaling of the learning rate similar to RMSprop and a moving average over the past gradients for its momentum calculation.

- **Nesterov-Accelerated Adaptive Moment Estimation (Nadam)**
  The last common optimizer function is the Nesterov-accelerated adaptive moment estimation, or Nadam [91]. It shares many similarities to its origin, the Adam, while being extended by a Nesterov momentum. The Nesterov momentum optimizer solves a problem related to the order of the gradient step execution and the momentum calculation for the standard momentum optimizer. The details about this problem exceed the scope of this thesis but can be found in a paper by Dozat [91].

2.3.5 Overfitting & Underfitting

This section highlights two of the most common problems associated with neural network models, namely overfitting and underfitting [71] (pp. 28-31). Overfitting describes the condition that a model performs extraordinarily well on the training data set but poorly on the test and validation data. In other words, the model learned the data during the training too well and is therefore not able to generalize the learned behavior for new and unknown data. There are multiple reasons why overfitting may occur and various methods to avoid it called regularizations.

One common cause for overfitting is too much training time. The longer the model is trained, the better it will learn the specific training data, distorting its ability to pick up and apply general experience [92]. Consequently, it is important to stop the training process when the model has only determined a generalized solution of the training data

but has not yet memorized every detail of the data. To cancel the training process as soon as the model shifts into a state of overfitting, a method called early stopping was invented [93]. This method can be described as a delayed stop function which monitors if the loss value for the test data keeps decreasing. If no improvement is noticed for multiple iterations, the method automatically stops the training process.

Another common and computationally inexpensive technique to avoid overfitting is called dropout [71] (pp. 357-359). Introduced by Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov in 2012 [94], the dropout regularization algorithm works by temporarily and randomly removing neurons and connections from the neural network. Thereby, it causes hidden neurons of the neural network to be unavailable during some parts of the training. Consequently, the neural network relies solely on the remaining portion to still achieve a satisfying score, which decreases the likelihood of co-adaption between neurons and ultimately results in less overfitting.

The third approach to avoid overfitting is based on the so-called L1 and L2 regularization methods [95] (pp. 204-208). These methods add a weight penalty to pre-defined neural network layers during the training process, which discourages the neural network from using large weights and from the inclusion of certain irrelevant inputs. Lower weights will typically lead to less overfitting. Both algorithms work differently in how they penalize the size of a weight. In detail, the L1 algorithm leads to a Gaussian distribution of the weights, while the L2 algorithm leads to a Laplace distribution.

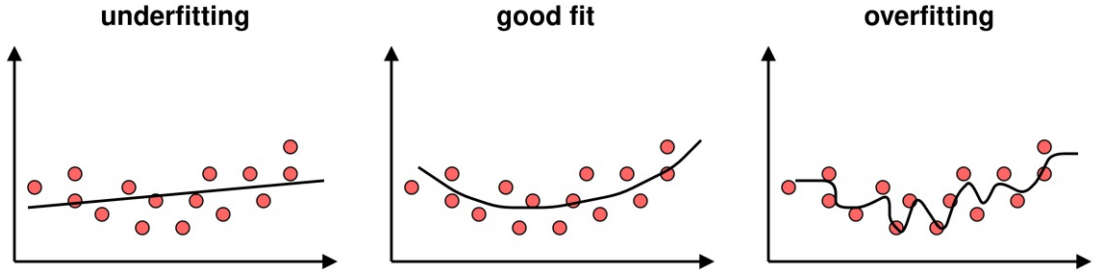
Overfitting can also have its roots in a model structure that is too complex [71] (pp. 28-30). Too many learnable parameters result in the model's inability to generalize simple problems sufficiently, because the network might be able to memorize the input data itself. Furthermore, a model that is too complex might also cause the neural network to detect subtle but irrelevant patterns, which mainly constitute noise. In this case, reducing the number of layers or parameters usually helps to reduce the likelihood of overfitting.

Underfitting defines the second common problem associated with neural networks models [71] (p. 30). It generally occurs when the model is too simple and therefore not able to learn the complex relationships between the input features, resulting in its inability to predict the desired outcome. The main method to avoid underfitting is to build a more complex model. In cases where underfitting is caused by methods to prevent overfitting, removing the respective methods might help to solve the problem. Figure 2.13 illustrates three simple models to visualize the three states of underfitting, overfitting, and a good fit.

2.3.6 Hyperparameter Optimization

Hyperparameter optimization, also called hyperparameter tuning, is a process to find a good configuration for the hyperparameters of a neural network [96]. A hyperparameter is a parameter which's value is freely defined prior to the training process and is therefore not learned during the process. Examples for hyperparameters include the learning rate,

Figure 2.13: Illustration for underfitting, good fit and overfitting, (p. 144) [73]



the number of samples processed before the model is updated known as batch size, the number of iterations over the whole training set called epochs, the number of layers, the types of the layers, the number of neurons, the type of activation function, etc. Each hyperparameter of a neural network has an effect on its performance, influencing for example how fast the network learns and converges towards the optimal solution and if the network is capable of learning the problem in the first place. Especially for advanced problems, hyperparameter optimization is a main contributing factor to build a performing neural network.

There is no easy solution to determine the best settings for these parameters. Therefore, various techniques are based on trial-and-error, resulting in a time-consuming process that may require 80 to 90% of the total development time of a neural network. Basic brute-force search strategies with algorithms like Grid-Search (GS) and Random-Search (RS) represent comparatively simple methods [95] (pp. 198-202). To perform such a search strategy, the search space must be restricted by defining the possible values for each hyperparameter. This step is necessary since every parameter value increases the number of search operations exponentially. The Grid-Search algorithm iterates over every possible parameter combination in a deterministic way to build, train and test all possible neural networks. A Random-Search strategy, on the other hand, randomly selects parameter values from the search space to iterate across all possible parameter combinations. When comparing the two methods, RS proved a superior performance over GS because random selections waste less time iterating over poor configurations [97]. Regardless of the preferred technique, the early-stopping method as introduced in section 2.13 helps to reduce the search time by skipping over bad configurations more quickly.

The process of hyperparameter optimization represents a remarkably active research field that offers solutions much more advanced than the described search algorithms. While describing these advanced techniques goes beyond the scope of this thesis, some examples worth mentioning include the Ant Colony Optimization, Genetic Algorithms, Particle Swarm Optimization, and Simulated Annealing [95] (p. 202).

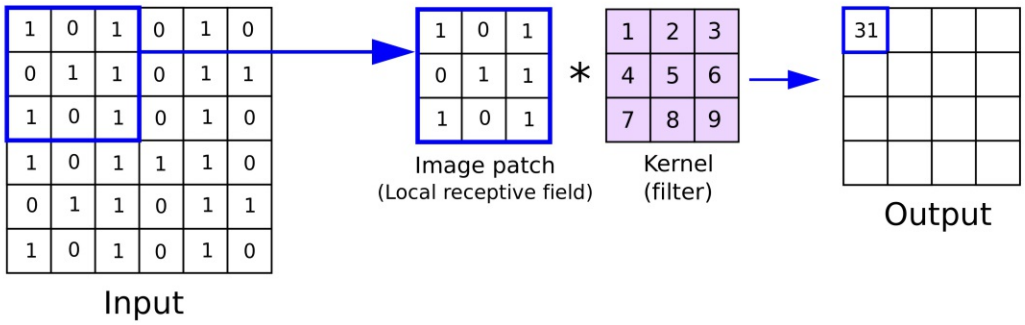
2.3.7 Neural Network Architectures

Next to the diverse functions and parameters, there are also various neural network architectures which are developed for different kinds of input data and use cases. Moreover, these architectures can be combined within one single network to solve more complex problems. This section introduces the architectural approaches of convolutional neural networks, two kinds of recurrent neural networks, as well as multiple transformer networks.

Convolutional Neural Networks

Convolutional neural networks, short CNN, represent a class of neural networks that mimic the human visual system. These networks, which were first introduced by Kunihiko Fukushima in 1980 [98], require vast amounts of data and computational power to operate, which is the main reason that they became popular only within the last decade. The core part of these networks are so-called convolutional layers which apply mathematical convolutions, also known as filters or kernels, to the incoming data to generate feature maps [75] (pp. 162-165). In simplified terms, this process is like downsizing the input data by extracting only its important parts to allow the network to focus on relevant data subsets. Figure 2.14 illustrates this step. The convolutional layer is commonly combined with a pooling layer to further downsample the resulting feature maps, which helps to make the results more robust and less location dependent. Furthermore, there are different types of pooling layers, with max-pooling and average-pooling being the most popular ones [75] (p. 166). Max-pooling simply extracts the maximum value from the input data while average-pooling calculates and returns the average value. The convolution and the pooling layers are usually applied multiple times in parallel or sequentially to extract different parts of the input. In their early days, convolutional neural networks were mainly used for visual tasks like image recognition. In recent years, however, they have been applied to many different areas like NLP and time series forecasting [99].

Figure 2.14: Illustrations of filter application, anhreynolds.com [100]



Recurrent Neural Networks

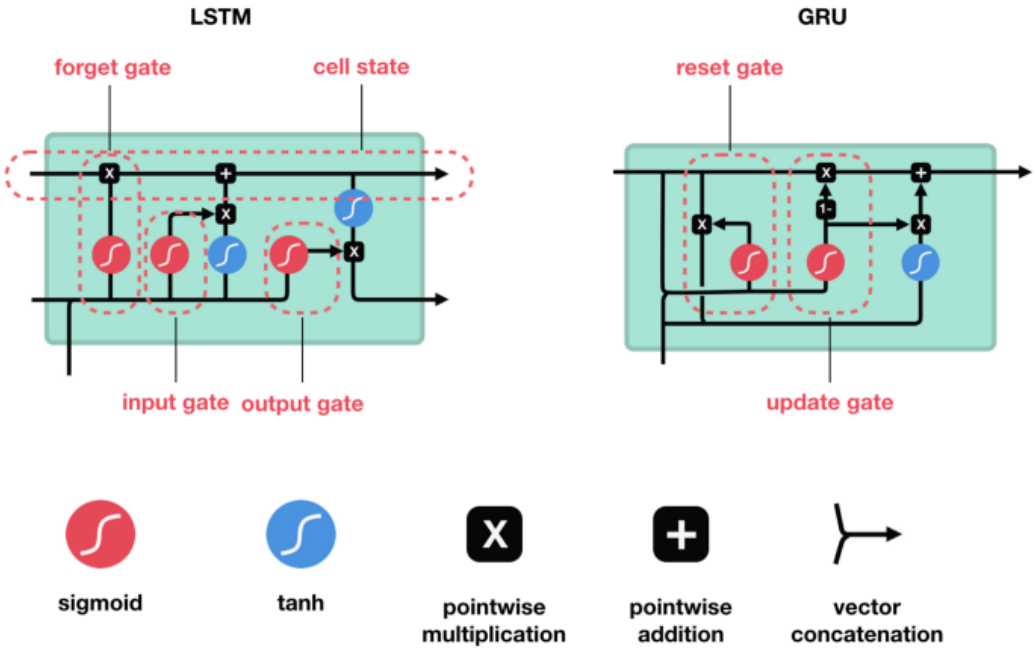
The second class of neural networks is called recurrent neural networks (RNN) and was invented to handle input data with a sequential relationship like time series data, text data, or audio data [75] (pp. 229-234). The unique feature of these networks is that they allow for recurrent connections between neurons. Such a recurrent connection is described as a connection with the neuron itself, with another neuron on the same layer, or with a neuron on a previous layer. The resulting advantage is that it allows the network to use the knowledge from previous inputs while predicting the next one. The main downside associated with basic RNNs, on the other hand, is their inability to recall old data which worsens as the time progresses. Simply put, the older the data, the lower the likelihood that it is memorized. The so-called vanishing gradient problem, which was researched by Hochreiter in 1991 and Bengio, et al. in 1994 [80], causes the networks to slowly forget processed input data. The gradient shrinks as it back propagates through time, continuously becoming smaller until it cannot contribute to the learning process anymore.

Aside from advantages and disadvantages in their basic form, RNNs can be of different architectural structures with the most common ones being called long short-term memory (LSTM) and gated recurrent unit (GRU) [101][102]. Both architectural forms were explicitly designed to solve the vanishing gradient problem by including a long-term memory mechanism. The concept of LSTM was introduced in 1997 by Hochreiter & Schmidhuber [101] while GRU was developed more recently in 2014 by Chung, Hyun & Bengio [102]. Both architectures use mechanisms called gates inside their recurrent cell logic. These gates are simple neural networks that decide which information is stored and which information is dismissed while the data is passed down the recurrent connection chain. LSTM cells use the three different types input-, output-, and forget gate while the new GRU architecture only requires a reset- and an update gate. Figure 2.15 displays a comparison between the inner logic of a LSTM and a GRU cell. Generally, a GRU cell is easier to calculate due to the simpler design and therefore allows faster training times. LSTM, on the other hand, might be able to learn more complex problems according to a hypothesis. In conclusion, it is usually best to test both architectures and pick the one that works better for the specific case.

Transformer Neural Networks

The last type of architecture described in this thesis is called transformer networks [104]. These networks use a technique known as attention to process sequential input data instead of the previously introduced recurrent connections. Attention is a relatively new concept of neural network which tries to mimic the cognitive attention of a human brain. In other words, with an attention mechanism the network can focus on the important parts of the input data and learn them during the training process, while ignoring irrelevant parts. The concept was first introduced in the paper *Attention Is All You Need* in 2017 by the Google research team [104]. Nowadays, attention-based models are the best performing models in the NLP space and are also getting adapted for other areas like visual computing.

Figure 2.15: Inner logic of LSTM & GRU cell, towardsdatascience.com [103]



The main advantage that these models share is that unlike with RNN models, the input data does not have to be processed in sequence [104]. This allows for a better parallelization of the training process which results in significantly reduced training times. Consequently, leading AI companies were able to pre-train big general purpose language representation models with enormous amounts of unannotated text data from all over the internet in their massive data centers. These models can later be fine-tuned for specific tasks with smaller datasets, requiring only a fraction of the computational resources compared to the training of the whole model. Today, there are several such pre-trained general-purpose models. A selection of the most famous ones is presented below:

- **BERT** (*Bidirectional Encoder Representations from Transformers*) was the first successful pre-trained transformer model which opened the gates for others to follow. Developed and published by the Google AI Language team in 2018 [105], it revolutionized the entire NLP section for machine learning with its outstanding performance. As the name suggests, the innovative model is bidirectionally trained which allows it to have a much deeper sense of the text flow and context compared to traditional single-direction models. More specifically, the BERT model uses the masked language modelling (MLM) as well as the next sentence prediction (NSP) strategy during the training process. MLM randomly masks words of a sentence, covering them from being read. Afterwards, the network attempts to predict these words based on the surrounding context. For the NSP strategy, the model processes

pairs of sentences during the training phase where the second sentence is random and unrelated to the first one in 50% of the cases. The neural network is afterwards required to predict this second sentences' origin to be related or not.

- The second major model, **Roberta** (*A Robustly Optimized BERT Pretraining Approach*), represents an improved version of the BERT model that was developed by Facebook in 2019 [106]. The model is based on an updated version of the hyperparameters to improve the MLM strategy of the network while disregarding the NSP strategy. Additionally, the roberta model uses an extended input text corpus compared to BERT by including the CC-News [107] dataset consisting of 63 million news articles, the OpenWebText dataset [108], as well as the stories dataset which contains petabytes of text data from all over the internet.
- **BART** (*Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*) is yet another model developed by the Facebook AI Research team in the year 2019. The model uses a seq2seq architecture combined with a bidirectional encoder and a left-to-right decoder for the training process. During the pre-training process the model attempts to reconstruct the input data which was corrupted by replacing spans of text with a single masking token and by randomly shuffling the order of the sentences. The details about the model architecture exceed the scope of this thesis but can be found in a paper by Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer [109].
- The **XLNet** model (*Generalized Autoregressive Pretraining for Language Understanding*) uses a general autoregressive learning strategy to solve a pretrain-finetune discrepancy which occurs in the BERT model. The xlnet paper [110] published by Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le in 2020 offers a detailed explanation for this learning strategy.
- **T5** (*Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*) is the latest model which was developed by the Google Research Team in 2020 [111]. It was trained with the new dataset Colossal Clean Crawled Corpus (C4) which represents a cleaned version of the CommonCrawl dataset. T5 is a shared text-to-text framework, meaning that every NLP task can be represented with text as an input as well as an output. This allows for the usage of the same code, hyperparameters, and loss function for the finetuning of any NLP task like text summarization, question answering, sentiment analysis, and more.

2.4 Related Work

In the last part of this chapter, we present further papers with a similar research context or methodology. Topics include neural network solutions that incorporate technical indicator data to predict the stock market as well as different methods to analyze the sentiment of news data.

