Copenhagen Business School
Department of Finance

Cryptocurrencies: Trading and Return Prediction via Machine Learning Methods

by

Tobias Kurt Stefan Deußer[a]
&
Lars Patrick Hillebrand[b]

Master Thesis in
Advanced Economics and Finance (M.Sc. – Cand.oecon)

Supervisor:	Associate Professor, Rasmus Tangsgaard Varneskov, Ph.D.
Page count:	78
Character count:	153,955

15th of May, 2018

[a]Student number: 107581
[b]Student number: 107939

Abstract

In this thesis we study to which extent modern machine learning algorithms and classical time series methods are capable of predicting the one-day-ahead price trend of the well known Bitcoin cryptocurrency. More specifically, we address this forecasting task by implementing four machine learning classifiers. These include the logistic regression model and three neural network architectures, namely the fully-connected-, GRU-, and convolutional neural network. We compare their results with the prediction performances of the classical ARMA model and the trend following momentum strategy.

Taking into account that an algorithm's forecasting accuracy inevitably depends on the quality of training data, we analyse a carefully selected set of 25 features. These include blockchain related market forces of supply and demand, global macroeconomic and financial development indicators and competing cryptocurrency market price data. Furthermore, we utilise attractiveness measures like Google Trends and Wikipedia Pageviews and additionally use natural language processing to create a Google News feature, which ideally captures the market sentiment concerning Bitcoin.

We find that the best GRU setup achieves a remarkable test accuracy of 62% in predicting the next-day Bitcoin price trend, followed by the convolutional neural network and logistic regression, which attain 59% and 57%, respectively. With a test accuracy of only 52%, the fully-connected neural network and the ARMA model rank last not being able to surpass the simple strategy of going long in Bitcoin. However, all these results are excelled by the computationally cheap and intriguingly simple momentum strategy. It equalises the GRU network's test accuracy of 62%, but achieves better performance with regards to its annualised Sharpe ratio and Jensen's alpha.

Acknowledgement

We would like to thank our thesis advisor, Rasmus Tangsgaard Varneskov, for his guidance during the research and writing process. He provided extensive feedback and kept us on track with precise and accurate comments. Furthermore, we are grateful to all professors and teachers who shared their knowledge with us and inspired us to reach new heights. Our fellow students also merit an acknowledgement. Without them our time spent studying would not have been as joyful and enriching as it was. Last but not least, we are indebted to everyone proof reading this thesis: Andi, Dom, Franz, Hannah, Marius, Max, Sandra, and Sebastian.

[Tobi]: First and foremost, I am thankful for the unwavering support and encouragement from my parents, Conny and Peter. Without them I would have been unable to embark on my educational and academic journey starting in elementary school and ending in a Master of Science in a foreign country, of which the current pinnacle is this thesis. My sister Sandra also deserves a mention here, as she has been an invaluable friend and comrade-in-arms during my life, always offering helpful advice and sharing laughs. Finally, a thank you to all my friends, without them this thesis would have been finished months earlier.

[Lars]: I dedicate this thesis to my parents, Uschi and Egbert, who have always encouraged and unconditionally supported me in my personal and academic decisions. They provided guidance when necessary and handled the relocation of their personal computer lifesaver to a foreign country exceptionally well. Moreover, I am particularly grateful to my beloved girlfriend, Hannah, for

backing me up and not dumping me when I chose to embark on my master studies abroad. Lastly, I owe a thank you to my Copenhagen flatmate and longtime friend, Marius, who I find guilty of fueling my Netflix consumption.

PS: We want to express our gratitude for the existence of the Google Search Engine, without its help our whole studies, including this thesis, would have been clearly a futile task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> "I think that the Internet is going to be one of the major forces for reducing the role of government. The one thing that's missing, but that will soon be developed, is a reliable e-cash, a method whereby on the Internet you can transfer funds from A to B without A knowing B or B knowing A."
>
> — Milton Friedman[1], 1999

1.1 Motivation

It was not until a decade later when in January 2009 Friedman's remarkably accurate prediction became reality. Satoshi Nakamoto, an unknown identity, created Bitcoin, the first decentralised digital payment system and currency that enables anonymous and cryptographically protected transactions among its users. It became the first kind of a new class of digital assets called cryptocurrencies that function as a medium of exchange and render centralised trusted entities like banks and other financial intermediaries redundant. Today, Bitcoin is the most prominent and valuable representative among over thousand other cryptocurrencies which are all powered by an underlying software architecture called blockchain, a decentralised distributed public ledger containing a digital currency's complete transaction history. However, not only restricted

[1]Recipient of the Nobel Memorial Price in Economics in 1976.

to financial records, the blockchain is considered the next breakthrough technology that has the potential to vastly increase the efficiency and security of global supply chains, asset ledgers and decentralised social networking. Since its full capability was realised by a broader public over the past year, it fostered the increased popularity of Bitcoin and generally cryptocurrencies, leading to a dramatic increase in value. Sharply peaking in January 2018, Figure 1.1 shows that the top 100 most valuable cryptocurrencies reached an aggregated market capitalisation of over 600 billion USD, an amount greater than the market value of Alphabet Inc., the parent company of Google. Though, as fast as the cryptocurrencies' values grew they also declined in the period thereafter, indicating the high degree of volatility in the market caused by large scale speculative exchange rate trading.

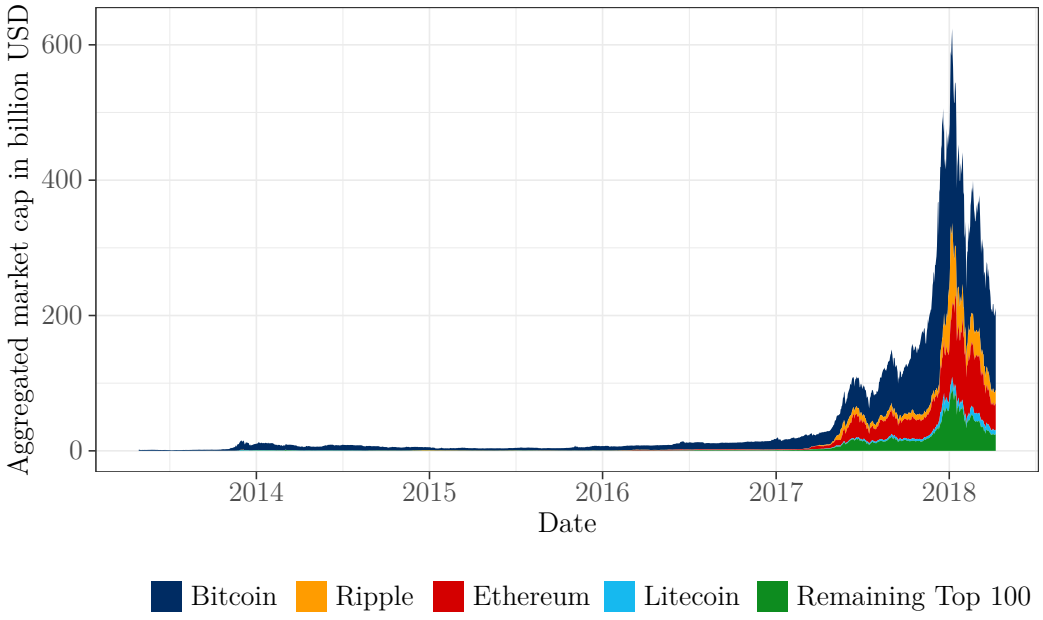

Figure 1.1: Aggregated market capitalisation in billion USD of the top 100 cryptocurrencies (sorted by market capitalisation).

Just like Bitcoin and the blockchain, another buzz word creating an immense media hype over the past few years is artificial intelligence (AI). The

broad term comprises the notion of machine learning, of which deep learning is considered the most promising subset (see Figure 1.2 for an illustration). Chollet and Allaire (2018) describe in their book, *Deep Learning with R*, that

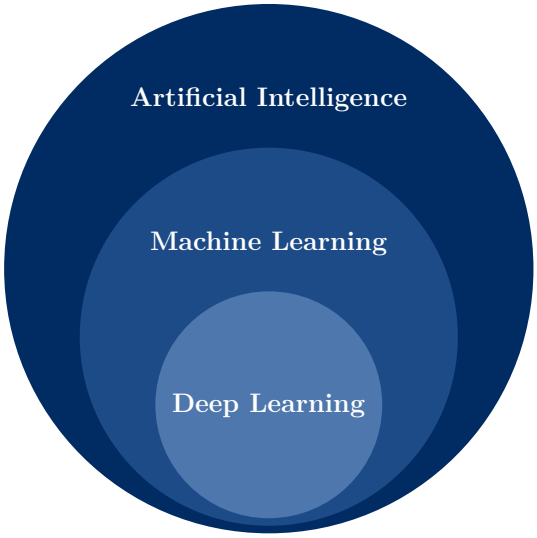

Figure 1.2: Illustrated relation between the terms "artificial intelligence", "machine learning" and "deep learning".

"we're promised a future of intelligent chat bots, self-driving cars, and virtual assistance". Taking into consideration the recent successes in face and image recognition (e.g. Apple's Face ID or medical imaging analysis[2]) and natural language processing (e.g. Amazon's Alexa) this future might be approaching faster than expected.

Being equally fascinated by the blockchain and AI technology plus Bitcoin's highly interesting characteristics as a volatile financial asset, we decided to employ machine learning methods and in particular deep learning to predict the one-day-ahead Bitcoin price trend in this work. We compare the prediction accuracy of different neural network architectures with classical time series models and implement the best performing model within an algorithmic trading framework.

[2] As an example, see Kallenberg et al. (2016), who successfully apply deep learning to mammographic breast cancer diagnosis.

The results presented herein reveal determinants of Bitcoin and explore the possibilities of generating returns uncorrelated to other asset classes. Building on these results one could create strategies beneficial in diversifying non-cryptocurrency portfolios, which might be of interest for various investors like hedge funds or asset managers.

To better understand the technicalities behind Bitcoin and the concept of machine learning, the next two subsections give a short explanation.

1.1.1 Bitcoin and Blockchain

At the core of the Bitcoin payment system[3] lie individual transactions which together form the blockchain, a decentralised public record of all Bitcoin (BTC) transactions ever executed. To execute a transaction each user of Bitcoin owns a key pair, consisting of a public and a corresponding private key, that is stored in his wallet. The public key, also called Bitcoin address and analogous to a bank account, holds all of the user's past transactions, and hence his BTC balance. The record of transactions inside the public key is visible to everyone. However, anonymity is ensured since a connection from the public key to the owner's identity cannot be made. The private key, cryptographically related to the public key and comparable to a bank account's password, is used to sign new transactions and thus, enables access to the owner's BTC balance. Only transactions signed by the correct corresponding private key will be validated by the network. Thereby, it is ensured that the transaction is executed by the rightful owner plus that the owner possesses enough BTC for the transaction.

As a concrete example, assume Alice wants to send 10 BTC to Bob and the past transaction record in her public key states a current balance of 20 BTC. First, Alice needs to sign the transaction. She puts both, her private key and the transaction details (amount of BTC sent, receiver), into her Bitcoin software (cryptographical signing algorithm), that returns a digital signature. Second, this signature gets broadcasted to the Bitcoin network for validation. Specific mining nodes (more on that in the next paragraph) validate the trans-

[3]First described in Nakamoto (2008).

action by running the signature and Alice's public key through a different cryptographical validation algorithm. If the private key, used for signing, indeed corresponds to the public key, the algorithm confirms Alice being the owner of the 10 BTC and not having spend them yet (initial balance of 20 BTC). The cryptographical achievement worth emphasising is, that the validation algorithm validates the transaction without actually knowing the private key. However, the mathematical concept behind this lies beyond the scope of this work.

The final element to fully understand Bitcoin is the blockchain and the concept of mining. Mining nodes are special computers in the Bitcoin network which not only validate transactions, but also gather them after their approval in chronological order into blocks. These blocks are then chained together via a cryptographic hash function (see Appendix A for an explanation) making the individual transactions tamper-proof and forming the blockchain. The blockchain's structure is illustrated in Figure 1.3. A single block consists of two parts, the header and the body. The body contains all transactions gathered inside the block and the header comprises different information used for chaining the next block. Bitcoin uses the proof-of-work system to ensure that on average only every ten minutes a new block is added to the chain. This is achieved by letting mining nodes compete to solve a computationally expensive math puzzle. The winning node adds its block to the blockchain and gets rewarded a certain amount of BTC in return for the work. In detail this proof-of-work process works as follows: to add Block 13 in Figure 1.3 to the chain a mining node puts all of Block 13's header information into the cryptographic hashing function (SHA-256). If the resulting hash value is smaller than a certain threshold the puzzle is considered solved and the block gets added to the chain. To randomise this trial and error process, which is the computationally intensive part of proof-of-work, the mining node iterates over the so called nonce[4], a single number at the end of the hash function input message.

To further secure the blockchain a mechanism called distributed consensus

[4]Number once used.