In the paper published by Chenjie Sang and Massimo Di Pierro in 2019, *Improving trading technical analysis with TensorFlow Long Short-Term Memory (LSTM) Neural Network* [112], technical indicators with LSTM-based neural networks are used to predict the stock market. To build the neural networks for their research, the authors used three of the most common technical indicators SMA, RSI, and MACD with their default configurations as input data. Furthermore, they defined the target value as the difference between the last two price ticks. To measure the performance of the network they compared their predictions with the default strategies of the used technical indicators. The authors measured a superior performance of their neural network compared to all default strategies and concluded that technical indicators in combination with neural networks are able to outperform most standard trading strategies with only minor configurations.

Omer Berat Sezera and Ahmet Murat Ozbayoglu combined a CNN with technical indicators to predict the stock market in their 2018 published paper *Algorithmic Financial Trading with Deep Convolutional Neural Networks* [113]. The authors take an entirely new approach by converting technical indicator data to chart images and feeding these images to the CNN for their predictions. The network uses Buy, Sell, and Hold signals as target values, which were labeled based on the top and bottom of an 11-day sliding window. The images were generated for 15 different technical indicators as well as multiple configurations for each indicator. The results of the CNN were compared to a Buy & Hold Strategy, a simple LSTM network, a MLP network as well as the default strategies for technical indicators with the data of multiple stocks and ETFs. The CNN managed to consistently outperform all mentioned reference strategies which can be considered an impressive achievement for such a unique approach.

In the third reference paper *Deep learning for stock market prediction from financial news articles* it is described how an analysis of financial news articles using technical indicators can predict the stock market [114]. The paper was researched and published by Manuel R. Vargas, Beatriz S. L. P. de Lima, and Alexandre G. Evsukoff in 2017. The authors used the headlines on financial news and fed them into a multi-layer network consisting of a word-embedding layer, a CNN layer as well as an LSTM layer. Furthermore, the data of seven popular technical indicators was processed in a separate LSTM network. The outputs of the two networks were combined with a fully connected layer to predict the movement of the stock market. The focus of this paper was to compare different text representation methods for the task of sentiment analysis. Therefore, the authors' compared their network to ten other prediction models from different referenced literature. With their experiment, the author confirmed that the technical indicator data had a positive impact on the prediction models and further concluded that event embedding outperforms sentence- and word embedding for the task of financial sentiment analysis.

In the paper *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models* written and published in 2019 [115], the author Dogu Tan Araci tried to solve the task of financial text predictions with a pre-trained transformer model. Tan Araci pre-trained the default BERT model with the TRC2- financial dataset, a corpus of 1.8 million financial news articles, and finetuned the model for the task of financial sentiment

analysis. For the finetuning process, the datasets Financial PhraseBank, consisting of 4845 sentiment-labeled financial news articles as well as FiQA Sentiment, consisting of 1174 sentiment-labeled financial news headlines and microblogging messages, were used. The model was compared to an LSTM classifier with GLoVe embeddings, an LSTM classifier with ELMo embeddings, and an ULMFit classifier. Summing up the results, the FinBERT model managed to outperform all reference models by at least 5% for the F1 score as well as 3% for the accuracy.

The papers presented above provide a general overview about techniques that are currently applied by other researchers to incorporate data from technical indicators as well as financial text data into their market predictions systems. Thereby, this summary highlights the broad spectrum of methods used for the integration of technical indicator data. Each analyzed paper applied different techniques, indicator settings, and machine learning models. One common similarity identified among most analyzed papers was the usage of LSTM and CNN architectures for the market prediction system. However, none of the papers tried to combine these two network architectures for the task of market predictions. Furthermore, only a few papers invested time into the optimization of hyperparameters of the neural networks and most of them neglected to discuss the generation of target values.

In the following chapters of this thesis, we describe the construction of a more complex neural network structure achieved by incorporating both LSTM and CNN layers. Moreover, we spent a significant time on the optimization process for the networks. Additionally, we set a unique focus on target values since we believe they are among the most important factors to build a successful market prediction system. The current research landscape on the usage of sentiment transformer models for the market prediction is scarce. Therefore, we also incorporated multiple transformer models into our prediction system and tested them to generate novel outcomes and close the identified research gap.

CHAPTER 3

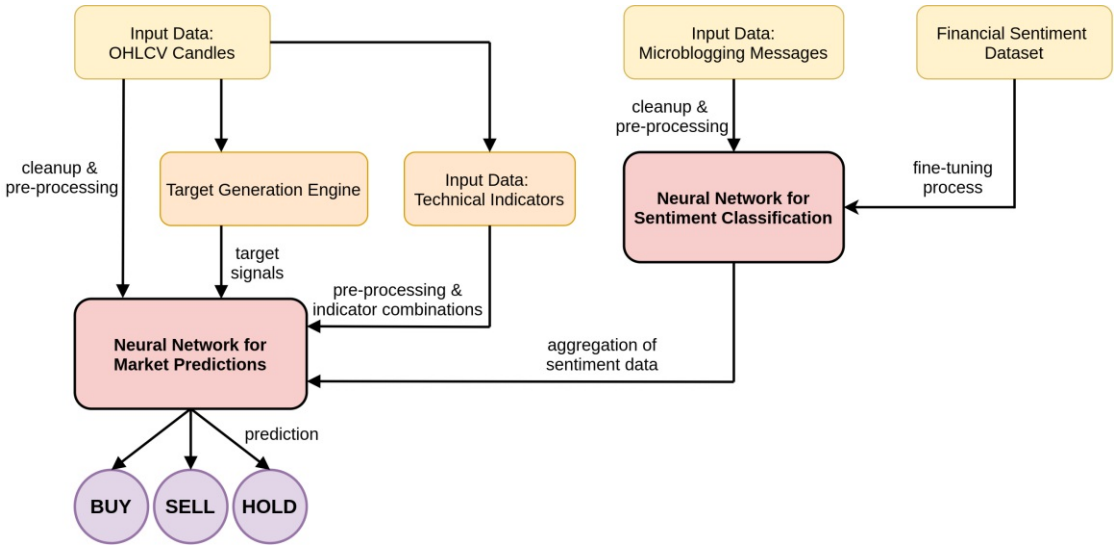
Design

This chapter outlines the methodological approach which was used to create this thesis' practical part. The first section gives a general overview of the developed system as well as its goals. Afterwards an in-depth description of the parts for the market prediction neural networks is provided. The following section 3.3 outlines the developed sentiment classification part in detail. Ultimately, the last section 3.4 describes the search & selection algorithm, also known as optimization algorithm which was used to identify the best parameter settings for the different parts of our designed system.

3.1 Overview & Goals

The main goal pursued through this thesis is to develop a neural network-based system which is able to generate trading signals that result in a profitable trading strategy. On this basis, we focus on a generic system design to allow the prediction of any asset class like stocks, crypto, and forex on the one hand and to permit the usage of an arbitrary number of time-based input sources on the other hand. For the analysis, we use OHLCV candles of the bitcoin cryptocurrency as well as bitcoin specific microblogging messages from twitter [116] and stocktwits [117] as input sources. Both sources are further processed – the OHLCV candles are used to generate technical indicators as described in 3.2.3 while the microblogging messages are converted into specific sentiment classes through the sentiment classification system outlined in 3.3. Furthermore, the closing price of the OHLCV candles is used for the target generation of the market prediction neural network. For the evaluation process of our main goal we developed a simulation engine which allows to visualize the generated trading signals and to calculate different trading metrics. Furthermore, it provides options for risk adjustments of the trading strategy. Concluding, figure 3.1 displays a simplified overview of how the system's various components are interconnected.

Figure 3.1: System Overview



3.2 Neural Network for Market Predictions

This section is dedicated to the different components that our neural networks for market predictions require. The initial part describes the OHLCV input data and provides information on how this data was cleaned and preprocessed. Next, the target generation engine which is used to generate training labels for the neural network is presented. The following section 3.2.3 describes the technical indicator input data as well as how this data was preprocessed and combined during the training process. Afterwards the neural network structure is discussed and parameters to modify the structure are presented. The last part focuses on the simulation engine which was used to test and evaluate the trained neural networks.

3.2.1 Input Data: OHLCV Candles

OHLCV represents an aggregated data format for the historic trading data of an underlying asset. This data format carries essential advantages, which are described in section 2.2.2. Summing up the most important ones, it provides insights about the traded volume as well as the highest and lowest trading price for a specified trading interval on the one hand. Furthermore, OHLCV data is the required data format for the calculations of technical indicators. For these reasons, we decided to use OHLCV candles as main data source for our market prediction neural networks.

As mentioned in section 2.3.2, neural networks require a huge amount of data for the training process to be able to identify useful patterns. Therefore, we decided to use the whole date-range, that was available at the time the system was developed, from the

Binance Exchange as OHLCV input data. This range starts at August 2017 and ends in November 2020. Another important factor for OHLCV data is the aggregation time unit also known as interval. A short interval between one minute and one hour is usually used for short-term predictions while a longer interval primarily suits for predictions regarding the long term. The value for this interval parameter was identified with the search & selection algorithm described in 3.4.

We applied both a process for data cleanup and data preprocessing, which represent crucial steps to design a well-performing neural network model as described in 2.3.2. The cleanup step included the generation of missing OHLCV candles with a linear interpolation method. The resulting cleaned input data could theoretically be used directly as both an input source and a regression target for a neural network which is exemplified by several researchers [118] [119] [119] [120]. This approach, however, leads to major overfitting problems in most cases.

Moreover, price data points follow a random walk between zero and $+\infty$. Therefore, networks are usually unable to find significant patterns within the price data since all trends look different. Given the resulting lack of predictive insights, the network uses the latest price point as its forecast since this data point is eminently close to the actual next one. Consequently, by learning a lagging price the network achieves the highest accuracy.

While some researchers state that they achieved accuracies of >90% for the price prediction regression problems with this approach, closer examination reveals that these accuracies are misleading and that the trained networks don't provide any real value.

To solve the problem associated with a lagging price prediction, we introduced a preprocessing step to transform the price data into a relative data format. This allows the neural networks to find patterns within price changes without requiring the asset price itself. Therefore, we calculated the percentage change between each two consecutive data points with the following formula:

$$pct\ change_x = \frac{p_x - p_{x-1}}{p_{x-1}} \tag{3.1}$$

where:
p_x is the current price
p_{x-1} is the previous price
$pct\ change_x$ is the percentage change between x-1 and x

Furthermore, as mentioned in 2.3.2, unscaled input data can result in a slow and unstable learning process for neural networks. Therefore, we decided to use an additional preprocessing method called the robust scaling technique [121]. This technique owes its name to the fact that it is robust to outliers which is achieved by scaling every data point to the interquartile range, which represents the range between the data's first and third quantile. We chose the robust scaling technique after discovering multiple bigger

outliers within our transformed OHLCV data. In order to implement the technique, the following formula is applied to each data point:

$$x_{robust} = \frac{x - Q_1(x)}{Q_3(x) - Q_1(x)} \tag{3.2}$$

3.2.2 Target Generation Engine

The next essential part of a good performing neural network includes the target values, also known as labels. As mentioned in 2.3.2 these values are used in supervised machine learning problems to define the solutions for the training data. The neural network requires these definitions to figure out what it is supposed to learn as well as to evaluate its performance. In 2.4 we discovered that most researchers apply surprisingly basic methods to generate these values without critically evaluating them. In contrast, we are convinced of their importance for generating a successful trading strategy and therefore included a separate target generation engine in our system design which has the sole purpose of producing adequate target values.

There are multiple approaches on how these target values can be defined, each carrying its own advantages and disadvantages. Aumayr, for example, discovered in his thesis [9] that a regression target significantly underperforms compared to a classification target for the task of bitcoin price predictions. Consequently, we decided to define a classification problem and chose the target values of BUY, SELL, and HOLD with the following meaning:

- **HOLD** signal means that any currently open position shall not be changed.
- A **BUY** signal will close any open short position and open a new long position. If there exists a long position already, then this signal is equivalent to a HOLD signal.
- A **SELL** signal will close any open long position and open a new short position if short-trading was enabled within the simulation engine 3.2.5. If there exists a short position already, then this signal is equivalent to a HOLD signal.

These targets provide a clear direction after each timestamp and can furthermore be easily implemented within a trading simulation. To generate the three target classes, the simplest solution can be defined with the following three equations:

- BUY: $p_x > p_{x-1}$
- SELL: $p_x < p_{x-1}$
- HOLD: $p_x == p_{x-1}$

where:
p_x is the current price
p_{x-1} is the price of the previous tick

This corresponds to the preferred solution used by most researchers for the development of neural networks for market predictions [122] [123]. The main problem associated with this simple solution, however, is the huge amount of BUY and SELL signals it generates since every modest price movement results in a new signal. This leads to a vast number of unnecessary and faulty trading decisions which decimate profits due to trading fees that occur for every trade. Therefore, we decided to add two threshold parameters in our target generation algorithm which allow us to regulate the number of generated BUY and SELL signals as well as to adjust the general risk level associated with them.

The first of these thresholds is called *minimum profit target*. It defines the minimum price change in percent that must be reached by a future price to generate a BUY or SELL signal. A *minimum profit target* of 10%, for example, means that the algorithm scans prices of the next timestamps and creates a BUY or SELL signal once a difference of 10% compared to the current timestamp is detected. This search process stops after a price with the specified difference was identified or when the end of the data is reached. Thereby, the minimum profit target threshold allows us to ignore small price movements while putting a focus on more significant changes. Consequently a higher *minimum profit target* will result in fewer BUY and SELL signals which leads to fewer trades.

The second threshold is named *maximum accepted loss* and describes the maximum allowed drawdown of the current price before a signal generated by the *minimum profit target* is invalidated. For example, a *maximum accepted loss* of 5% will convert a BUY signal into a HOLD signal when the price from the current timestamp drops by five or more percent between the current timestamp and the one which was identified by the *minimum profit target* search process. This allows us to manage the risk of the trading strategy by reducing the number of trades where a significant drawdown would occur.

Both threshold parameters are illustrated in figure 3.2. Furthermore, the pseudo code 3.1 demonstrates how the signal generation algorithm works in detail.

Listing 3.1: Signal Generation Algorithm

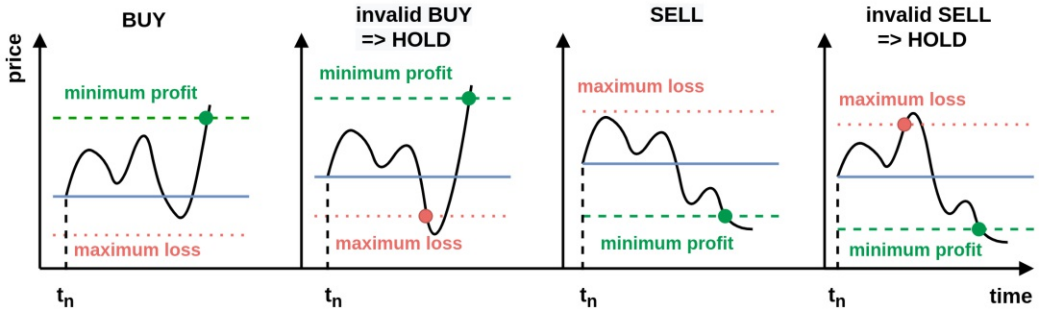
```
min_profit_up = 1 + (config.minimum_profit_target / 100)
min_profit_down = 1 - (config.minimum_profit_target / 100)
max_loss_up = 1 + (config.maximum_accepted_loss / 100)
max_loss_down = 1 - (config.maximum_accepted_loss / 100)

def generate_target(search_start_index):

    decision = HOLD

    current_value = price_data[i]
```

Figure 3.2: Illustration of threshold parameters for target generation




```
best_value = current_value
worst_value = current_value

buy_threshold = current_value * min_profit_up
sell_threshold = current_value * min_profit_down

buy_block_threshold = current_value * max_loss_down
sell_block_threshold = current_value * max_loss_up

buy_blocked = False
sell_blocked = False

for j in range(search_start_index, len(self.data)):

    future_value = price_data[j]

    if best_value < future_value:
        best_value = future_value

    if worst_value > future_value:
        worst_value = future_value

    if buy_blocked is False\
        and future_value < buy_block_threshold:
        buy_blocked = True

    if sell_blocked is False\
        and future_value > sell_block_threshold:
        sell_blocked = True

    if buy_blocked and sell_blocked:
        decision = HOLD
        break
```

```
            if buy_blocked is False and future_value > buy_threshold:
                decision = BUY
                break

            if sell_blocked is False and future_value < sell_threshold:
                decision = SELL
                break

            if j == len(self.data) - 1:
                decision = HOLD

        return decision
```

While not featuring high complexity, this algorithm is more advanced than the most common approach by other researchers which represents a simple comparison with the subsequent price tick as described in 2.4.