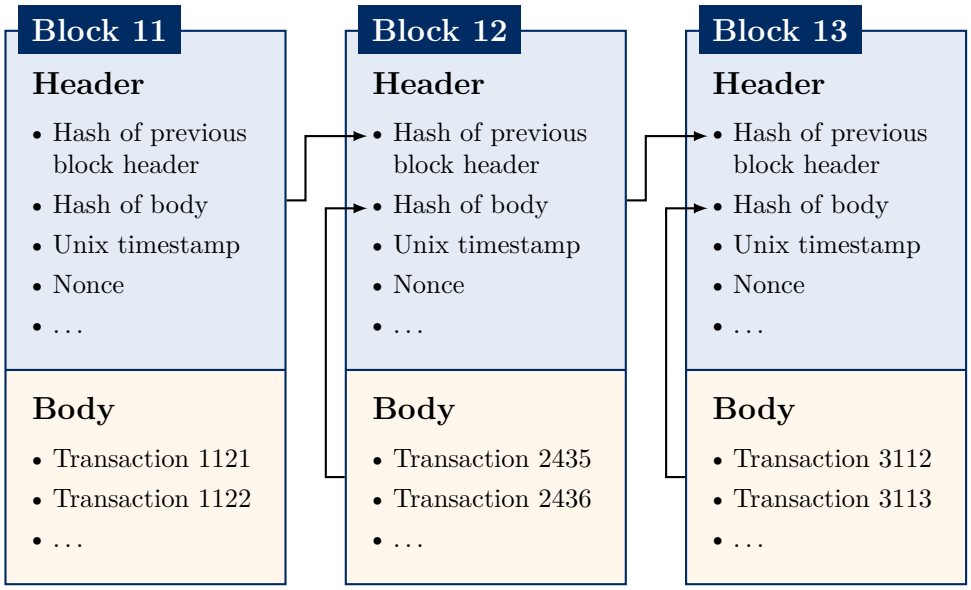

Figure 1.3: Simplified illustration of the blockchain used within the Bitcoin payment system.

verifies proof-of-work. If a mining node broadcasts its solution (hash value) to the network all other nodes verify the solution before the block gets added to the blockchain and synchronised across the system. Together, cryptographic hashing, proof-of-work and distributed consensus ensure that the blockchain is virtually tamper-proof (no altering or manipulation possible), securing all transactions and thus, BTC balances.

1.1.2 Machine Learning

In classical programming, humans insert certain rules (a program) and data to be processed according to these rules and the outcome is answers. In contrast, Turing (1950), while also introducing the famous *Turing test*, reflected on whether a computer could be capable of learning and of originality. His conclusion was, unsurprisingly to us now, that yes, they are able to learn and apply the learned principals to the real world. Machine learning comes directly from this insight: a computer is fed with data plus answers to the data and learns patterns as well as rules so that he is able give answers when encoun-

tering new, unseen data. Thus, a machine learning system is *trained* and not explicitly programmed.

To achieve this learning process, a machine learning algorithm needs three components (names inspired by Chollet and Allaire (2018)):

- **Input data**, often described as $\boldsymbol{x}$. For example, if the algorithm is trained to discover whether or not a patient has the Parkinson's disease, one could look into the speech pattern found in sound recordings of patients (see, amongst others, Tsanas et al. (2012)). In the case of predicting the returns of financial assets one could look and test for a huge amount of explanatory variables. More on that in the Section *Feature Selection*.

- **Examples of the expected output**, the corresponding $\boldsymbol{y}$ to the before mentioned $\boldsymbol{x}$, also called the *label* of each observation. For the case of classifying Parkinson's disease this would be a binary variable describing whether the patient has the disease or not. For the prediction of returns of a financial asset this can either be the actual return or a binary variable describing whether the return was positive or negative.

- **A way to measure if the algorithm is doing a good job**, also described as the *loss measure*. This indicates how far off the algorithm is in predicting the expected output with the current prediction. This measurement is used as a feedback signal to make adjustments to the procedure with which the machine learning algorithm works. This adjustment is why the system is able to learn from the data. A typical loss measure for a classification task is cross-entropy (for an explanation see Chapter 3, Subsection *Fully-Connected Neural Network*) and for regression tasks the mean squared error.

The process of learning can also be seen as a search for better and more useful representations for a given task of the data at hand. This process can involve, but is not limited to, coordinate changes, linear projections, translations, and nonlinear operations. Generally, machine learning methods are not creative in

finding these transformations. All they do is search through a predefined set of operations, the so called *hypothesis space* $\mathcal{H}$. For example, a hypothesis space of a neural network consists of all possible combinations of weights for each neuron (more on neural networks later). Usually, a machine learning algorithm will stop either after a predefined number of adjustments or when a certain threshold for the loss measure is reached.

1.2 Related Literature

Even though Bitcoin is a relatively new asset, researchers have spent a considerable amount of time on how to predict its return and understand its determinants. Looking into the latter, Garcia et al. (2014) hypothesised that fluctuations in the Bitcoin price are driven by the interplay between different social phenomena like volume of word-of-mouth communication in social media, volume of information search on Google and user base growth. Using these factors and lagged prices, they constructed a vector autoregression to reproduce the price development. A similar strategy was applied in Matta et al. (2015), who analysed 1,924,891 tweets combined with Google Trends. As a conclusion, they confirmed the findings in Garcia et al. (2014) that social media and volume of searches may help predicting movements in Bitcoin prices.

However, trying to predict the Bitcoin return explicitly with machine learning approaches has been looked into much more thoroughly. This might be due to the fact that both, machine learning and Bitcoin, originate in the field of computer science or that both are increasingly popular these days. One of the first researches into this topic was conducted by Shah and Zhang (2014), who applied a Bayesian regression to data collected before the 6^{th} of May 2014 and obtained a return of 89% in 50 days with a Sharpe ratio of 4.1. Madan et al. (2015) achieved a Bitcoin trend accuracy of $50\% - 55\%$ with an algorithm using both a random forest and a generalised least squares model. In the same year, Greaves and Au (2015) compared the accuracy of forecasting the direction of Bitcoin returns of a logistic regression, support vector machine, and neural

network and found the neural network to perform the best with an accuracy of 55.1%. Kim et al. (2016) analysed user comments in online cryptocurrency communities and were able to predict the 6 day ahead Bitcoin price with a staggering accuracy of 79.6%, but their model proved inadequate when smaller or larger time horizons were considered. In a thesis similar to this one, McNally (2016) looked into the predictive power of recurrent neural networks when applied to the Bitcoin price. He found that a long short-term memory model is better suited for this task than a standard recurrent neural network, as they had an accuracy of 52.8% and 50.3%, respectively. Amjad and Shah (2017) used random forest, logistic regression, and linear discriminant analysis models on Bitcoin data quoted in Chinese Yuans and realised an accuracy close to 80%. In a comparison between a long short term memory model, a gated recurrent unit model, and an ARIMA with dynamic regression, Torres and Qiu (2018) found no considerable difference between these approaches, but observed that the gated recurrent unit model takes significantly less time to train. Finally, Jang and Lee (2018) exploited data similar to the one we used by training a Bayesian neural network and reported a root mean squared error of 0.0069 on their test set.

The phenomenon of consistent positive returns of trend following strategies[5] is much less studied in cryptocurrencies than in traditional asset classes, which might be due to the relatively tender age and the still small market capitalisation. Nevertheless, Garcia and Schweitzer (2015) have shown positive results for a momentum strategy on Bitcoin data ranging from February 2011 to December 2014. Hong (2017) also found persistence in returns for one to eight weeks for time series momentum in Bitcoin and credited this shorter time span compared to traditional asset classes to the "much quicker nature and shorter term memory of Bitcoin investors" (Hong (2017), p. 265). Rohrbach et al. (2017) compared momentum strategies across various currencies, including cryptocurrencies, and found while it is not profitable anymore for G10 currencies, it generated a return of up to 56.94% p.a. and had a Sharpe ratio

[5]For an in-depth explanation on trend following strategies see Chapter 3, Subsection *Trend Following Strategy (Momentum).*

of 1.68 for a cryptocurrency portfolio. However, the backtest suffered from a small period of 18 months.

In this thesis we study and compare both approaches, which to our knowledge has not been done yet. On the one hand, the more *classical* approach, where we try to predict the one-day-ahead Bitcoin trend using a momentum strategy as well as an ARMA model. On the other hand, we apply various neural network configurations, that attempt to learn patterns helpful in forecasting the previously mentioned Bitcoin trend. Furthermore, we employ a carefully selected set of predictors, including, among others, various blockchain-, cryptocurrency-, and macroeconomic data, plus news data generated via natural language processing. After extensive research into the topic, we came to the conclusion that such a thorough study of possible predictors and feature combinations for Bitcoin trend forecasting has not been attempted before.

In the following our work is structured in four parts. The next chapter presents the data and preprocessing steps used in the forecasting task. Chapter 3 explains the theory and methodological approach of our analysis. Chapter 4 displays and reviews results and Chapter 5 discusses limitations and concludes.

Chapter 2

Data

In our analysis we employ a broad range of features from different categories and data sources which we assume to have predictive power with regards to the one-day-ahead Bitcoin price. The next sections explicitly explain these features and the underlying data retrieval process. Furthermore, we provide details on the various preprocessing steps conducted, including natural language processing, and present descriptive statistics for selected features.

2.1 Feature Selection

As a new digital financial asset, the price determinants of Bitcoin are subject to intensive academic research. Ciaian et al. (2016) divide potential price drivers in three categories: blockchain related market forces of supply and demand, global macroeconomic and financial development indicators and attractiveness indicators measuring the public interest and hype around Bitcoin. In our analysis we follow Kristoufek (2015) and leverage an extensive set of features covering all three categories. To further capture relationships and correlations within the cryptocurrency market we also include market price information from the three largest competitors of Bitcoin, namely Ether, Ripple and Litecoin. Hence, along with the Bitcoin price data itself our feature set comprises five categories containing a total of 26 features. Table 2.1 offers a compact and structured overview of these features, their category, definition and data source.

Category	Feature	Description	Source
Bitcoin price	`btc trend`	Binary feature indicating the daily Bitcoin price trend, 1 equals an "up" move and 0 a "down" move.	Self engineered
	`btc close price`	Daily Bitcoin closing price in USD, taken from Bitstamp exchange.	Quandl
	`btc open price`	Daily Bitcoin opening price in USD, taken from Bitstamp exchange.	Quandl
	`btc high price`	Daily Bitcoin high price in USD, taken from Bitstamp exchange.	Quandl
	`btc low price`	Daily Bitcoin low price in USD, taken from Bitstamp exchange.	Quandl
	`btc volume`	Daily volume of traded Bitcoin at Bitstamp exchange (in BTC).	Quandl
Bitcoin blockchain	`btc addresses`	Daily number of unique Bitcoin addresses used.	Quandl
	`btc hash rate`	Estimated daily number of giga hashes per second (in billion) the Bitcoin network is performing for proof-of-work.	Quandl
	`btc miner revenue`	(Number of Bitcoins mined per day + transaction fees) · market price in USD.	Quandl
	`btc cost per tx`	Miners revenue divided by number of BTC transactions (daily in USD).	Quandl
	`btc block size`	Average daily block size in megabyte.	Quandl
	`btc tx volume`	Estimated total output volume of all Bitcoin transactions per day without value change (in BTC).	Quandl
	`btc number tx`	Total number of unique Bitcoin transactions per day.	Quandl
	`btc tx conf time`	Daily median time for a transaction to get validated and added to a block.	Quandl
Other cryptocurrencies	`eth close price`	Daily Ether closing price in USD, taken from CCCAGG (CryptoCompare Current Aggregate) index[1].	CryptoCompare
	`xrp close price`	Daily Ripple closing price in USD, taken from CCCAGG index.	CryptoCompare
	`ltc close price`	Daily Litecoin closing price in USD, taken from CCCAGG index.	CryptoCompare
Macroeconomy	`fed funds rate`	Daily federal funds rate in %.	FRED[2]

	`gold price`	Daily gold price in USD per troy ounce.	Quandl
	`msci price`	Daily MSCI World Index closing price in USD.	Onvista
	`usd/eur rate`	Daily USD/EUR closing exchange rate (quoted in EUR).	Yahoo Finance
	`vix`	Daily closing value of the CBOE[3] Volatility Index (VIX)[4].	Yahoo Finance
News and momentum	`google trends`	Daily Google Trends statistic of the keyword "Bitcoin".	Google
	`wiki views`	Daily pageviews statistic of the "Bitcoin" article on Wikipedia.	Wikipedia
	`google news`	Daily Google News related to "Bitcoin". Detailed description on feature collection and processing is presented in 2.2.	Google
	`momentum`	Binary feature computed from the aggregated BTC log return over the past 14 days.	Self engineered

Table 2.1: Data description and -sources.

In the following paragraphs we specifically explain the economic idea and data retrieval process behind each category and the corresponding features mentioned in Table 2.1. For the whole empirical analysis including data collection, preprocessing, modeling, forecasting and evaluation we employ the `R` programming language developed by the R Core Team (2017) and use the `R` packages `ggplot2`[5] and `xtable`[6] to generate empirical figures and tables. Further, we restrict our study to a period of about five years and four month, ranging from the 1st of January, 2013 to the 24th of April, 2018.