3.2.3 Input Data: Technical Indicators

Another source of input data described in 2.2.3 is derived through technical indicators, which are clearly defined mathematical calculations that are applied to the historic trading data of an asset. As mentioned before, one main focus of this thesis is to evaluate the predictive capabilities of multiple different technical indicators. To test this objective, we calculated 62 different technical indicators from the indicator groups described in 2.2.3. Afterwards, up to three of these indicator results were randomly combined during the training process of a single neural network to identify potential synergies. Considering the enormous number of resulting trained neural networks, only the default parameter configuration was evaluated for every indicator as defined by the talib library [60]. This allowed us to stay within the limitations set by our available computing capacities. A list of all selected indicators and their used configuration can be found in the evaluation chapter 5.3.2.

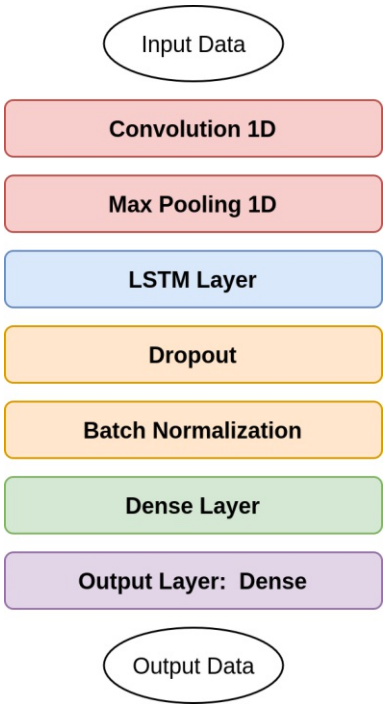
As described in 2.3.2, the preprocessing step is crucial for any input feature. We applied different preprocessing techniques depending on the output of the indicator. Indicators that use the same scale as the asset price and that are furthermore designed for direct comparison to that price, for example, require special attention. This is due to the fact that the network, by contrast, only receives a transformed version of the asset price and is therefore not able to compare these values directly. To preprocess these indicators we calculated the difference between the asset price and the indicator value and normalized the results afterwards with the robust scaling technique mentioned in 3.2.1. This procedure allows the network to interpret the data in the same way as human traders would. Indicators that are bound between fixed values like [0, 100] or [-100, +100], on the other hand, only require scaling as their preprocessing method. Indicators with unbound values that do not use the assets price scale, lastly, are preprocessed similar to the OHLCV data. Therefore, the percentage change is calculated followed by a scaling of the values. Regardless of the specific applied technique, a successfully executed

preprocessing procedure allows for the results to be fed directly into the neural network as an additional input source.

3.2.4 Neural Network Structure

Further vital parts for a good performing neural network model are its structure and the hyperparameters it uses. As mentioned in 2.3.7, for every problem there are countless neural network structures to potentially solve it. This, on the other hand, results in the necessary process to find a good one, which is oftentimes done through a simple trial and error approach. To skip much of the trial-and-error phase, we decided to build on existing knowledge and use the best performing model from Aumayr [9] as our base model. The structure of this model is illustrated in figure 3.3. In order to identify a well-performing structure for our specific use case of market predictions, we then modified the base model utilizing our optimization algorithm described in 3.4.

Figure 3.3: Neural Network: Base Model Structure



More precisely, our adjustments to the base model allowed us to freely modify 13 inner parts of the neural network with external parameters. Seven of these parameters were used to directly modify the neural network structure as explained in more detail below:

- Parameter to disable the CNN layer.
- Parameter to adjust the filter size of the convolution layer.

- Parameter to switch between two different RNN layers: LSTM and GRU.
- Parameter to adjust the size of the RNN layer.
- Parameter to disable the Dense layer.
- Parameter to adjust the size of the Dense layer.
- Parameter to change the activation function of the Dense layer.

While developing the neural network we encountered multiple overfitting problems, which commonly happens as described in 2.13. Therefore, we added multiple dropout layers between the most important hidden layers of the neural network like the CNN, RNN, and Dense layer. Additionally, we implemented an early stopping technique to stop the training process once the validation loss no longer showed improvements over multiple iterations. On top of that, three of the mentioned parameters were used to configure regularization methods against overfitting as listed below:

- L1 & L2 parameters for the RNN layer.
- A parameter to adjust the dropout size of all dropout layers.

The last three parameters were used to adjust the learning process of the neural networks. More specifically, the first one was used to adapt the optimizer function itself, the second one to modify its learning rate, and the last one to adjust its weight decay.

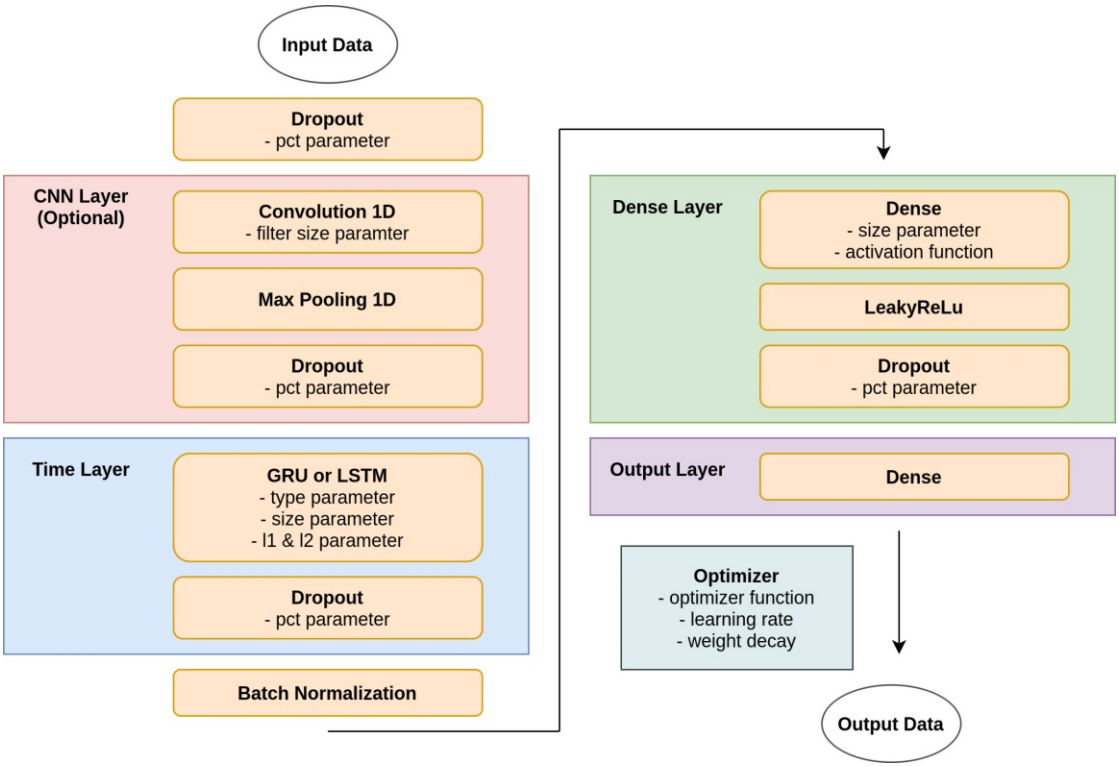
The final neural network structure including all modifications and configurable parameters is illustrated by figure 3.4.

3.2.5 Simulation Engine

The last part of the system for market predictions is a simulation engine to evaluate and compare all of the trained neural networks. On the one hand, the developed engine supports different settings to adjust the risk level of the trading strategy. These settings include the position size, which can be set in absolute or in relative terms, as well as the stop loss and trailing stop loss functions. On the other hand, the engine includes an option for short trading, which was enabled by default for most of the evaluated trading strategies, as well as an opportunity to specify a trading fee. This fee was set to 0.2% based on the Binance exchange. The various simulation settings were only adjusted in the final evaluation step of the thesis which can be found in the evaluation chapter at 5.4.2.

In general, the simulation engine calculates multiple trading statistics to compare the different strategies. These include the ROI, the sharpe ratio, the sortino ratio, the maximum drawdown, as well as the number of trades, where it specifies between total,

Figure 3.4: Neural Network Structure



positive, and negative ones. Additionally, the engine prints a performance chart for a visual comparison between the buy-and-hold strategy and the simulated strategy. This chart can optionally display the entry and exit timestamps for each trade.

3.3 Financial Sentiment Analysis

Another core part of our research concerns the effect of microblogging sentiment data on the performance of the previously generated trading strategies. This section provides detailed insights into the development of the underlying analysis process for the case of financial microblogging data. The first part is dedicated to the neural networks that were used as well as their training process. In the second part, the necessary microblogging data that was processed by the developed sentiment networks are outlined.

3.3.1 Neural Networks for Sentiment Classification

In our pursuit of developing a neural network classification system, different architectures were considered to identify the best solution for the task at hand. As previously mentioned in 2.3.7, an unparalleled performance for NLP tasks was found to be achieved with a

transformer architecture. Therefore, we decided to build, fine-tune and evaluate the leading pretrained transformer models: *BERT*, *Roberta*, *BART*, *XLNet*, and *T5*. A detailed description for most of them can be found in chapter 2.3.7. Moreover, an extended version of the Roberta model – the *Roberta-Twitter* model – was additionally evaluated, which was further pretrained with a dataset consisting of 58 million tweets [124].

The most important aspect of transformer models is the fine-tuning process which allows us to train the neural network for a very specific NLP task. In this case, the task is defined as a sentiment classification for financial microblogging messages with the three output classes positive, neutral, and negative. Furthermore, the process requires labeled input data which explicitly solves the task or a very similar one. One main advantage of pretrained transformer models is that the fine-tuning process requires only a small dataset compared to the vast amounts of data necessary for training processes of regular neural networks [105]. Another factor which further increases the relevance of these models is the fact that publicly available, labeled sentiment datasets in the space of finance exist only in an eminently limited amount.

The specific financial sentiment datasets which were used for the fine-tuning process are described in the following list:

- **Finbank-50**
  Finbank-50 is based on the *FinancialPhraseBank-v1.0* [125] dataset, which contains 4840 sentences extracted from financial news articles and their corresponding sentiment class. The sentiment was determined by 16 financial experts from the *Aalto University School of Business*.

- **Finbank-balanced**
  Finbank-balanced is a modified version of the *finbank-50* dataset and contains the same amount of sentences for each sentiment class. We sorted the sentences based on their expert agreement-level and selected the top 600 for each class.

- **Semeval-headlines**
  Semeval-headlines is a combination of three different datasets. It contains the news headlines from the *SemEval-2017 Task 5* project website [126], the original SemEval-2017 Task 5 dataset provided by the developers and the FiQA-2018 Task 1 dataset [127]. All three datasets contain financial news headlines and were classified by three independent financial experts.

- **Semeval-microblog**
  Semeval-microblog has the same three origins as the *Semeval-headlines* dataset. It contains financial microblogging messages from twitter and stocktwits and was classified by the same three independent financial experts as the *Semeval-headlines* dataset.

- **Stocktwits**
  Stocktwits was constructed from our downloaded stocktwits data 3.3.2. It contains the 1500 most liked messages with a *bullish* tag for the positive sentiment class and the 1500 most liked messages with a *bearish* tag for the negative one.

After the fine-tuning process, we utilized the search and selection algorithm from section 3.4 once again to identify the best performing transformer architecture for our financial sentiment classification task. The same algorithm was used to optimize the neural networks hyperparameters.

3.3.2 Input Data: Microblogging Messages

After determining the best model, new and suitable text data were required as an input to derive meaning and potentially increase the performance of the trading strategies. On the one hand, we decided to use microblogging data from the social network Twitter given its tremendous popularity among the cryptocurrency community. Additionally, data from stockwits was utilized, which represents a comparable microblogging platform that focuses on the investment space. Stockwits has the additional benefit of complementary information about its users trading behavior. Overall, both platforms are built around public messages with a limited number of words and an extensive use of hashtags to specify the topic of the messages.

To download the historic messages of the two platforms we then developed two different scraper tools. For Twitter, messages with the hashtags *#bitcoin* and *#btc* were scraped. Similarly, we focused on messages marked with the bitcoin trading symbol $BTC.X$ for the stocktwits platform. Concerning the range of dates, we used the same one as for the OHLCV data: August 2017 to November 2020. The downloaded data was afterwards cleaned by removing messages that potentially contain noise, do not add meaning, or cannot be analyzed as described below:

- removal of non-English messages
- removal of duplicate messages
- removal of short messages
- removal of messages which mainly consist of images, links or hashtags
- removal of messages from users with too few followers

Lastly, the cleaned messages were fed into the previously selected transformer model to classify their sentiment.

The generated and classified sentiment values were then aggregated based on the chosen OHLCV data interval from section 3.4, which is necessary to match the length of each input feature for our market prediction neural network.

3.4 Search & Selection Algorithm (Optimization Algorithm)

The system design we constructed contains multiple customizable parameters, each of which affects certain parts of the generated trading strategy like the accepted risk, the performance, the likelihood to find a successful strategy, or the accuracy of the sentiment predictions. Therefore, to achieve the goal of generating a profitable trading strategy, it was necessary to design a generic parameter optimization algorithm to identify suitable values for each parameter. Below, all parameters of the market prediction neural network are described chronologically in the same order they were optimized in. Furthermore, the different optimization foci are outlined in short.

1. **OHLCV input data: Interval**
   The first parameter we optimized is the interval of our OHLCV input data. This parameter defines the aggregation level of our primary input source and was optimized with the goal of identifying the most profitable value for short-term predictions.

2. **Target Generation Engine: minimum profit target, maximum accepted loss**
   As described in section 3.2.2, *minimum profit target* and *maximum accepted loss* can be used to adjust the taken risk of the learned trading strategy as well as to regulate the number of entered trades. We optimized these parameters with the goal of a balanced profit-to-risk ratio.

3. **Neural network basics: batch size, sequence length**
   Furthermore, we optimized the two basic neural network parameters *sequence length* and *batch size*. The sequence length specifies the number of timestamps that are fed to the network as a single training example while the batch size specifies the number of training examples which are used during a single iteration of the training process. Both parameters were optimized based on the most profitable trading performance.

4. **Neural network structure: 13 customizable parameters**
   Additionally, we applied the optimization algorithm to improve all thirteen parameters mentioned in 3.2.4 which affect the inner structure as well as the hyperparameters of the neural network. These parameters were again optimized for the most profitable trading strategy.

5. **Sentiment Neural Networks: Model & Hyperparameters**
   Second to last, we used the optimization algorithm to find the most suitable pretrained transformer model for the sentiment classification of financial microblogging data. Additionally, the algorithm was used to select the hyperparameters of the chosen transformer model.

6. **Indicator input: selection of indicators**
   After identifying suitable parameter values for every part of the system, we used the optimization algorithm to find indicators as well as indicator combinations which lead to the trading strategies with the highest sortino ratios.

The optimization algorithm we developed consists of three main parts:

1. The search process which iterates over all possible parameter values for the selected parameter group from the list above.
2. The evaluation process which trains the network and generates analytical information.
3. The selection process which selects the best performing parameter values based on the evaluation results.

For the search process we used a random search technique to iterate over all possible parameter combinations, since it proved to be more efficient than a simple grid search operation as elaborated in 2.3.6. Furthermore, it was important to keep the search space for each parameter as small as possible, since any additional value increases the required evaluation time exponentially.

During the following evaluation step of the algorithm, the parameter values from the current search iteration are fed into the system. The system then self-adjusts its inner parts based on the provided configurations, which can include actions such as downloading new OHLCV data if the interval changes, generating new target values if the parameters of the target generation engine change, or adapting the neural network if any of its parameters change. Afterwards, a neural network is trained, and the resulting weights are stored in a file. Moreover, the performance of the network, the simulated performance of the resulting trading strategy as well as the current configuration are logged to a separate file.

Furthermore, it is important to note that both the search and the evaluation process of the algorithm run in endless loops until they are stopped. This is crucial since only a large amount of trained neural networks provide the necessary information to evaluate the performance distribution of each parameter value which derives its real performance impact. Additionally, this helps to reduce the effect of outliers since every trained neural network is unique and even with a good parameter combination it might not find a good trading strategy.

After multiple neural networks were trained for each parameter value, the search & evaluation process was stopped to initiate the selection process. During this step, the value distributions for each parameter were compared between all trained networks and the trained networks which outperformed a buy-and-hold strategy. On the basis of this comparison, we selected the best values for each parameter. For some parameter groups

like the neural network structure and the indicator inputs, multiple search and selection loops were performed to test different sets of parameter values and consequently reduce the search space over time. A detailed reasoning for each parameter selection is provided in the evaluation chapter.

With the optimization process finished, we started the implementation of the system as described in the next chapter.