The first category in Table 2.1, "Bitcoin price", contains the binary dependent variable of interest we attempt to predict, the daily BTC price trend, which

[1]CCCAGG is a daily 24 hour volume weighted average aggregating transaction data from over 70 exchanges. It is calculated for each crypto coin in each currency it is trading in (here: CCCAGG BTC/USD). For details, see https://www.cryptocompare.com/media/12318004/cccagg.pdf, retrieved on 09/05/2018.
[2]Federal Reserve Economic Data.
[3]Chicago Board Options Exchange.
[4]Implied volatility index calculated from S&P 500 index options.
[5]Wickham (2009).
[6]Dahl (2016).

is calculated from the daily BTC closing price in USD as

$$\texttt{btc trend}_t = \begin{cases} 1, & \text{if } \texttt{btc close price}_t \geq \texttt{btc close price}_{t-1}, \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, we select classical trading features adding more granularity and information to the BTC time series. In particular these features are the daily BTC open-, high- and low- price in USD as well as the daily volume of traded BTC (in BTC). All price and trade data is taken from the Bitstamp exchange and accessed via Quandl.com leveraging their open API (Application Programming Interface) using the `Quandl R` package from Raymond McTaggart et al. (2016).

The second category comprises Bitcoin blockchain data, also accessed via Quandl. Specifically, we selected eight technical features from the blockchain network capturing forces of supply and demand within the Bitcoin payment system. On the demand side `btc addresses`, `btc number tx` and `btc tx volume` proxy the size and usage of the Bitcoin economy. Since Bitcoin is primarily utilised for transactions and exchange rate trading, let us emphasise the difference between `btc volume` from the previous category and `btc tx volume`. The former represents the BTC volume generated by exchange rate trading and the latter the effective transaction volume used for purchases. On the supply side the Bitcoin hash rate measures the mining productivity assuring a balanced and deterministic BTC supply. In addition, the miners revenue and average daily block size indicate the profitability of the network, which is closely related to the incentive to mine and hence, to the supply of Bitcoin. Lastly, `btc cost per tx` and `btc tx conf time` proxy the practical and monetary expenses a Bitcoin user faces during a transaction, which is expected to correlate negatively with the demand.

The third category aims to capture correlations within the cryptocurrency market. It holds the closing prices in USD from the three biggest[7] Bitcoin competitors, namely Ether (ETH), Ripple (XRP) and Litecoin (LTC). The

[7] By market capitalisation.

time series are taken from cryptocompare.com utilising their public API for historical coin data.

The fourth category includes macroeconomic variables reflecting, amongst others, the global price level, industrial development and degree of financial distress. Because the European Union (EU) is the largest trade partner of the US, we chose the USD/EUR exchange rate (base currency USD, quoted in EUR), extracted from Yahoo Finance via the `quantmod R` package[8], to control for price level changes in the US relative to the EU. As our Bitcoin price data is denominated in USD it likely is affected by USD value changes relative to other currencies like euro. As an example, assume that the USD appreciates against the euro, it most likely also appreciates against the BTC. Hence, an increase in the USD/EUR exchange rate decreases the BTC price denoted in USD, since one BTC can be bought for a smaller USD amount. Moreover, we selected the MSCI world price index, taken from onvista.de, to proxy the global industrial development. It is a broad equity index representing the "large and mid-cap equity performance across 23 developed market countries"[9]. The gold price and VIX index attempt to capture potential BTC *safe haven*/commodity characteristics and its exposure to financial distress, respectively. Finally, we also added the fed funds rate, taken from FRED, as an indicator for US economic growth, controlling inflation and thereby, the price level.

The last category contains four indicators quantifying the public interest and momentum in Bitcoin on a daily frequency. Following Kristoufek (2013), we employ the Google Trends and Wikipedia pageviews statistics to proxy the daily amount of Bitcoin search queries on Google and to measure the daily pageviews quantity of the Wikipedia Bitcoin article, respectively. Since the 1st of July, 2015 Wikipedia provides their pageviews statistics via a new API. Unfortunately, structured and easily searchable statistics from before that date are no longer available. We accessed the API via the `pageviews R` package provided by Keyes and Lewis (2016). The data for the `google trends` time series was collected via the `gtrendsR R` package from Massicotte and Eddel-

[8]Ryan and Ulrich (2017a).
[9]See https://msci.com/world for more details.

buettel (2018). Note that Google normalises its Google Trends statistics over each requested time period to a maximum value of 100. As the longest period for a single API call is limited to half a year, we combined the different periods by scaling them according to their values from overlapping dates. Additionally, we manually engineered a Google News and momentum feature. The momentum feature is defined as a binary variable, computed from the aggregated BTC closing log returns[10] over the past 14 days. More specifically,

$$\texttt{momentum}_t = \begin{cases} 1, & \text{if } \sum_{i=1}^{T} \texttt{btc close return}_{t-i} \geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

where $T = 14$. The economic intuition behind the concept of trend following and time series momentum is extensively described in Chapter 3 in the *Trend Following Strategy (Momentum)* Subsection.

To mine raw Google News headlines and articles, we coded an automatic web scraper in `R` utilising, amongst others, the `rvest` and `jsonlite` `R` packages from Wickham (2016) and Ooms (2014). For each date in the analysed time period, the scraper mined headlines, articles, and website names from the ten most prominent "Bitcoin" search results on Google News. The cleaning and preprocessing of this raw text data is explained in detail in the next section, covering all preprocessing steps.

2.2 Data Preprocessing

2.2.1 General Preprocessing

Before turning to natural language processing, this subsection presents the general data cleaning and preprocessing steps, equally applied to all features. Alongside, we also introduce the notion of stationarity.

After merging the different features into a single multivariate time series

[10] See the *General Preprocessing* Subsection for details regarding the creation of log returns.

object[11] the first task consists of dealing with missing values (`NA`s). To preserve the full sample size we zero-impute missing values at the beginning of a time series (e.g. for younger features like `eth-`, `xrp-`, and `ltc close price` or the `wiki views` statistics) and linearly interpolate `NA`s enclosed by existing observations. The latter is especially relevant for financial features which only trade during the week, like `msci price`, `gold price`, `usd/eur rate` and `vix`.

Prior to the next preprocessing step, we introduce the statistical concept of (weak) stationarity. A time series y_t is said to be weakly stationary, if "its expected value and population variance are independent of time and if the population covariance between its values at time t and $t+s$ depends on s but not on time" (see Dougherty (2002), p.365). More specifically, weak stationarity is established if $\mathbb{E}[y_t] = \mu$, which is a constant, and $\text{Cov}[y_t, y_{t+s}] = \gamma_s$, which only depends on s, but is invariant to time shifts. In practice, this property is required to linearly model time series data via classical methods like ARMA (see Chapter 3, Section 3.1.2 for a detailed explanation). Further, it ensures the meaningfulness of sample statistics like mean, variance and correlation, which are then valid descriptors of unknown future observations, due to their time invariance. Lastly, stationarity eliminates trends and thereby the problem of spurious causality. Assume a linear regression, where Bob's body weight from birth to his 18th year of life is regressed on Alice's body weight over the same period. Both time series exhibit a positive trend, and thus, are highly correlated. However, the regression outcome of a significant causal relationship between Alice's and Bob's body weight is clearly wrong and misleading.