CHAPTER 4

Implementation

This chapter describes the implementation details of the system design introduced in chapter 3. The first section presents the different technologies and languages that were used during the development process. The second part provides a detailed description for each part of the developed system. The last part gives a short overview over different optimization procedures which were applied during the development process as well as the corresponding hardware that was utilized.

4.1 Technologies & Languages

Regarding the programming language for the designed system, we decided to mainly use Python. Python is defined as a dynamically typed, high-level general-purpose language [128] that is particularly popular within the scientific community. It is often used for mathematical, statistical as well as machine learning tasks. One important advantage that Python has over other languages is the vast number of libraries which support the development of the mentioned tasks and especially neural networks. The following list provides an overview of the machine learning libraries which were used to develop the market prediction neural networks as well as the sentiment models:

- **Tensorflow** represents the most well-known library for machine learning. It is developed and maintained by the Google Brain Team [129] and has a broad developer community to find support when problems are encountered. Furthermore, Tensorflow is an opensource project that is mainly used for the development of neural networks. Therefore, we utilized the library to develop our neural networks for market prediction.
- **PyTorch-Lightning** is a relative new library with it's first release in 2019. It is an extension of the PyTorch library [130]. PyTorch itself is the second biggest machine

learning library and is developed and maintained by the Facebook AI Research Lab [131]. PyTorch Lighting provides access to a lot of helper functions which makes the development of neural networks faster and easier and helps to avoid boilerplate code. We used this library for the fine-tuning process of the sentiment neural networks.

- **Huggingface** is a machine learning library which focuses on NLP tasks and provides access to a range of pretrained transformer models [132]. On the one hand, we used these pre-trained models for our sentiment networks. Furthermore, we utilized the functions of the library to prepare the input data for the fine-tuning process.

- **Wandb** is a library for the *Weights & Biases* [133] web service which provides analysis tools for machine learning models. We used the library to log and archive all training results as well as to implement some parts of the search & selection algorithm.

Aside from the machine learning libraries, the developed system uses a number of additional libraries, which are listed and briefly explained below:

- **Matplotlib** is the default plotting library for Python [134] and was used to visualize the trading strategies.

- **Pandas** is a library for data management, data manipulation, and data analysis [135]. It was used to clean and preprocess the input data.

- **NumPy** is a library for advanced numeric operations as well as array and matrix operations [136]. We used it for sorting, randomizing, and caching the input data.

- **scikit-learn** is a popular machine learning library which provides access to many machine learning algorithms as well as preprocessing methods [137]. We applied it to calculate certain metrics as well as to scale and normalize the input data.

- **TA-lib** provides methods for the calculation of various technical indicators [60]. We used it precisely for this purpose.

- **Snscrape** is a scraping library for social networks [138] and was utilized to download historic tweets for the sentiment analysis.

- **requests** is a simple http library [139]. It was used to build a message scraper for the stocktwits website as well as to download the OHLCV data.

- **Joblib** is a library which provides tools for parallel programming [140]. We used it to implement runtime optimizations of the target generation engine.

Next to Python, we developed the system using Bash shell scripts [141] in combination with Unix command line tools [142] and the Perl programming language [143]. These scripts were mainly used for the first cleanup and preprocessing steps of the microblogging messages since these messages were saved to multiple files during the download process and required about 10GB of space. In particular, we used it to combine and sort the files as well as to remove duplicate lines and replace special characters. The code snippet 4.1 displays all the used bash scripts.

Listing 4.1: Bash Commands

```
# merge files
cat file1.csv file2.csv > file_merged.csv

# replace newline with escaped newline if \
  line does not start with 15 numbers
perl -0pe 's/\n(?!\d{15})/\\n/g' file_merged.csv > file_merged2.csv

# replace multi-escaped newline with single escaped newline
perl -0pe 's/\\+n/\\n/g' file_merged2.csv > file_merged3.csv

# remove carriage return
sed -i -e 's/\r//g' file_merged3.csv

# sort file & remove duplicates
sort -g -u file_merged3.csv > file_finished.csv
```

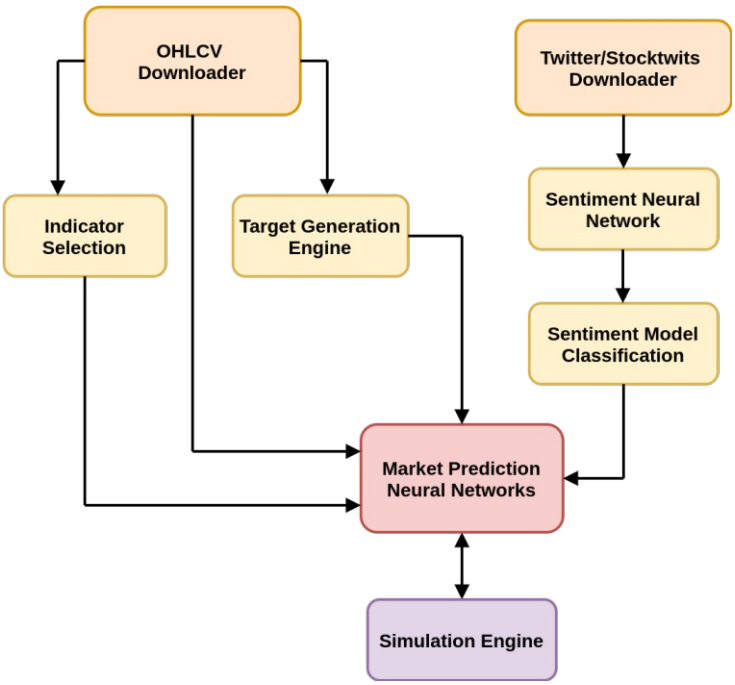
4.2 System Implementation

We implemented the system to be as flexible and automated as possible. This allows us to easily update the structure and parameters of the neural networks and to use as many different input sources as the user of our system requires. Additionally, it allows the system to work with any kind of asset e.g. Stocks, Forex, Crypto, and any time-based input source like price, technical data or fundamental data. Lastly, this flexible design provides the possibility to test different target settings, risk levels as well as different trading options like short trading. The implementation itself is split into eight modules which interact with each other. These modules are listed in sections 4.2.1 to 4.2.8. Furthermore, figure 4.1 displays how these different modules interact with each other to form the whole system.

4.2.1 OHLCV Downloader

The first module is the *OHLCV Downloader* which is used to download the OHLCV candles from the Binance exchange. The downloader fetches the data with multiple *GET* requests from the */api/v3/klines* endpoint of the official Binance API [144]. The listing 4.2 displays an example result from the endpoint with a description of every attribute. The

Figure 4.1: Illustration of module interaction



downloader module supports single and concurrent downloads and provides settings to download data for different intervals, cryptocurrencies as well as to specify the daterange of the fetched data. After the download is finished the *JSON* data is converted to a Pandas dataframe and preprocessed according to the description in 3.2.1.

Listing 4.2: Binance kline example

```
[
  [
    1499040000000,      // Open time
    "0.01634790",       // Open
    "0.80000000",       // High
    "0.01575800",       // Low
    "0.01577100",       // Close
    "148976.11427815",  // Volume
    1499644799999,      // Close time
    "2434.19055334",    // Quote asset volume
    308,                // Number of trades
    "1756.87402397",    // Taker buy base asset volume
    "28.46694368",      // Taker buy quote asset volume
    "17928899.62484339" // Ignore.
  ]
]
```

4.2.2 Target Generation Engine

The second module is the *Target Generation Engine* and implements the algorithm described in 3.2.2. This module takes the preprocessed OHLCV dataframe from the *OHLCV Downloader* 4.2.1 as input and calculates the trading signal for each row. The module can be configured with the parameters *minimum profit target* and *maximum accepted loss* to customize the taken risk and to adjust the number of entered trades. A detailed description for these parameters can be found in 3.2.2. The generated target values are added to the OHLCV dataframe which is afterwards persisted on the hard drive to avoid unnecessary re-computations between multiple training processes with the same threshold values and OHLCV data.

4.2.3 Market Prediction Neural Networks

The third module provides the entire functionality to build, optimize, train, and evaluate the market prediction neural networks. The core part of this module is an abstract model class which provides functions to train and evaluate a neural network, to predict the signal for a single input value as well as to save and load the weights for a trained network. This abstract model is implemented by our customizable base model which can be configured with a configuration object. The model itself was built with the *Tensorflow* library and uses the parameters from the configuration object to adjust its inner model structure as well as to set the values of the hyperparameters. All the different parameters for the configuration object can be found in 3.2.4. The search process of the search & selection algorithm 3.4 was implemented with the *sweep* tool from the *Wandb* library [133]. This tool only requires the possible parameter values and a search strategy for the iterative process before it can be initiated to train the neural networks in an endless loop. We decided to use the random search strategy as already mentioned in 3.4. After a search process has been manually stopped, the trained networks can be evaluated and compared with the provided evaluation methods of the module. This methods use the *Simulation Engine* module for the performance calculation of the obtained trading strategy, to print multiple comparison charts about the trained networks and to save different metrics for the network and the trading strategy to a file which is afterwards used for the final manual model selection.

4.2.4 Indicator Selection

The *Indicator Selection* is the forth module of the implemented system. This module is responsible for the calculation of the indicator values, the search & selection process for the best indicators as well as the generation of the indicator combinations. For the calculation process we implemented a generic indicator function which is able to calculate all indicators from the *talib* library while only requiring a simple configuration object. This object contains information about which data from the OHLCV candle should be used for the calculations, the indicator configuration itself as well as preprocessing steps that will be applied on the indicator results. The search & selection process is again a

semi-automated process and implemented with the *sweep* tool of the *Wandb* library. To find the best indicator combinations multiple search & selection processes were executed. The first execution of this optimization algorithm iterated over all possible indicators and was required to specify the best performing single indicators. All subsequent runs generated indicator combinations with the results from their previous selection run as well as the best single indicators from the first run. These search & selection runs can be executed repeatedly to combine as many indicators as wanted.

4.2.5 Simulation Engine

The next module is the implementation of the *Simulation Engine*. This module is responsible for the simulation of trading strategies based on the results of the trained market prediction neural networks. As mentioned in 3.2.5 each simulation calculates multiple statistics and plots a chart which visualizes a performance comparisons with the buy-and-hold strategy.

In detail, the *Simulation Engine* uses an iterative approach that starts at the earliest timestamp of the provided data and stops at the last one. During each iteration the data of the current timestamp is supplied to the trained neural network for the signal prediction. These signals are then used by the engine to check the current trading position and, if necessary, update it e.g. when the signal changes from BUY to SELL or from SELL to BUY. Every time a trading position is adapted, a trading fee is subtracted from the current position size to simulate a more realistic trading environment. When short trading is enabled, the open position switches directly from LONG to SHORT and vice versa, meaning that there always remains an open trading position.

Specifically, the engine supports the configurations *Position Size*, *Stop Loss*, *Trailing Stop Loss*, *Short Trading*, and *Date Range* as described in 3.2.5. Each of these parameters potentially contained multiple values during a single execution. We developed the engine to automatically calculate the simulations for every possible parameter combination. Furthermore, if more than one simulation was executed for a single neural network, the performance charts of the simulations were displayed side by side. This allowed for a faster visual comparison of different settings like date ranges, short trading, stop loses, and other risk settings.

4.2.6 Twitter/Stocktwits Downloader

The sixth module consists of two download tools for the twitter and stocktwits messages. The twitter downloader uses the *Snscrape* [138] library while the stocktwits tool uses the *requests* library to download the messages. Both tools are able to automatically manage their download speed to avoid being banned for exceeding the rate limits of the platforms. All downloaded messages are stored in csv files which need the be manually preprocessed with the provided bash scripts described in 4.1. Afterwards the messages are loaded into a Pandas dataframe before a second preprocessing procedure is executed. This procedure cleans the dataframe with multiple filter methods which are described in 3.3.2.

4.2.7 Sentiment Neural Networks

The next module provides the whole functionality for the *Sentiment Neural Networks*. This module is based on the *Pytorch-Lighthing* and *Huggingface* machine learning libraries in contrast to the *Tensorflow* library which was used for the market prediction neural networks. The module consists of an abstract model class which provides the base structure to fine-tune, evaluate, and test the six selected transformer models: *BERT*, *Roberta*, *Roberta-Twitter*, *BART*, *XLNet*, and *T5*. In detail, we built two generic implementations for the abstract base class. The first one uses the class *AutoModelForSequenceClassification* while the second one is based on the class *AutoModelForSeq2SeqLM*, both of which are abstract classes provided by the *Huggingface* library. *AutoModelForSequenceClassification* adds an output layer to the provided model with the task of sequence classification. It can be used for the *BERT*, *Roberta*, *Roberta-Twitter*, and *XLNet* models. *AutoModelForSeq2SeqLM*, on the other hand, adds an output layer for sequence predictions and can be used by models with a sequence to sequence architecture like *Bart* and *T5*.

Generally, the training- and evaluation process was implemented with the *Pytorch-Lighthing* library while the pretrained models were provided by the *Huggingface* library. The model selection- and the hyperparameter optimization process were again performed by the search & selection algorithm, which was once more implemented with the *sweep* tool of the *Wandb* library.

4.2.8 Sentiment Model Classification

The last module was used for the classification of the twitter and stocktwits messages as well as to transform the classified data into a time series input feature for the market prediction neural network. First, the module loaded a fine-tuned sentiment neural network from the previous section and used it to predict the probabilities of each sentiment class (*positive*, *neutral*, and *negative*) for every message. Afterwards, the sentiment values were aggregated to match the interval of the other input features with the *resample* and *agg:sum* methods of the *Pandas* library. The resulting data was afterwards used by the market prediction neural network as an additional input feature.

4.3 Optimizations & Hardware

The last section of the chapter outlines different code optimizations which were applied while developing the described system. Then, the utilized hardware is presented as well as information about the training times.

4.3.1 Optimizations

We developed the system to perform thousands of neural network training- and parameter optimization processes. These processes require substantial amounts of computer resources and computation time. Therefore, to reduce the overall runtime for the procedures, we optimized multiple parts of the system's code.

For the target generation engine we used the *joblib* Python extension to perform concurrent calculations of the target values. This optimization splits the input data into separate parts and performs the target calculations for each part simultaneously on different cores of the systems. Additionally we added a cache system to improve the performance of the forward and backward search process for the threshold values. Afterwards the results are persisted on the disk to avoid re-computations for the same target configurations.

Our simulation engine is strongly interconnected with the training process of the market prediction neural networks because the simulation results are used to guide the learning process towards the best performing strategy. Since both the training- and the simulation process take a long computation time it is important to execute them in parallel. Therefore, we developed a Tensorflow callback class to execute the simulation process after each epoch of the training procedures. This callback runs in a parallel process which allows the system to train the next epoch of the neural network while simultaneously computing the simulations of the previous one.

The last optimization was implemented in the sentiment classification process. The default prediction methods of the *Pytorch* and *Pytorch-Lighting* libraries use the CPU and can only be utilized to predict a single item at a time, which is painstakingly slow and was not feasible for our dataset of 10+ million messages. Therefore, we implemented an algorithm to transform the microblogging messages into tensor-batches which could afterwards be used by the GPU for the prediction process. This step improved the performance of our classification process by a factor of 100 compared to the default CPU prediction method.

4.3.2 Hardware

The training processes of neural networks are mainly based on different matrix operations and require large amounts of computational resources for their calculations. Graphic processing units (GPUs) are able to perform these operations in a substantially more efficient way than CPUs because of their high memory bandwidth and their parallel execution capabilities. Consequently, most machine learning problems are solved by high-end GPUs. Additionally, the big datasets which are often used for the training processes of deep neural networks require a vast amount of memory. For these reasons, we decided to acquire a new workstation with the following configuration:

CPU: AMD® Ryzen 7 3700x 8-core processor × 16
GPU: GeForce RTX 2070 SUPER
Ram: 64 GB

This workstation required up to five minutes for the training process of a single epoch for the market prediction neural network with the CPU. By using the GPU, on the other hand, the training time for a single epoch was cut to roughly 20 seconds. The fine-tuning process for the sentiment models represents a more time-consuming task that took up to 20 minutes for a single epoch on the GPU depending on the selected transformer model.

The overall training time for all models was about three to four months with a 24/7 runtime.

Concluding, the newly acquired hardware has proven to be crucial as it substantially increased the system's performance and significantly shortened the development and evaluation time.

CHAPTER 5

Results and Evaluation

This chapter covers the different experiment setups, the produced results, and the performance of the generated models. Section 5.1 describes how the final parameter settings for the different parts of the market prediction neural network were obtained. Afterwards, the section 5.2 describes the evaluation procedure and results for the transformer models for the financial sentiment analysis. In the following section 5.3 the used technical indicators and the selection process, which was used to find the best performing indicator combinations, are addressed. In the last section 5.4, the trained market prediction models are evaluated as well as compared and the remaining research questions from section 1.2 are answered.

5.1 Market Prediction Neural Networks

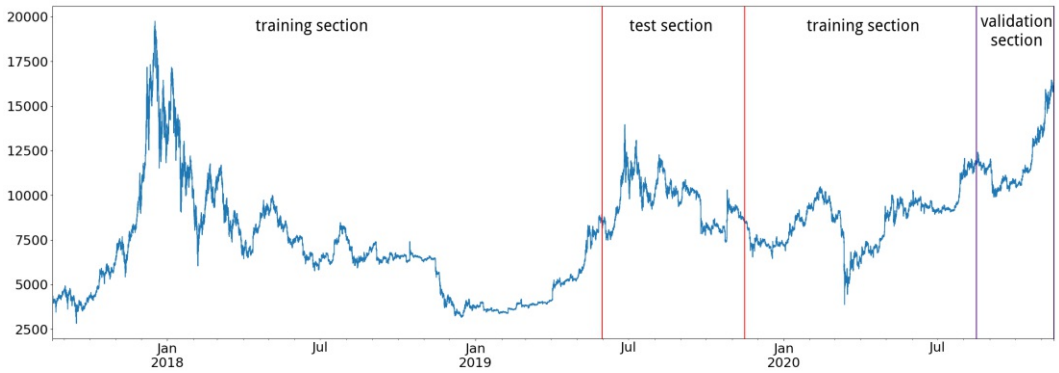
The first part of this section describes how we split the OHLCV input data for the training, testing and validation steps of the market prediction neural network. Afterwards, sections 5.1.1 to 5.1.1 present the optimization results and the decisions to obtain the final parameter configurations for the different parts of the market prediction procedure.

5.1.1 OHLCV Data - train, test & validation Sections

As mentioned in section 3.2.1, the main input source for the market predictions are the OHLCV candles from the Binance exchange. The analyzed time frame ranges from *2017-08-18* to *2020-11-15* and was split into three parts for the training, testing, and validation of the models. The periods for testing and validation were chosen strategically. Training, on the other hand, occurred continuously throughout the time frame excluding these previously mentioned sections. The range for testing starts at *2019-05-30*, ends at *2019-11-15*, and was selected because it contains characteristics of a bull market cycle, a bear market cycle and a sideways market cycle. Naturally, the last three months of the

input data were used for the validation of the models. Figure 5.1 displays the whole date range and marks the specific sections.

Figure 5.1: Price input data with sections



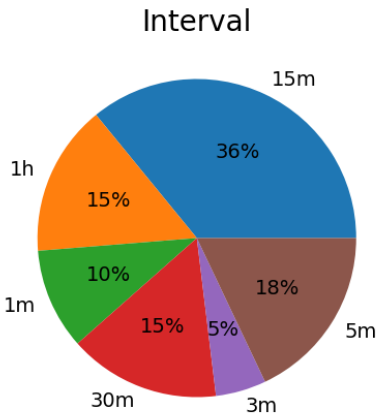
Optimization: OHLCV Interval

The first parameter we optimized is the *Interval* parameter of the aggregated OHLCV data. As mentioned in section 3.2.1, short intervals are mostly used for short term predictions while longer intervals are utilized for long term prognosis. With the goal of short- to mid-term predictions we used the following values for the optimization procedure of the *Interval* parameter:

- 1 minute
- 3 minutes
- 5 minutes
- 15 minutes
- 30 minutes
- 1 hour

The distribution of the interval values for the best performing models is displayed with the pie chart in figure 5.2. As the figure shows, the 15 minutes interval outperformed all other intervals by at least 17% and was consequently selected as the default value for all further models.

Figure 5.2: Optimization results: Interval



Parameter	Values
Minimum Profit Target	0.1, 0.3, 0.5, 0.7, 1.0, 2.0, 3.0, 5.0
Maximum Accepted Loss	0.1, 0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 5.0

Table 5.1: Evaluated parameter values: Minimum Profit Target & Maximum Accepted Loss

Optimization: Minimum Profit Target & Maximum Accepted Loss

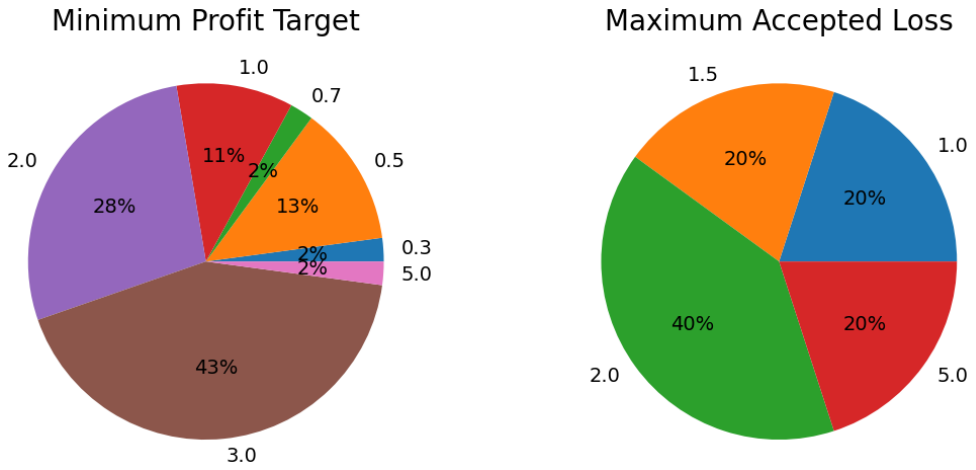
The next analyzed parameters are the *Minimum Profit Target* and the *Maximum Accepted Loss* which are used within the *Target Generation Engine* and affect the number of generated *BUY*, *SELL* and *HOLD* signals. A detailed description for these two parameters can be found in section 3.2.2.

Table 5.1 displays an overview of all the tested *Minimum Profit Target* and the *Maximum Accepted Loss* values. A comprehensive breakdown of the test's outcome is illustrated underneath by figure 5.3. The left side of the figure shows the distribution of the *Minimum Profit Target* parameter for the models which outperformed a buy-and-hold strategy. A *Minimum Profit Target* of 3% was chosen for all further target values based on the finding that it outperformed all the other settings by at least 14.9%. The right side of figure 5.3 displays the *Maximum Accepted Loss* value distribution for all models that outperformed a buy-and-hold strategy and used a *Minimum Profit Target* of 3%. The dominant observed strategy was a *Maximum Accepted Loss* of 2% which outperformed all other settings and was selected for all further target values.

Optimization: Sequence Length & Batch Size

The next group of analyzed parameters includes the *Sequence Length*, which specifies the number of timestamps that are fed to the network as a single training example, and the

Figure 5.3: Optimization results: Minimum Profit Target & Maximum Accepted Loss



Parameter	Values
Sequence Length	50, 100, 200, 400, 1000
Batch Size	32, 50, 64, 200, 300, 1000

Table 5.2: Evaluated parameter values: Sequence Length & Batch Size

Batch Size, which is the number of used training examples during a single iteration of the training process.

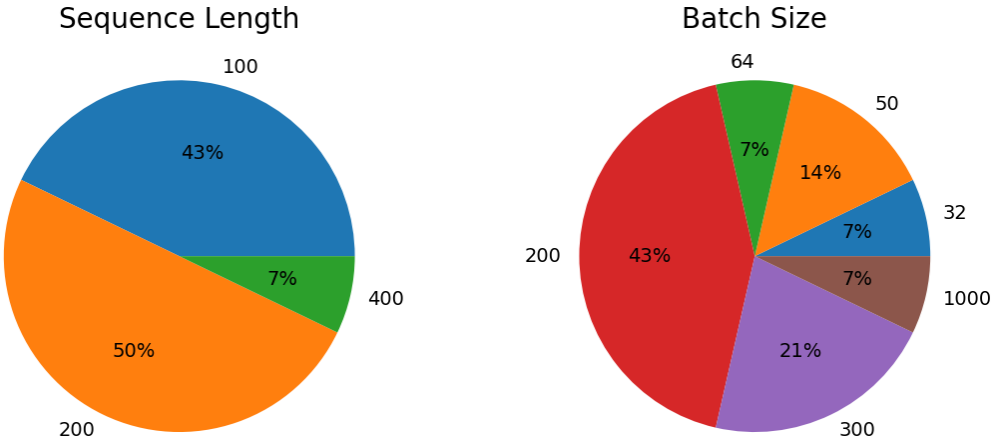
Table 5.2 displays the evaluated parameter values for the optimization process. The result distribution for the *Sequence Length* and the *Batch Size* for the best performing models is illustrated in figure 5.4. 50% of the models were using a *Sequence Length* of 200 and 42.9% were using a *Batch Size* of 200, thus these settings were considered suitable and chosen for all further neural networks.

Optimization: Neural Network Structure & Hyperparameters

As described in section 3.2.4, we added thirteen optimizable parameters to Aumayr's base model which are able to change the inner network structure and to affect the learning process of the neural networks. These parameters were optimized with three separate search & selection loops to find a proper configuration for the final neural network setup. The evaluated parameter values for these three loops are displayed in table 5.3.

Within a time span of two weeks, 11481 models were trained for the optimization process of the neural network structure and hyperparameters. All models which outperformed a buy-and-hold trading strategy during the test daterange were then extracted. To validate the isolated impact, the number of occurrences for each parameter value was

Figure 5.4: Optimization results: Sequence Length & Batch Size



	first loop	second loop	third loop
Dropout	0.4, 0.5, 0.6	0.3, 0.4	0.1, 0.2, 0.3, 0.4
CNN filter size	0, 8, 16, 32, 64, 128	32, 64	64, 80, 100, 128, 150, 180
RNN type	LSTM, GRU	LSTM	LSTM, GRU
RNN size	8, 16, 32, 64, 128, 256	64	128
Dense size	0, 8, 16, 32, 64, 128, 256	64	256
L1	0, 0.1, 1e-3, 1e-4, 1e-5	0, 0.1, 1e-3, 1e-4, 1e-5	0, 1e-5
L2	0, 0.1, 1e-3, 1e-4, 1e-5	0, 0.1, 1e-3, 1e-4, 1e-5	1e-4, 1e-5
Weight Decay	1e-7, 1e-6, 1e-5, 5e-4 1e-4, 5e-3, 5e-2	1e-4, 5e-4, 1e-3, 5e-2, 0.01	5e-4
Learning Rate	5e-3, 1e-3, 5e-4, 1e-4	5e-4	5e-4
Optimizer	adam, nadam, sgd, rmsprop	nadam	nadam, rmsprop, adadelta, adagrad
Activation	linear, relu, elu, selu, softmax	softmax, selu	softmax, selu, relu

Table 5.3: Evaluated parameter values: Market Prediction Neural Network

calculated for the list of all models as well as for the list of the best performing models. If a parameter value had a higher number of occurrences in the list of best models, then this parameter value had a positive impact on the performance of the neural network. The best parameter values were selected based on the highest percentage increase in the list of these best models. The calculated percentage of occurrences are listed in table 5.4 for the first loop, 5.5 for the second loop, and 5.6 for the third loop. The selected values for the subsequent loops are highlighted in each table.

Table 5.7 displays the final parameter configurations which were used for all further analysis. As the table shows, the LSTM layer outperformed the GRU layer. This result was to be expected since the GRU layer is a simplified version of the LSTM layer. Furthermore, it becomes apparent that the inclusion of the CNN layer and the Dense

Layer mayorly contributed to the overall performance of the models, which confirms the research results from Aumayr [9].

5.2 Financial Sentiment Neural Networks

This section provides the detailed steps to find the best performing transformer model for the financial sentiment analysis of microblogging messages. The first part is dedicated to the datasets for the fine-tuning process of the pre-trained transformer models. The next paragraphs describe the optimization process, its results, and the final configuration for the neural network hyperparameters. The last part of this section outlines the classification process of the microblogging messages.

Financial Sentiment Datasets

Several financial sentiment datasets were utilized in different combinations in order to train, test, and validate the sentiment transformer models. The different datasets and their origin are presented in section 3.3.2. We constructed the datasets *All* and *All (Finbank-balanced)* from the obtained financial sentiment datasets for the training, testing, and validation procedure of the transformer models. For this purpose, they were randomly split into 80% training data and equally 10% test and validation data. A description of these two datasets can be found in the list below.

- **All**
  All is a merged dataset of *Finbank-50*, *Semeval-headlines*, *Semeval-microblog* and *Stocktwits*. The class distribution of this dataset is listed below:
  - positive classes: 4836
  - negative classes: 3321
  - neutral classes: 3267
- **All (Finbank-balanced)**
  All (Finbank-balanced) is a merged dataset of *Finbank-balanced*, *Semeval-headlines*, *Semeval-microblog* and *Stocktwits*, and has the following class distribution:
  - positive classes: 4044
  - negative classes: 3321
  - neutral classes: 998

Optimization: Sentiment Neural Network - Hyperparameters

We optimized the following hyperparameters of the sentiment models with our search & selection algorithm:

Parameters	values	% of all models	% of best models	percent change
Dropout	0.6	20.4	5.9	-14.5
	0.5	39	28.8	-10.2
	0.4	40.6	65.3	24.7
Learning Rate	0.005	16	7.2	-8.8
	0.001	23.2	22.5	-0.7
	0.0005	33.9	48.7	14.8
	0.0001	26.9	22.5	-4.4
CNN filter size	0	7.4	2.1	-5.3
	8	13.5	8.5	-5
	16	14.5	10.2	-4.3
	32	18.6	22.9	4.3
	64	23.7	26.3	2.6
	128	22.2	30.1	7.9
RNN type	lstm	34.1	32.6	-1.5
	gru	65.9	67.4	1.5
RNN size	8	13.5	6.8	-6.7
	16	13.9	5.5	-8.4
	32	11.8	5.1	-6.7
	64	16.1	16.9	0.8
	128	23.5	35.2	11.7
	256	21.2	30.5	9.3
Dense size	0	14.8	18.2	3.4
	8	12.5	7.2	-5.3
	16	16.6	5.9	-10.7
	32	11.6	11.4	-0.2
	64	15	22	7
	128	13.5	12.7	-0.8
	256	16.1	22.5	6.4
Optimizer	adam	13.1	10.2	-2.9
	nadam	21.3	32.2	10.9
	sgd	29.3	16.1	-13.2
	rmsprop	36.3	41.1	4.8
Activation	softmax	14.8	17.4	2.6
	relu	17	18.2	1.2
	selu	25	27.5	2.5
	elu	27.1	24.2	-2.9
	linear	16.1	12.7	-3.4
Weight Decay	0.05	21.3	19.9	-1.4
	0.005	16.1	15.3	-0.8
	0.0005	12.9	14	1.1
	0.0001	15.9	25	9.1
	1.00E-05	13.9	11	-2.9
	1.00E-06	12.5	11.4	-1.1
	1.00E-07	7.5	3.4	-4.1

Table 5.4: Market Prediction Neural Network - first optimization loop

Parameters	values	% of all models	% of best models	percent change
Dropout	0.3	49.8	52.6	2.8
	0.4	50.2	47.4	-2.8
CNN size	32	49.7	53.7	4
	64	50.3	46.3	-4
Activation	relu	52.1	53.3	1.2
	softmax	47.9	46.7	-1.2
L1	0	22.5	30.1	7.6
	0.1	24.2	0.9	-23.3
	0.001	16.7	18.9	2.2
	0.0001	20.7	28.1	7.4
	1.00E-05	15.9	22.1	6.2
L2	0	19.2	20.6	1.4
	0.1	19.8	13.1	-6.7
	0.001	21.7	23.5	1.8
	0.0001	18.8	20.9	2.1
	1.00E-05	20.4	21.9	1.5
Weight Decay	0.05	20.1	20.4	0.3
	0.01	20.6	18.6	-2
	0.001	16.1	14.3	-1.8
	0.0005	21.4	24.3	2.9
	0.0001	21.9	22.4	0.5

Table 5.5: Market Prediction Neural Network - second optimization loop

- **model**
  Specifies the transformer model.

- **datasource**
  The training, testing, and validation data for the model.

- **learning rate**
  The learning rate for the defined model.

- **learning rate warmup**
  If set to *true* the learning rate slowly increases till it reaches the value specified in *learning rate* during the first epoch.

- **semeval split**
  The semeval dataset contains sentiment values between -1 and 1. With this parameter we can define a threshold around 0 for the neutral sentiment class.

- **used weighted loss**
  If set to *true* a weighted loss function is used. This can improve the results for imbalanced data.