A common method to render a time series weakly stationary implies taking the ith order difference, where in most cases the first difference is sufficient. Because many of our features are financial asset prices we additionally take the natural logarithm prior to differencing. This facilitates the interpretation of the transformed features because they now represent log returns. Formally, the log difference transformation can be written as

$$\Delta \ln[y_t] = \ln[y_t] - \ln[y_{t-1}]. \tag{2.1}$$

[11] xts object, leveraging the `xts` `R` package from Ryan and Ulrich (2017b).

For features holding zero values we apply a slight modification of (2.1), namely

$$\Delta \ln[y_t + 1] = \ln[y_t + 1] - \ln[y_{t-1} + 1].$$

To verify whether our log differenced time series are truly stationary we employ the Augmented Dickey-Fuller unit root test developed by Dickey and Fuller (1979) and made available from the `vars R` package[12]. Its null hypothesis, H_0, implies the existence of a unit root and thus, non-stationarity. We first conduct the test for all features with an included trend component. If its coefficient for a certain feature is insignificant, we drop the trend and re-test. Table 2.2 shows the test results. For the whole dataset it can be seen that the test statistic in absolute terms is strictly greater than its critical value at the 1% significance level. Hence, the null hypothesis of a unit root can be rejected at the 99% confidence level, which implies all features being weakly stationary.

During the last preprocessing step, we normalise the input features to zero mean and unit variance, called z-score normalisation. Formally, this can be expressed as

$$z_t = \frac{y_t - \mu}{\sigma},$$

where μ and σ depict the time invariant population mean and standard deviation. z-score normalisation assures that the input data is centered and bounded within a sufficiently small interval around zero, which eliminates the vanishing gradient problem during the training and weight updating of neural networks via backpropagation. In short, too large or small input values would cause the gradient (first derivative) of certain activation functions (e.g. the sigmoid function) to converge to zero, which prevents effective weight updating and thereby, learning. Backpropagation and neural network learning is specifically explained in Chapter 3 in the *Fully-Connected Neural Network* Subsection. It is important to note that we calculate the normalisation statistics, μ and σ, based on only the training data. Hence, we implement the

[12] Pfaff (2008).

Feature (log difference)	Test statistic	Critical value ($p < 0.01$)
`btc close return`	−12.90	−3.43
`btc open return`	−7.55	−3.43
`btc high return`	−7.76	−3.43
`btc low return`	−7.56	−3.43
`btc volume`	−18.34	−3.43
`btc addresses`	−12.48	−3.43
`btc hash rate`	−7.34	−3.43
`btc miner revenue`	−8.27	−3.96
`btc cost per tx`	−9.08	−3.43
`btc block size`	−9.02	−3.96
`btc tx volume`	−7.89	−3.43
`btc number tx`	−28.61	−3.43
`btc tx conf time`	−7.14	−3.43
`eth close return`	−10.25	−3.43
`xrp close return`	−11.14	−3.43
`ltc close return`	−9.74	−3.43
`fed funds rate`	−14.37	−3.43
`gold return`	−8.27	−3.96
`usd/eur rate`	−6.93	−3.96
`msci return`	−18.86	−3.43
`vix`	−26.67	−3.43
`google trends`	−14.94	−3.43
`wiki views`	−24.59	−3.43

Table 2.2: Results from the Augmented Dickey-Fuller unit root test.

z-score normalisation within the expanding window validation, which is part of the model selection process described in detail in Chapter 3, Section 3.2.1.

2.2.2 Natural Language Processing

Natural Language Processing (NLP) describes the non-trivial task of processing human language data (e.g. text) such that a computer is able to understand and interpret its content. Since every software and machine learning algorithm requires numerical inputs at the computational level, the main difficulty lays within the conversion of raw text data into integer values, without losing its semantics. To conduct this text vectorisation a range of different methods has been proposed over the past years. One of the most simple but still popular approaches is the Bag-of-Words (BoW) model, having its early roots in a linguistic article by Harris (1954). The basic idea of Bag-of-Words is to create a tokenised unsorted dictionary, holding all unique words from a given text corpus, which might comprise several documents or sentences. As an example, let us assume our corpus only consists of the following two news headlines:

1. "Bitcoin hits record high".
2. "Bitcoin is doomed to fail".

Hence, the dictionary is made of eight unique words, tokenised as "Bitcoin" $\rightarrow$ 1, "hits" $\rightarrow$ 2, "record" $\rightarrow$ 3, ..., "to" $\rightarrow$ 7 and "fail" $\rightarrow$ 8. Given this dictionary, the simplest way of vectorising the headlines is via binary encoding, which is visualised in Table 2.3.

Headline	Tokenised dictionary							
	1	2	3	4	5	6	7	8
"Bitcoin hits record high"	[1	1	1	1	0	0	0	0]
"Bitcoin is doomed to fail"	[1	0	0	0	1	1	1	1]

Table 2.3: Simple document vectorisation via Bag-of-Word and binary encoding.

However, this method suffers from several shortcomings. First, the dimension of the resulting vectors increases one to one with the dictionary size, leading quickly to high dimensional vectors with more than 10,000 entries.

Second, most of these entries are filled with zeros, which is why the vectors are considered sparse and computational and informational inefficient. Third, BoW discards word syntax and thereby, loses important context information.

The recent developments of powerful word embedding models like *word2vec* from Mikolov et al. (2013) and *GloVe* (Global Vectors) from Pennington et al. (2014) solve these limitations. In this work we particularly employ the GloVe algorithm to generate low dimensional word and news headline vectors of fixed size which keep their semantics and contextual meaning. Before turning to the theory and implementation of GloVe, let us briefly describe the preprocessing of the raw Google News headlines. Since raw text is messy and contains many negligible information with regards to meaning and sentiment, its cleaning is highly important for the performance of text vectorisation models like GloVe. Utilising the `textclean`[13], `tm`[14] and `textstem`[15] `R` packages, we restrict our headlines to the standard US ASCII character set and convert all words to lower case. Furthermore, we replace contractions such as "hasn't" or "can't" with their full form, namely "has not" and "cannot". Next, we remove punctuation and frequent stop words like pronouns, prepositions and other words not adding any contextual meaning. Lastly, we group inflected word forms together to a single base form via lemmatisation (e.g. convert "running" and "ran" into their common lemma, "run"). The following before-after comparison of a sample news headline illustrates the conducted cleaning:

Before: "Hedge Funds Push the Price of Bitcoin to New Highs".