Parameters	values	% of all models	% of best models	percent change
Dropout	0.4	25.7	17	-8.7
	0.3	26.8	22.6	-4.2
	0.2	25.8	29.5	3.7
	0.1	21.7	30.9	9.2
CNN filter size	64	13.4	13.7	0.3
	80	15.9	15.8	-0.1
	100	17.2	16.7	-0.5
	128	17.1	15.9	-1.2
	150	18.6	19.5	0.9
	180	17.8	18.5	0.7
RNN type	lstm	44.8	54	9.2
	gru	55.2	46	-9.2
Dense size	64	47.4	49.8	2.4
	256	52.6	50.2	-2.4
Optimizer	nadam	18.8	20.8	2
	adadelta	26.2	23.7	-2.5
	adagrad	35.5	29.1	-6.4
	rmsprop	19.5	26.4	6.9
Activation	softmax	32.6	30.2	2.4
	relu	33.5	35.9	2.4
	selu	33.9	33.8	-0.1
L1	0	42.4	45.1	2.7
	1.00E-05	57.6	54.9	-2.7
L2	0.0001	42.9	44.9	2
	1.00E-05	57.1	55.1	-2

Table 5.6: Market Prediction Neural Network - third optimization loop

Parameter	Value
Dropout	0.1
CNN filter size	64
RNN type	LSTM
RNN size	128
Dense size	256
L1	0
L2	0.0001
Weight Decay	0.0005
Learning Rate	0.0005
Optimizer	rmsprop
Activation	softmax

Table 5.7: Market Prediction Neural Network - final configuration

Parameter	Values
model	bert, xlnet, roberta, roberta-twitter, t5, bart
learning rates	
bert	2e-05, 3e-05, 1e-05, 5e-06
xlnet	3e-05, 2e-05, 5e-06, 1e-05
roberta	5e-06, 8e-06, 1e-05, 2e-5
roberta-twitter	5e-06, 8e-06, 1e-05, 2e-5
t5	5e-04, 7e-04, 1e-03, 2e-03
bart	4.8e-05, 2e-05, 6e-05, 3e-05
input data	All, All-FinBank_balanced
learning rate warmup	true, false
semeval split	0, 0.05, 0.1, 0.2
use weighted loss	true, false

Table 5.8: Evaluated parameter values: Sentiment Model

model name	macro f1	macro precision	macro recall
xlnet	0.7999	0.8081	0.7998
bert	0.7872	0.7958	0.7847
t5	0.7998	0.8034	0.8033
bart	0.8115	0.8174	0.8112
roberta	0.8322	0.8366	0.8326
roberta-twitter	0.8303	0.8345	0.8302

Table 5.9: Average performance for each model

Table 5.8 displays all the parameter values which were used during the optimization process to find the best configuration for the sentiment model.

Table 5.9 shows the average performance result for each of the trained model types. It becomes apparent that the *roberta* and the *roberta-twitter* models outperformed the other models by 2-3%. Ultimately, the *roberta* model was selected as the final sentiment model due to its superior performance, albeit marginal.

Afterwards, a second hyperparameter optimization process was performed to derive the final configuration for the roberta model. For this process, the best model was selected based on the *macro f1 score*. The parameters for this final sentiment model are listed in table 5.10 and its performance is displayed in table 5.11.

The result in table 5.11 shows a macro f1 score of 84.8% for the model, which can be interpreted as a superior performance compared to most other publicly known financial sentiment classification models [145] [146]. A score above 80% can generally be regarded as extremely promising, since sentiment is highly subjective and different people have different opinions on the same piece of text [147]. Furthermore, the model surpassed the financial sentiment specific models from the SemEval Task 2017 [148] and FiQA 2018

Parameter	Value
transformer model	roberta
datasource	All
learning rate	5e-06
learning rate warmup	true
semeval split	0.1
use weighted loss	true

Table 5.10: Roberta parameter configuration

Parameter	Value	Parameter description
macro f1	0.848	f1 score with same importance for each class
macro precision	0.822	precision with same importance for each class
macro recall	0.84	recall with same importance for each class
weighted f1	0.849	f1 score with weighted classes
weighted precision	0.854	precision score with weighted classes
weighted recall	0.85	recall score with weighted classes
positive f1	0.855	f1 score for the 'positive' class
positive precision	0.807	precision for the 'positive' class
positive recall	0.909	recall for the 'positive' class
negative f1	0.834	f1 score for the 'negative' class
negative precision	0.867	precision for the 'negative' class
negative recall	0.804	recall for the 'negative' class
neutral f1	0.856	f1 score for the 'neutral' class
neutral precision	0.91	precision for the 'neutral' class
neutral recall	0.807	recall for the 'neutral' class

Table 5.11: Roberta model performance

[149] which were trained with the same and/or a similar dataset as our models.

Sentiment Classification of Microblogging Data

For the classification of microblogging data, a twitter and a stockswits scraper was built as described in section 4.2.6. These download tools were used to obtain *15,935,356* twitter messages for the hash tags *#btc* and *#bitcoin* and *1,169,117* stockstwits messages for the trading symbol BTC.X. The applied date range for the downloaded messages was equal to the one utilized for the price data: *2017-08-18* to *2020-11-15*.

Afterwards, the messages were classified through the optimized roberta sentiment model. In the last step, the classified sentiment data was transformed into a 15-minute interval which enabled its use as an additional input source for the market prediction neural network.

5.3 Technical Indicator Selection

In this section, the process to find and select the best technical indicator combinations is described. The first part outlines the available technical indicators and the data which was used for their calculations. In the second part, a qualitative evaluation of the selection process as well as the results for each selection round are provided.

5.3.1 Technical Indicators

We tested and evaluated 62 different technical indicators from the talib library [150] which are described in depth in section 2.2.3. The indicators are grouped into five categories - namely *Overlap Studies*, *Momentum*, *Volume*, *Volatility*, and *Statistical Functions*. A detailed list of all indicators, their category, input data, and used settings can be found in the tables 5.12 to 5.16. The input data column of these tables describes the different parts of the OHLCV candle which were used as input for the indicator calculations. The default settings were used for every indicator calculation, which are listed in the settings column. Additionally, all indicators use a default time period of 14 (tp14), which means that 14 previous candles are used for the calculation of the next indicator value.

5.3.2 Optimization: Technical Indicators

During the technical indicator evaluation stage, multiple evaluation loops were used to find the best performing indicator combination as mentioned in section 3.2.3. Each loop trained a few thousand neural networks with a random set of input indicators. After each loop, the number of used input indicators was increased by one, starting with one indicator for the first loop. The number of possible input indicators was then reduced after each loop and limited to the best performing ones of the previous loops. In total, three loops were used and the best performing indicator combinations were then compared. To evaluate the models, a similar procedure as in the parameter optimization process for the neural network structure was followed:

- Extraction of all models which outperformed a buy-and-hold trading strategy and storage into a separate list called *best models*
- Calculation of the percentage occurrence for each indicator combination for the list of *best models* and for the list of *all models*
- Calculation of the difference between the percentage occurrence between the two lists
- If an indicator combination has a higher percentage occurrence in the best list then this indicator combination performed better than the average
- If the percentage increase for an indicator combination is above 50% then this combination is selected for the next selection loop

Indicator Name	Input Data	Settings
ADX	high, low, close	tp14
ADXR	high, low, close	tp14
APO	close	fastperiod: 12, slowperiod: 26, matype: 0
AROON	high, low	tp14
AROONOSC	high, low	tp14
BOP	open, high, low, close	tp14
CCI	high, low, close	tp14
CMO	close	tp14
DX	high, low, close	tp14
MACD	close	fastperiod: 12, slowperiod: 26, signalperiod: 26
MFI	high, low, close	tp14, volume
MINUS_DI	high, low, close	tp14
MINUS_DM	high, low	tp14
MOM	close	timeperiod: 10
PLUS_DI	high, low, close	tp14
PLUS_DM	high, low	tp14
PPO	close	fastperiod: 12, slowperiod: 26, matype: 0
ROC	close	timeperiod: 10
RSI	close	tp14
STOCH	high, low, close	fastk_period: 5, slowk_period: 3, slowk_matype: 0, slowd_period: 3, slowd_matype: 0
STOCHF	high, low, close	fastk_period: 5, fastd_period: 3, fastd_matype: 0
STOCHRSI	close	timeperiod: 14, fastk_period: 5, fastd_period: 3, fastd_matype: 0
TRIX	close	timeperiod: 30
ULTOSC	high, low, close	timeperiod1: 7, timeperiod2: 14, timeperiod3: 28
WILLR	high, low, close	tp14

Table 5.12: Momentum Indicators

Indicator Name	Input Data	Settings
ADOSC	high, low, close, volumn	fastperiod: 3, slowperiod: 10
RSI_vol	volume	tp14
OBV	close, volume	

Table 5.13: Volume Indicators

Indicator Name	Input Data	Settings
ATR	high, low, close	tp14
TRANGE	high, low, close	

Table 5.14: Volatility Indicators

Indicator Name	Input Data	Settings
BBANDS	close	timeperiod: 5, nbdevup: 2, nbdevdn: 2, matype: 0
DEMA	close	timeperiod: 30
EMA	close	timeperiod: 30
HT_TRENDLINE	close	
KAMA	close	timeperiod: 30
MAMA	close	fastlimit: 0, slowlimit: 0
SMA	close	timeperiod: 30
SAR	high, low	acceleration: 0, maximum: 0
T3	close	timeperiod: 5, vfactor: 0
TEMA	close	timeperiod: 30
TRIMA	close	timeperiod: 30
WMA	close	timeperiod: 30

Table 5.15: Price Indicators (Overlap Studies)

Indicator Name	Input Data	Settings
BETA	high, low	timeperiod: 5
CORREL	high, low	timeperiod: 30
LINEARREG	close	tp14
LINEARREG_ANGLE	close	tp14
LINEARREG_INTERCEPT	close	tp14
LINEARREG_SLOPE	close	tp14
STDDEV	close	timeperiod: 5, nbdev: 1
TSF	close	tp14
VAR	close	timeperiod: 5, nbdev: 1

Table 5.16: Statistical Indicators

The best performing indicators for the first and second evaluation loop are listed in the tables 5.17 and 5.18. The third evaluation loop did not contain any model that outperformed the buy-and-hold trading strategy.

The tables include the following values for each indicator combination:

- # all: total number of trained models
- # best: number of models which outperformed a buy-and-hold trading strategy
- pct all: percentage of all trained models
- pct best: percentage of the best trained models
- pct diff: difference in percent between *pct all* and *pct best*

As the results clearly demonstrate, indicators have a significant impact on the performance of our models. The indicators PPO, AROONOSC, and ROC of the first selection loop, for example, have a percent improvement of more than 200% over the average indicator

	# all	# best	pct all	pct best	pct diff
PPO	820	180	1.71	6.43	276.8
AROONOSC	840	170	1.75	6.08	247.4
ROC	737	141	1.53	5.04	228.4
CCI	880	147	1.83	5.25	186.74
BOP	1160	175	2.42	6.25	158.96
CMO	860	104	1.79	3.72	107.58
MFI	1160	137	2.42	4.9	102.73
WILLR	1000	118	2.08	4.22	102.55
RSI	1080	119	2.25	4.25	89.14
STOCHRSI	880	87	1.83	3.11	69.7

Table 5.17: Results for first indicator selection loop

	# all	# best	pct all	pct best	pct diff
PPO, ROC	385	110	1.82	4.56	150.71
ROC, STOCHRSI	431	106	2.04	4.4	115.81
BOP, PPO	348	84	1.64	3.48	111.81
PPO, AROONOSC	473	110	2.24	4.56	104.06
AROONOSC, ROC	363	82	1.72	3.4	98.22
AROONOSC, MFI	479	104	2.26	4.31	90.52
AROONOSC, STOCHRSI	444	92	2.1	3.82	81.82
ROC, CCI	303	57	1.43	2.36	65.07
BOP, ROC	312	58	1.47	2.41	63.12
MFI, ROC	372	68	1.76	2.82	60.4

Table 5.18: Results for second indicator selection loop

result. The percent improvement for the best indicator combinations in the second selection outperform the average results by up to 150%. Evaluating these results, we can conclude that the usage of technical indicators is able to vastly improve the performance of our models. While the level of performance increase varies depending on the specific indicator combination, the overall trend seems robust. Therefore, we can answer the following research question with a yes:

Can the utilization of different combinations of technical indicators improve the results of the price prediction?

To compare the results of the different indicator selection loops, the average accuracy, loss, sharpe ratio and sortino ratio was calculated for all models that outperformed a buy-and-hold strategy. The results of these calculations are displayed in table 5.19. The numbers depict a modest performance increase as well as a slight decrease in the validation loss of the trained network on the second indicator loop.

	no indicator	1x indicator	2x indicators
avg accuracy	0.368	0.368	0.361
avg loss	1.237	1.263	1.306
avg sharpe ratio	1.378	1.691	1.777
avg sortino ratio	2.135	3.046	3.255

Table 5.19: Average performance of indicator combinations

5.4 Final Result Evaluation

In this section, the performance of Aumayr's base model is compared to the optimized deep learning models constructed for this thesis. Afterwards, multiple simulations are performed to differentiate between the best technical indicators of each selection round and to further test the impact of the sentiment data. The final part of this section outlines adjustments to the trading settings to lower the risk of the final trading strategy.

5.4.1 Model Comparison - Base, Optimized, Sentiment

For the evaluation of the price prediction performance, a comparison between Aumayr's base model, the optimized base model, and the optimized base model with sentiment data was carried out. 150 neural networks were trained for each model and a calculation of the average accuracy, loss, sharpe ratio, and sortino ratio was conducted. An overview of the calculated results is presented in table 5.20. As the table displays, the optimized models were able to achieve a lower validation loss compared to the base model which indicates the new model's ability to learn the desired targets more accurately. The base model and the sentiment model achieved a similar performance, while the optimized model outperformed all other models. These results show that we are able to improve the learning capabilities and predictions of our neural networks by optimizing their structure and hyperparameters, which allows us to answer the following research question with a yes:

Is it possible to improve the results of a price prediction system through an optimization of the internal structure and the parameters of its neural network? Analyzed through the case of Aumayr's price prediction system.

Furthermore, the results do not show any indication that the sentiment data improves the performance of the models. Additional tests were performed in the model simulation section to further evaluate their impact on the model performances.

5.4.2 Model Simulations

For the final step, we performed multiple simulations with the simulation engine. These simulations were executed for the best performing models from each indicator selection loop, which were defined as the ones with the lowest validation loss value. Additionally,

	Base Model	Optimized Model	Sentiment Model
avg accuracy	0.364	0.368	0.365
avg loss	1.748	1.237	1.263
avg sharpe ratio	1.299	1.378	1.27
avg sortino ratio	2.011	2.135	1.967

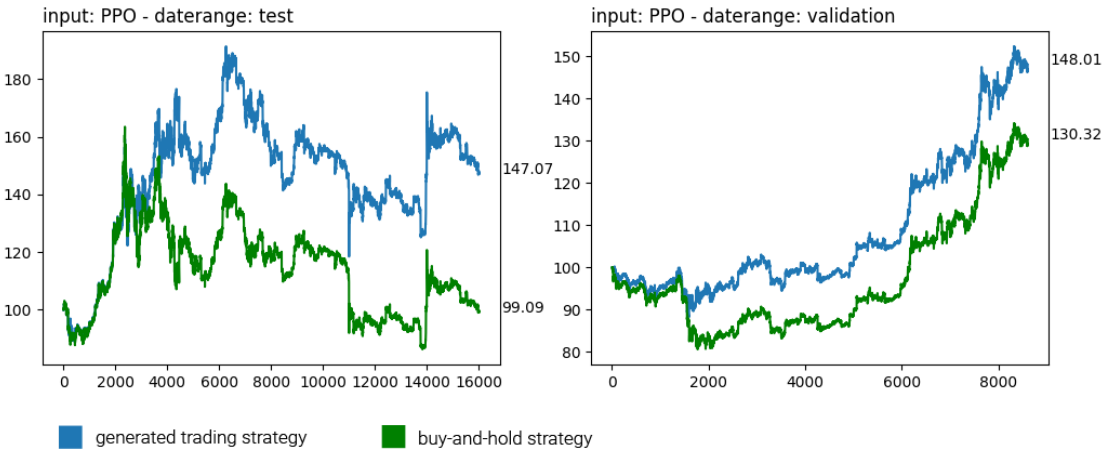
Table 5.20: Average model performance - Base, Optimized, Sentiment

each of the best performing models was retrained with sentiment data as a supplementary input source to further analyze the performance impact of sentiment data. At the end of this section, we generated statistical values for the overall best performing model and tested different simulation settings to reduce the risk of the neural network trading strategy.

The simulation results for the best models of the first two indicator selection loops are displayed in the figures 5.5 and 5.7. Additionally, each of this figures displays the performance of the buy-and-hold strategy for a direct side by side comparison. The results of the same indicator combinations but with sentiment data as an additional input source are presented in the figures 5.6 and 5.8.