After: "hedge fund push price bitcoin new high".

After cleaning the raw headlines we implement the GloVe algorithm via the `text2vec` `R` package, developed by Selivanov and Wang (2018). The first ingredient of GloVe is the creation of a word co-occurrence matrix, denoted by X. Each element, X_{ij}, of this matrix tabulates how often word i occurs in the context of word j. The context of word j is defined as a fixed size

[13]Rinker (2018).
[14]Feinerer et al. (2008).
[15]Rinker (2017).

symmetrical window which comprises k words left and k words right from j. In our implementation we set the context window size to $k = 5$. The dimensionality of X equals the dictionary size of the text corpus, denoted as V. Hence, matrix X is generated by sliding the context window over every word j in the corpus and counting how often word i appears within that window. To account for the fact that more distant word pairs likely share a weaker contextual relationship, each individual occurrence count of word i is weighted by $\frac{1}{d}$, where d denotes the amount of words i and j are apart.

For illustration purposes Table 2.4 shows the word co-occurrence matrix of the example text corpus used in Table 2.3 (dictionary size of $V = 8$). For simplification the context window size k is set to 1, and thus no distance dependent weighting takes place ($d = 1$).

	"Bitcoin"	"hits"	"record"	"high"	"is"	"doomed"	"to"	"fail"
"Bitcoin"	0	1	0	0	1	0	0	0
"hits"	1	0	1	0	0	0	0	0
"record"	0	1	0	1	0	0	0	0
"high"	0	0	1	0	0	0	0	0
"is"	1	0	0	0	0	1	0	0
"doomed"	0	0	0	0	1	0	1	0
"to"	0	0	0	0	0	1	0	1
"fail"	0	0	0	0	0	0	1	0

Table 2.4: Illustration of word co-occurrence matrix based on the text corpus of Table 2.3.

The GloVe authors utilise the above described word co-occurrence matrix to develop the following cost function

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right) \left(\boldsymbol{w}_i^T \tilde{\boldsymbol{w}}_j + b_i + \tilde{b}_j - \ln[X_{ij}]\right)^2,$$

where $\boldsymbol{w}_i$ and $\tilde{\boldsymbol{w}}_j$ represent the desired vectors of main word i and context word j and b_i and $\tilde{b}_j$ are additional scalar bias terms of both words. Lastly, $f(\cdot)$ is a weighting function, preventing extremely common word pairs from

receiving too much weight, which would hinder learning from less frequent pairs. The cost function, J, is iteratively minimised by updating both weight vectors, randomly initialised in the beginning, using Adaptive Gradient Descent[16] (Adagrad). The authors find that summing up the computed main and context weight vectors yields the best result with regards to capturing word semantics. Hence, we define the final word vectors as

$$\boldsymbol{w}^{\star} = \boldsymbol{w} + \tilde{\boldsymbol{w}}.$$

Also note that the dimension of $\boldsymbol{w}^{\star}$ can freely be chosen at the beginning of model training. In our GloVe implementation we train 50-dimensional word vectors for all 12,567 unique words contained in the created Google News headline dictionary. Table 2.5 shows in an exemplary manner the successful preservation of semantics within the computed word vectors. Unarguably, the

	Cosine similarity to "blockchain" word vector
technology	0.86
ledger	0.84
distribute	0.78
decentralize	0.74
digital	0.73

Table 2.5: Five closest word vectors to the "blockchain" word vector based on cosine similarity.

five closest words to "blockchain" measured by cosine similarity are indeed semantically related to it.

The last step to incorporate the Bitcoin related Google News headlines as an input feature for a machine learning algorithm is the averaging of the headlines' individual word vectors. By doing so we create so called document- or in our case headline vectors with the same dimension of the original word vectors.

[16]Similar to ADAM, which is in detail described in Chapter 3, Subsection 3.1.1.

Since we scraped ten headlines per day, we end up with a 500-dimensional Google News input feature ($10 \cdot 50$-dimensional headline vectors).

2.3 Descriptive Statistics

This section shows the descriptive statistics of the data we are using in our analysis. Table 2.6 illustrates the mean, standard deviation, minimum, and maximum of all features which were previously transformed into log differences. One observation is the difference between `btc low return` and the other Bitcoin price variables. The severe volatility of this feature is caused by huge intraday price drops and can also be observed in the extreme values for the minimum and maximum.

The generally high volatility in Bitcoin can further be seen in Figure 2.1, which highlights the intense price explosion and following rapid decline during 2017 and early 2018. If this figure is compared to Figure 1.1 (the aggregated market capitalisation of the top 100 cryptocurrencies) it becomes apparent that the main driver of the market capitalisation of cryptocurrencies is in fact the Bitcoin price.

Another example for this extraordinary volatility is portrayed in Figure 2.2, which compares the log return of Bitcoin to the one of the MSCI World Index. Note the unparalleled negative spike in early 2013, when the Bitcoin value dropped from 266 US$ to 105 US$, a staggering 61% decline, in just six hours on the 10^{th} of April[17]. This was the result of Bitcoin's main exchange at that time, MTGox, struggling to handle trades during this day, which lead to a panic sell-off among traders[18].

[17] See http://www.businessinsider.com/bitcoin-crash-biggest-market-declines-2013-4, retrieved on 09/05/2018.
[18] See http://www.bbc.com/news/technology-22105322, retrieved on 09/05/2018.

Feature (log difference)	Mean	Standard Deviation	Minimum	Maximum
btc close return	0.0034	0.0500	−0.6639	0.3375
btc open return	0.0034	0.0504	−0.6929	0.3360
btc high return	0.0034	0.0448	−0.4689	0.3328
btc low return	0.0034	0.2020	−5.9713	6.0256
btc volume	0.0012	0.6001	−1.9020	2.3110
btc addresses	0.0015	0.1331	−0.5175	0.5394
btc hash rate	0.0072	0.1240	−0.5486	0.5943
btc miner revenue	0.0030	0.1288	−0.6048	0.6393
btc cost per tx	0.0021	0.1526	−0.6140	0.5217
btc block size	0.0012	0.1413	−0.5882	0.6481
btc tx volume	0.0001	0.3133	−1.4687	2.0232
btc number tx	0.0011	0.1277	−0.5606	0.5087
btc tx conf time	−0.0002	0.2270	−1.2465	1.1753
eth close return	0.0028	0.0587	−0.9163	0.3830
xrp close return	0.0021	0.0937	−0.9973	1.0280
ltc close return	0.0021	0.0742	−0.9345	0.8904
fed funds rate	0.0005	0.0147	−0.1178	0.1750
gold return	−0.0002	0.0096	−0.0960	0.0484
usd/eur rate	0.0003	0.0068	−0.0503	0.0257
msci return	0.0000	0.0025	−0.0136	0.0122
vix	0.0001	0.0793	−0.2998	0.7682
google trends	0.0018	0.1469	−0.8518	1.1916
wiki views	0.0004	0.1769	−2.1654	2.5956

Table 2.6: Descriptive Statistics.