As visualized in the charts, sentiment data has no positive impact on the performance of the models. Therefore, the conclusion can be drawn that it is not possible to improve the accuracy of the price prediction algorithm with our sentiment data.

Figure 5.5: PPO indicator simulation for test and validation daterange



The overall best performing input data for the developed market prediction neural network is the single indicator PPO. Table 5.22 displays the statistical values of this model while its simulation is illustrated in figure 5.5. Multiple additional simulations were performed to find proper trading settings for short trading, stop loss, and the maximum trading

Figure 5.6: PPO indicator with sentiment simulation for test and validation daterange

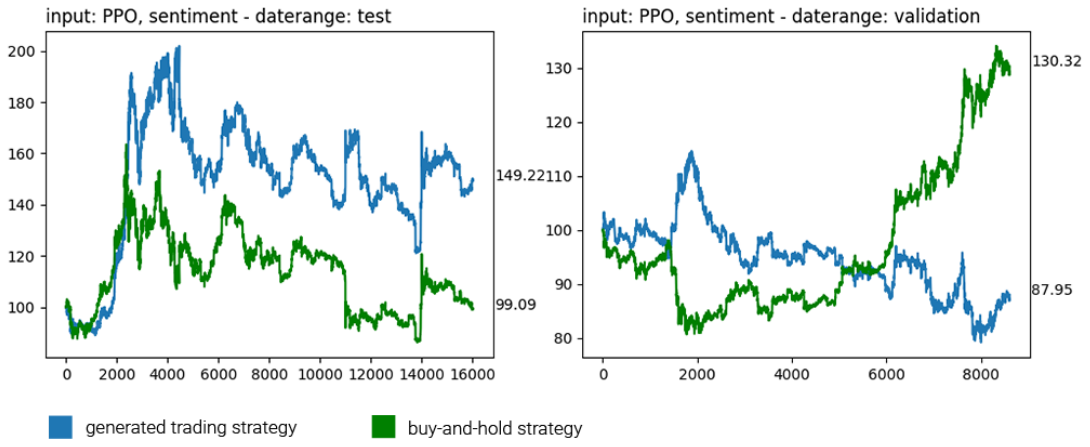
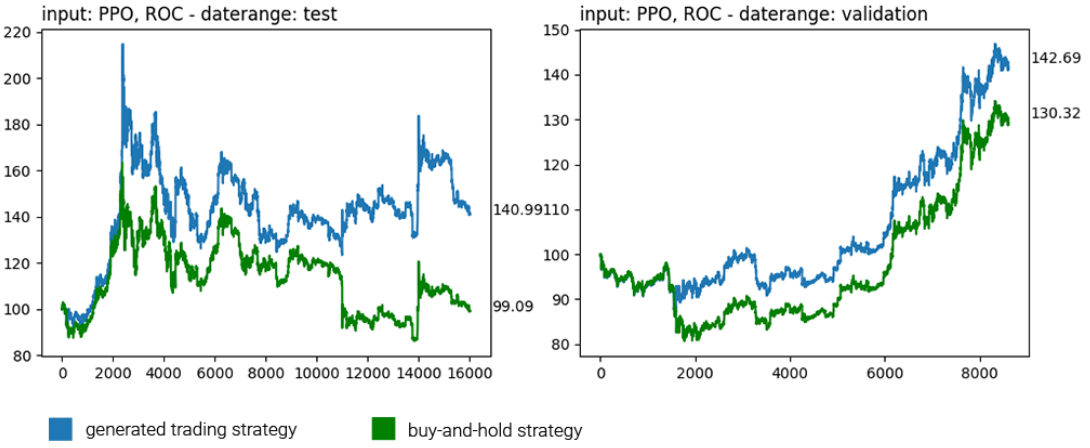


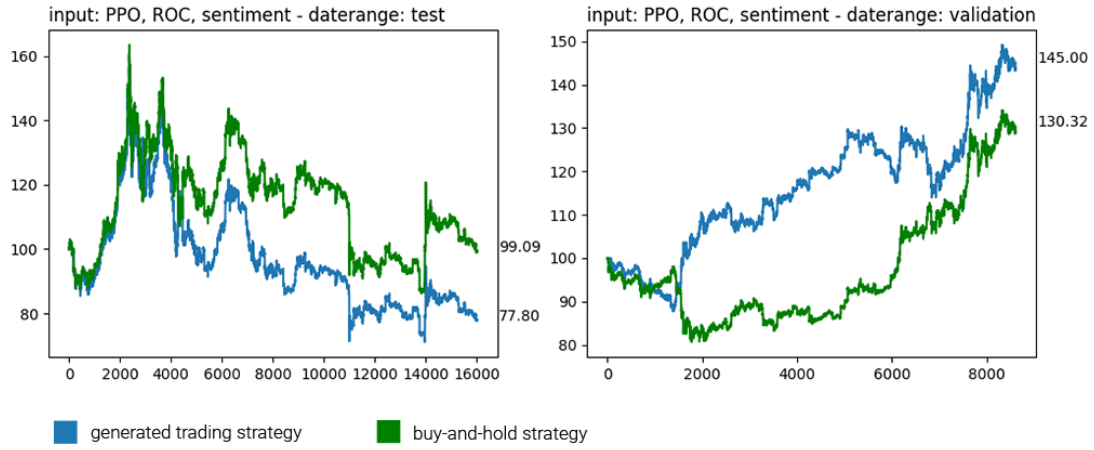
Figure 5.7: PPO + ROC indicator simulation for test and validation daterange



amount as well as to further reduce the risk of the generated trading strategy. To reduce the risk exposure, we identified a configuration with a low maximum drawdown that still retained a reasonably good sortino ratio. The final settings of the low-risk model were the following:

- maximum trade amount: 33% of account value
- short trading: enabled
- stop loss: 2%

Figure 5.8: PPO + ROC indicator with sentiment simulation for test and validation daterange



	test	validation
profit/loss	1%	29.65%
sharpe ratio	0.43	2.72
sortino ratio	0.6	3.52
max drawdown	44.40%	15.50%

Table 5.21: Statistics for buy-and-hold strategy

	test	validation
profit/loss	147%	148%
sharpe ratio	1.46	3.94
sortino ratio	2.15	6.66
max drawdown	33	9
number of trades	43	5
positive trades	27	3
negative trades	15	1

Table 5.22: Statistics for PPO indicator simulation

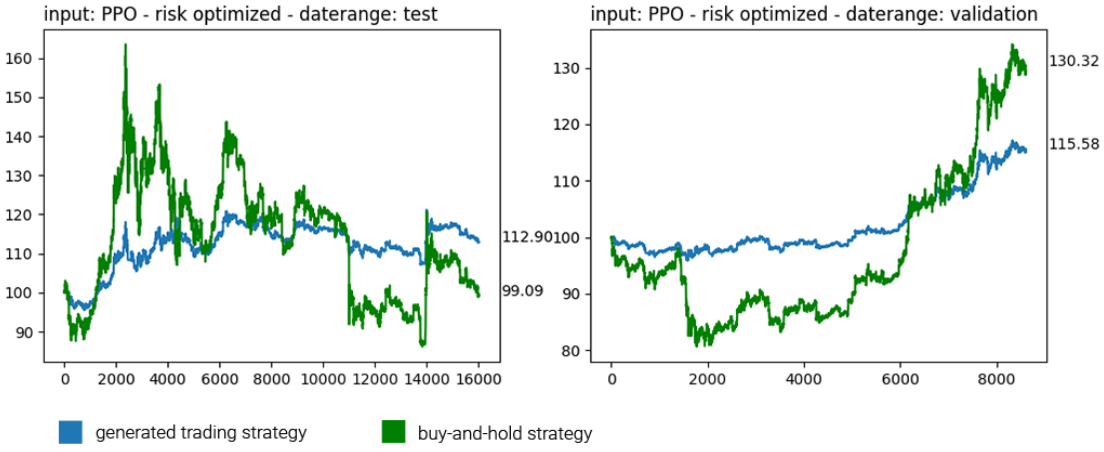
With these settings we achieved a maximum drawdown of 11%, compared to the 33% of the default trade configuration. The statistical values of the model as well as its performance are illustrated in table 5.23 and figure 5.9 respectively.

The results show that the low-risk model is significantly less volatile than both the buy-and-hold trading strategy and the model with the default settings. However, when comparing the figures 5.9 and 5.5 it becomes apparent that these settings also involve the disadvantage of a much lower performance compared to the default trading configuration.

	test	validation
profit/loss	113%	116%
sharpe ratio	1.1	3.66
sortino ratio	1.51	5.76
max drawdown	11	3
number of trades	97	14
positive trades	21	2
negative trades	13	1

Table 5.23: Statistics for PPO indicator simulation with risk optimized settings

Figure 5.9: PPO indicator simulation with risk optimized settings for test and validation daterange



CHAPTER 6

Summary

6.1 Conclusion

This thesis was set out to evaluate the hypothesis that an advanced neural network is able to perform crypto market predictions that are able to outperform a buy-and-hold trading strategy. To test this hypothesis and answer related research questions, we developed a neural network-based prediction engine for the bitcoin price. The system is designed to use price data, technical indicators and sentiment data as input features and can perform the following tasks in a semi-automated way:

- Find a good risk adjusted target value for the neural networks.
- Optimize the structure and internal parameters of a given neural network for the prediction of the bitcoin price.
- Find combinations of different technical indicators with the best predictive ability for the bitcoin price.
- Analyze the sentiment of financial microblogging data from twitter and stocktwits.
- Perform trading simulations and find suitable settings for an appropriate risk level.

In the first step to create this system, we focused on the identification of appropriate target values by evaluating multiple settings for both the *minimum profit target* and the *maximum accepted loss*. Through this set of experiments, we discovered that the best results for the sharpe ratio were achieved with a *minimum profit target* of 3% in combination with a *maximum accepted loss* of 2%.

Next, we tested different neural network structures during the optimization process of the market prediction neural networks. It was found that a combination of a CNN,

LSTM and a Dense layer have the best predictive abilities for a bitcoin price with a 15 minute interval. Specifically, the final neural network settings contain a CNN layer with a filter size of 64, a LSTM layer with 128 inner layers and a Dense layer of size 256. We discovered that networks without a CNN layer have a much lower performance and require a longer training time than the networks which contain a CNN layer, which leads us to the conclusion that the CNN layer is able to successfully extract important details and simplify our used input data. The GRU layer underperformed the LSTM layer which suggests that the more complex LSTM layer can abstract and learn more significant details from our time series input data than the newer simplified GRU layer.

After the structure and internal parameters of the market prediction neural network were optimized, we focused on the selection process of the input features. We selected 62 technical indicators from 5 categories and evaluated the performance of different combinations of these indicators. During these tests, up to 3 indicators were combined and tested simultaneously. The evaluations showed highly promising results with a vast number of indicators and indicator combinations that significantly outperformed the base model and a buy-and-hold trading strategy. The best performing indicator, namely the percentage price oscillator (PPO), outperformed the buy-and-hold strategy by 47% for the test-daterange and 18% for the validation daterange. Additionally, the maximum drawdown of the generated strategy is superior to the buy-and-hold strategy, resulting in a lower downwards risk and a higher upwards potential.

For the next part of the system – the sentiment analysis – we utilized and fine-tuned the most advanced pre-trained nlp transformer models to evaluate financial microblogging data. In the course of our experiments, the roberta models significantly outperformed all other models for the task of sentiment classification. The best performing model was afterwards used for the classification of the twitter and stocktwits messages.

The classified sentiment data was then used as an additional input source for the market prediction neural networks. Comparing the same models with and without the sentiment data, we discovered that adding sentiment data had no beneficial effects but indeed decreased the models' performance. We derived the conclusion that the generated sentiment data contained too much random noise and was therefore not able to improve the prediction results for our neural networks.

In the last part of our experiments, we performed multiple trading simulations for the PPO-indicator model to further reduce the risk of the underlying trading strategy. During this procedure, we evaluated various settings like stop-loss, position size, and short trading. The most promising analyzed configuration of the generated trading strategy allowed a maximum trade amount of 33% of the current account value, enabled short trading, and implemented a stop loss of 2%. Through these settings we were able to reduce the maximum drawdown by 66% but consequently reduced the sharpe ratio by 0.4.

Overall, the underlying experiments of this thesis presented some highly promising results for the process of neural network market predictions with technical indicator combinations. The sentiment model achieved reasonably good results on its own but failed to improve

the performance of market prediction neural networks. Additionally, we presented a set of methods to approach the tradeoff between profit and accepted risk and to reduce the exposure to risk for specific trading strategies.

6.2 Further Work

This thesis offers a practical experiment for the prediction of crypto market prices, which offers for a variety of extensions. On the one hand, we want to invite further researchers to use more and/or different data on any of the applied methods of the system. As an example, more price intervals can be analyzed, more technical indicators added, more indicator combinations used, other cryptocurrencies utilized, or different sources scanned as sentiment data. Most of the technical indicators also provide additional configuration parameters, which can be adjusted to vastly increase the input space of the neural network. The downside of these methods is the substantial amount of computing power required to perform them.

Furthermore, future research possibilities can be approached by adding new features and adjusting existing input features. Switching the price data feed from a USD base to a BTC base, for example, might lead to valuable prediction results as the paper [151] suggests. Other than that, the system we developed is generic enough to be used with any kind of time series input data. Consequently, it can be tested with different asset classes like forex and stocks. Another interesting option is the combination of multiple assets which would allow to test if one asset influences the price of another one. A promising candidate for such an analysis is bitcoin used in combination with any other cryptocurrency since bitcoin is mostly viewed as a leading indicator for bigger movements in the crypto market [152]. Using multiple price intervals as input source is another potential research topic. It would allow the neural network to learn long-term and short-term price movements simultaneously. Lastly, there are many hidden patterns within the cryptocurrency market. Most of the bigger market down movements, for example, might start on a weekend or before the end of the month. Features like weekday or day of the month could be added to test if such anomalies can be identified by the neural network to increase its performance.

Another area for future research is the approach used to identify the best performing models. As an example, the sortino or sharpe ratio could be utilized directly as a loss function. Other papers that already achieved this task and produced promising results include [153] [154].

One last option to build upon our findings is to further extend the structure of the neural networks. For instance, researchers can use an ensemble of multiple neural networks, construct a network with parallel layers, or use advanced layers like bidirectional LSTMs. These methods were demonstrated successfully by other researchers in the stock market as discussed in [155] [156] [157]. Neuroevolution is another interesting approach to find a fitting neural network structure and has been successfully applied in the past: [158], [159].

List of Figures

List of Tables

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[2] "Cryptocurrency list." [online] accessed 15.04.2021, available at `https://coinmarketcap.com/all/views/all/`.

[3] "Ethereum." [online] accessed 15.04.2021, available at `https://www.ethereum.org/`.

[4] "Monero." [online] accessed 15.04.2021, available at `https://www.getmonero.org/`.

[5] "Cryptocurrency key features." [online] accessed 15.04.2021, available at `https://coinut.com/blog/features-cryptocurrency-bitcoin/`.

[6] T. Li, D. Shin, and B. Wang, "Cryptocurrency pump-and-dump schemes," *Available at SSRN*, 2018.

[7] B. G. Malkiel, "The efficient market hypothesis and its critics," *Journal of economic perspectives*, vol. 17, no. 1, pp. 59–82, 2003.

[8] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS one*, vol. 12, no. 7, p. e0180944, 2017.

[9] Aumayr, "Automated prognosis of the development of cryptocurrency prices," Master's thesis, Vienna University of Technology, 2019.

[10] T.-M. Dulău and M. Dulau, "Cryptocurrency – sentiment analysis in social media," *Acta Marisiensis. Seria Technologica*, vol. 16, pp. 1–6, 12 2019.

[11] S. Nasekin and C. Y.-H. Chen, "Deep learning-based cryptocurrency sentiment construction," *Digital Finance*, pp. 1–29, 2020.

[12] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[13] S. Al-Megren, S. Alsalamah, L. Altoaimy, H. Alsalamah, L. Soltanisehat, E. Almutairi, and A. 'Sandy' Pentland, "Blockchain use cases in digital sectors: A review of the literature," pp. 1417–1424, 2018.

[14] J. A. Lybeck, *A global history of the financial crash of 2007–10*. Cambridge University Press, 2011.

[15] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[16] M. Jakobsson and A. Juels, *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*, pp. 258–272. Boston, MA: Springer US, 1999.

[17] A. Goriacheva, N. Jakubenko, O. Pogodina, and D. Silnov, "Anonymization technologies of cryptocurrency transactions as money laundering instrument," *KnE Social Sciences*, pp. 46–53, 2018.

[18] J. Bohr and M. Bashir, "Who uses bitcoin? an exploration of the bitcoin community," in *2014 Twelfth Annual International Conference on Privacy, Security and Trust*, pp. 94–101, 2014.

[19] A. Yelowitz and M. Wilson, "Characteristics of bitcoin users: an analysis of google search data," *Applied Economics Letters*, vol. 22, no. 13, pp. 1030–1036, 2015.

[20] "Tesla buys bitcoin." [online] accessed 15.04.2021, available at `https://www.cnbc.com/2021/02/08/tesla-buys-1point5-billion-in-bitcoin.html`.

[21] "Microstrategy buys bitcoin." https://www.cnbc.com/2021/06/21/microstrategy-owns-over-3-billion-worth-of-bitcoin-after-new-purchase.html.

[22] "Ray dalio owns bitcoin." https://www.coindesk.com/consensus-ray-dalio-i-have-some-bitcoin.

[23] A. F. Cifuentes, "Bitcoin in troubled economies: the potential of cryptocurrencies in argentina and venezuela," *Latin American Law Review*, no. 3, pp. 99–116, 2019.

[24] A. Kliber, P. Marszałek, I. Musiałkowska, and K. Świerczyńska, "Bitcoin: Safe haven, hedge or diversifier? perception of bitcoin in the context of a country's economic situation—a stochastic volatility approach," *Physica A: Statistical Mechanics and its Applications*, vol. 524, pp. 246–257, 2019.

[25] "Wikipedia - pizza day." [online] accessed 15.04.2021, available at `https://en.wikipedia.org/wiki/Pizza_Day`.

[26] "Coinmarketcap - bitcoin price chart." [online] accessed 15.04.2021, available at `https://coinmarketcap.com/currencies/bitcoin/`.

[27] R. J. Teweles and E. S. Bradley, *The stock market*, vol. 64. John Wiley & Sons, 1998.

[28] D. Heremans and A. M. Pacces, "Regulation of banking and financial markets," in *Encyclopedia of law and economics*, Edward Elgar Publishing Limited, 2000.

[29] T. Beck and R. Levine, "Stock markets, banks, and growth: Panel evidence," *Journal of Banking & Finance*, vol. 28, no. 3, pp. 423–442, 2004.

[30] J. J. Murphy and J. J. Murphy, *Technical analysis of the financial markets*. Fishkill, N.Y.: New York Institute of Finance, 1999.

[31] C. Kirkpatrick and J. Dahlquist, *Technical Analysis: The Complete Resource for Financial Market Technicians*. FT Press, first ed., 2006.

[32] "Binance - cryptocurrency exchange." [online] accessed 15.04.2021, available at `https://www.binance.com/en`.

[33] "Coinbase - cryptocurrency exchange." [online] accessed 15.04.2021, available at `https://www.coinbase.com/`.

[34] "Kraken - cryptocurrency exchange." [online] accessed 15.04.2021, available at `https://www.kraken.com/`.

[35] "Bitpanda - cryptocurrency exchange." [online] accessed 15.04.2021, available at `https://www.bitpanda.com/en`.

[36] L. X. Lin, E. Budish, L. W. Cong, Z. He, J. H. Bergquist, M. S. Panesir, J. Kelly, M. Lauer, R. Prinster, S. Zhang, *et al.*, "Deconstructing decentralized exchanges," *Stanford Journal of Blockchain Law & Policy*, vol. 2, 2019.

[37] "Coinpaprika - list of cryptocurrencies." [online] accessed 15.04.2021, available at `https://www.coinparika.com/`.

[38] "Tradingview - chart analysis webservice." [online] accessed 15.04.2021, available at `https://www.tradingview.com/`.

[39] H. Nabilou, "How to regulate bitcoin? Decentralized regulation for a decentralized cryptocurrency," *International Journal of Law and Information Technology*, vol. 27, pp. 266–291, 09 2019.

[40] A. Berentsen and F. Schär, "Stablecoins: The quest for a low-volatility cryptocurrency," *The economics of Fintech and digital currencies*, pp. 65–75, 2019.

[41] "Bybit - margin trading with 100x." [online] accessed 15.04.2021, available at `https://learn.bybit.com/trading/what-is-margin-trading-definitive-guide-to-trading-on-margin/`.

[42] D. Twomey and A. Mann, "Fraud and manipulation within cryptocurrency markets," *Corruption and Fraud in Financial Markets: Malpractice, Misconduct and Manipulation Edited by Alexander, C. and Cumming, D*, pp. 205–250, 2020.

[43] M. La Morgia, A. Mei, F. Sassi, and J. Stefa, "The doge of wall street: Analysis and detection of pump and dump cryptocurrency manipulations," *arXiv preprint arXiv:2105.00733*, 2021.

[44] F. Fang, C. Ventre, M. Basios, H. Kong, L. Kanthan, L. Li, D. Martinez-Regoband, and F. Wu, "Cryptocurrency trading: a comprehensive survey," *arXiv preprint arXiv:2003.11352*, 2020.

[45] N. A. Kyriazis, "A survey on efficiency and profitable trading opportunities in cryptocurrency markets," *Journal of Risk and Financial Management*, vol. 12, no. 2, 2019.

[46] T. K. GOYAL, *The Professional Trader: Art, Science and Psychology of Professional Trading*. FT Press, 2018.

[47] D. J. Jordan and J. D. Diltz, "The profitability of day traders," *Financial Analysts Journal*, vol. 59, no. 6, pp. 85–94, 2003.

[48] B. M. Barber, Y.-T. Lee, Y.-J. Liu, and T. Odean, "Do day traders make money?," *University of California, Berkeley, working paper*, 2005.

[49] "Investopedia - definition trading position." [online] accessed 15.04.2021, available at `https://www.investopedia.com/terms/p/position.asp`.

[50] "Investopedia - position sizing." [online] accessed 15.04.2021, available at `https://www.investopedia.com/terms/p/positionsizing.asp`.

[51] J. L. Kelly Jr, "A new interpretation of information rate," in *The Kelly capital growth investment criterion: theory and practice*, pp. 25–34, World Scientific, 2011.

[52] "Investopedia - buy and hold." [online] accessed 15.04.2021, available at `https://www.investopedia.com/terms/b/buyandhold.asp`.

[53] "The case for low-cost index-fund investing, pdf." [online] accessed 15.04.2021, available at `https://personal.vanguard.com/pdf/ISGIDX.pdf`.

[54] "Investopedia - bull market." [online] accessed 15.04.2021, available at `https://www.investopedia.com/terms/b/bullmarket.asp`.

[55] "Investopedia - bear market." [online] accessed 15.04.2021, available at `https://www.investopedia.com/terms/b/bearmarket.asp`.

[56] W. F. Sharpe, "The sharpe ratio," *Journal of portfolio management*, vol. 21, no. 1, pp. 49–58, 1994.

[57] R. Elliott, *The wave principle*. 1938.

[58] "Fibonacci sequence pattern." https://www.smithsonianmag.com/science-nature/fibonacci-sequence-stock-market-180974487/.

[59] "Investopedia - technical indicator." [online] accessed 15.04.2021, available at https://www.investopedia.com/terms/t/technicalindicator.asp.

[60] "talib, python library." [online] accessed 15.04.2021, available at https://mrjbq7.github.io/ta-lib/.

[61] J. Bollinger, "Using bollinger bands," *Stocks & Commodities*, vol. 10, no. 2, pp. 47–51, 1992.

[62] J. W. Wilder, *New concepts in technical trading systems*. Greensboro, N.C: Trend Research, 1978 - 1978.

[63] "Investopedia - percentage price oscillator (ppo)." [online] accessed 15.04.2021, available at https://www.investopedia.com/terms/p/ppo.asp.

[64] "Stockcharts - balance of power (bop)." [online] accessed 15.04.2021, available at https://school.stockcharts.com/doku.php?id=technical_indicators:balance_of_power.

[65] "Technical analysis of stocks and commodities magazine," *August*, 2001.

[66] J. E. Granville, *Granville's New Key to Stock Market Profits*. 1963.

[67] E. D. Liddy, "Natural language processing," 2001.

[68] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[69] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[70] M. Awad and R. Khanna, *Machine Learning*, pp. 1–18. Berkeley, CA: Apress, 2015.

[71] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[72] "Illustration for neuron calculation." [online] accessed 15.04.2021, available at https://bit.ly/2X0V1GM.

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

[74] 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.

[75] Y. Sugomori, *Java Deep Learning Essentials*. Packt Publishing Ltd, 2016.

[76] M. Bernico, *Deep learning quick reference: useful hacks for training and optimizing deep neural networks with TensorFlow and Keras*. Packt Publishing Ltd, 2018.

[77] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[78] S. Sharma and S. Sharma, "Activation functions in neural networks,"

[79] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[80] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[81] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines.," in *ICML* (J. Fürnkranz and T. Joachims, eds.), pp. 807–814, Omnipress, 2010.

[82] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 2016.

[83] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Proceedings of the 31st international conference on neural information processing systems*, pp. 972–981, 2017.

[84] D. M. Allen, "Mean square error of prediction as a criterion for selecting variables," *Technometrics*, vol. 13, no. 3, pp. 469–475, 1971.

[85] "Learning rate illustration." [online] accessed 15.04.2021, available at `https://morioh.com/p/c13af4aa3fd3`.

[86] "Gradient decent illustration." [online] accessed 15.04.2021, available at `https://coinut.com/blog/features-cryptocurrency-bitcoin/`.

[87] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[88] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[89] "towardsdatascience - rmsprop." [online] accessed 15.04.2021, available at https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[90] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[91] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[92] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," *Advances in neural information processing systems*, pp. 402–408, 2001.

[93] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 1998.

[94] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[95] J. Heaton, *AIFH, volume 3: deep learning and neural networks*. Heaton Research, 2015.

[96] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," *arXiv preprint arXiv:1502.02127*, 2015.

[97] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[98] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 826–834, 1983.

[99] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.

[100] "Cnn - illustration of filter application." [online] accessed 15.04.2021, available at https://anhreynolds.com/blogs/cnn.html.

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

[102] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[103] "Gru & lstm comparison." [online] accessed 15.04.2021, available at https://bit.ly/38UuZat.

[104] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[105] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[106] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[107] J. Mackenzie, R. Benham, M. Petri, J. R. Trippas, J. S. Culpepper, and A. Moffat, "Cc-news-en: A large english news corpus," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 3077–3084, 2020.

[108] "Clone of openai's unreleased webtext dataset." [online] accessed 15.04.2021, available at `https://github.com/jcpeterson/openwebtext`.

[109] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[110] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.

[111] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[112] C. Sang and M. Di Pierro, "Improving trading technical analysis with tensorflow long short-term memory (lstm) neural network," *The Journal of Finance and Data Science*, vol. 5, no. 1, pp. 1–11, 2019.

[113] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018.

[114] M. R. Vargas, B. S. De Lima, and A. G. Evsukoff, "Deep learning for stock market prediction from financial news articles," in *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, pp. 60–65, IEEE, 2017.

[115] D. Araci, "Finbert: Financial sentiment analysis with pre-trained language models," *arXiv preprint arXiv:1908.10063*, 2019.

[116] "twitter." [online] accessed 30.04.2021, available at https://twitter.com.

[117] "stocktwits." [online] accessed 30.04.2021, available at https://stocktwits.com/.

[118] A. Samarawickrama and T. Fernando, "A recurrent neural network approach in predicting daily stock prices an application to the sri lankan stock market," in *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, 2017.

[119] J. Du, Q. Liu, K. Chen, and J. Wang, "Forecasting stock prices in two ways based on lstm neural network," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1083–1086, 2019.

[120] D. Selvamuthu, V. Kumar, and A. Mishra, "Indian stock market prediction using artificial neural networks on tick data," *Financial Innovation*, vol. 5, no. 1, pp. 1–12, 2019.

[121] "scikit-learn - robustscaler." [online] accessed 15.04.2021, available at https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html.

[122] D. Nelson, A. Pereira, and R. de Oliveira, "Stock market's price movement prediction with lstm neural networks," pp. 1419–1426, 05 2017.

[123] W. Long, Z. Lu, and L. Cui, "Deep learning-based feature engineering for stock price movement prediction," *Knowledge-Based Systems*, vol. 164, pp. 163–173, 2019.

[124] F. Barbieri, J. Camacho-Collados, L. Neves, and L. Espinosa-Anke, "Tweeteval: Unified benchmark and comparative evaluation for tweet classification," *arXiv preprint arXiv:2010.12421*, 2020.

[125] P. Malo, A. Sinha, P. Takala, P. Korhonen, and J. Wallenius, "Financialphrasebank-v1.0," 07 2013.

[126] I. centre for Data Analytics, "Semeval-2017 task 5 datasets," 2017.

[127] D. Costa and N. Felix, "Inf-ufg at fiqa 2018 task 1: Predicting sentiments and aspects on financial tweets and news headlines," p. 1967–1971, 04 2018.

[128] D. Kuhlman, *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009.

[129] "tensorflow." [online] accessed 30.04.2021, available at https://www.tensorflow.org/.

[130] "pytorch lightning." [online] accessed 30.04.2021, available at https://www.pytorchlightning.ai/.

[131] "pytorch." [online] accessed 30.04.2021, available at https://pytorch.org/.

[132] "huggingface." [online] accessed 30.04.2021, available at https://huggingface.co/.

[133] "wandb." [online] accessed 30.04.2021, available at https://wandb.ai/.

[134] "matplotlib." [online] accessed 30.04.2021, available at https://matplotlib.org/.

[135] "pandas." [online] accessed 30.04.2021, available at https://pandas.pydata.org/.

[136] "numpy." [online] accessed 30.04.2021, available at https://numpy.org/.

[137] "scikit-learn." [online] accessed 30.04.2021, available at https://scikit-learn.org/.

[138] "snscrape." [online] accessed 30.04.2021, available at https://github.com/JustAnotherArchivist/snscrape.

[139] "requests." [online] accessed 30.04.2021, available at https://docs.python-requests.org/en/master/.

[140] "joblib." [online] accessed 30.04.2021, available at https://joblib.readthedocs.io/en/latest/.

[141] "bash." [online] accessed 30.04.2021, available at https://www.gnu.org/software/bash/.

[142] "unix commands/tools." [online] accessed 30.04.2021, available at https://en.wikipedia.org/wiki/List_of_Unix_commands.

[143] "perl." [online] accessed 30.04.2021, available at https://www.perl.org/.

[144] "binance-rest-api." [online] accessed 30.04.2021, available at https://github.com/binance/binance-spot-api-docs/blob/master/rest-api.md.

[145] X. Man, T. Luo, and J. Lin, "Financial sentiment analysis(fsa): A survey," in *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, pp. 617–622, 2019.

[146] M. G. Sousa, K. Sakiyama, L. d. S. Rodrigues, P. H. Moraes, E. R. Fernandes, and E. T. Matsubara, "Bert for stock market sentiment analysis," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1597–1601, 2019.

[147] P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala, "Good debt or bad debt: Detecting semantic orientations in economic texts," *Journal of the Association for Information Science and Technology*, vol. 65, 2014.

[148] S. Rosenthal, N. Farra, and P. Nakov, "Semeval-2017 task 4: Sentiment analysis in twitter," in *Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)*, pp. 502–518, 2017.

[149] "Fiqa 2018." [online] accessed 15.04.2021, available at `https://sites.google.com/view/fiqa/home`.

[150] "ta-lib." [online] accessed 30.04.2021, available at `https://mrjbq7.github.io/ta-lib/`.

[151] L. Alessandretti, A. ElBahrawy, L. M. Aiello, and A. Baronchelli, "Machine learning the cryptocurrency market," *Available at SSRN 3183792*, 2018.

[152] P. Ciaian, M. Rajcaniova, and d'Artis Kancs, "Virtual relationships: Short- and long-run evidence from bitcoin and altcoin markets," *Journal of International Financial Markets, Institutions and Money*, vol. 52, pp. 173–195, 2018.

[153] G. Tegnér, "Recurrent neural networks for financial asset forecasting," 2018.

[154] B. Lim, S. Zohren, and S. Roberts, "Enhancing time-series momentum strategies using deep neural networks," *The Journal of Financial Data Science*, vol. 1, no. 4, pp. 19–38, 2019.

[155] 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, 2017.

[156] J. Eapen, D. Bein, and A. Verma, "Novel deep learning model with cnn and bi-directional lstm for improved stock market index prediction," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0264–0270, 2019.

[157] K. A. Althelaya, E.-S. M. El-Alfy, and S. Mohammed, "Evaluation of bidirectional lstm for short-and long-term stock market prediction," in *2018 9th International Conference on Information and Communication Systems (ICICS)*, pp. 151–156, 2018.

[158] S. Ji, J. Kim, and H. Im, "A comparative study of bitcoin price prediction using deep learning," *Mathematics*, vol. 7, p. 898, 09 2019.

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