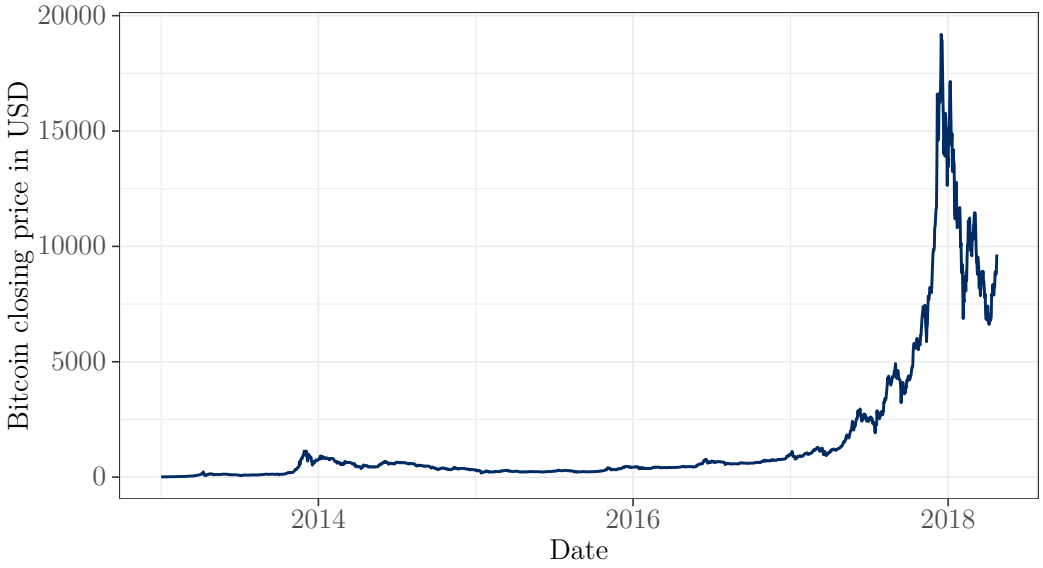

Figure 2.1: Bitcoin closing price.

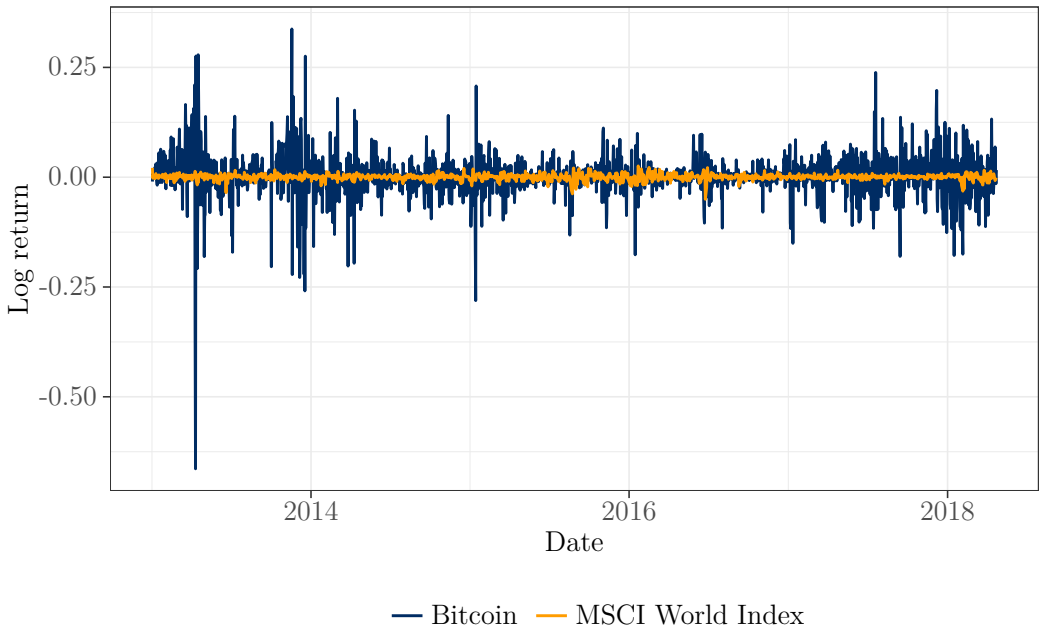

Figure 2.2: Bitcoin and MSCI World Index in log returns.

Chapter 3

Methodology

The goal of this chapter is twofold. In the first section, we lay the theoretical foundation for the subsequently applied machine learning and classical time series methods. These methods play an important role in the second section, where we explicitly explain our methodological framework to predict the one-day-ahead Bitcoin price trend and to evaluate corresponding trading strategies.

3.1 Theory

To account for methodological differences in modern machine learning and classical time series approaches, we split these topics into two subsections.

3.1.1 Machine Learning Methods

Logistic Regression

It is almost always a good idea to start with a simple and inexpensive to compute model when considering machine learning tasks before one deploys the cavalry in form of complex models like a deep neural network. In our case, for the prediction of a binary variable (Bitcoin trend), the most basic approach would be to apply a logistic regression to the data at hand. This ensures that the complexity we add to our model actually helps explaining the data. The logistic regression model was developed by Cox (1958) and has been in use ever since. Logistic regressions belong to the class of generalised linear models and

are thus relatively easy to interpret and calculate. The form of a generalised linear model is always

$$\mathbb{E}[y_t|\boldsymbol{x}_t] = \theta(\boldsymbol{w}^T\boldsymbol{x}_t), \tag{3.1}$$

where y_t is the signal to be predicted, $\boldsymbol{x}_t$ the independent variables, $\boldsymbol{w}^T$ the transposed weights, and $\theta(\cdot)$ the link function. Equation (3.1) can be rewritten as

$$\mathbb{P}[y_t = 1|\boldsymbol{x}_t] = \theta(\boldsymbol{w}^T\boldsymbol{x}_t), \tag{3.2}$$

since our dependent variable, the Bitcoin trend, is binary. In the case of the linear regression model, the link function is simply $\theta_{linear}(x) = x$ whereas for the logistic regression framework it is the sigmoid function, which is defined by

$$\theta_{sigmoid}(x) = \frac{1}{1+e^{-x}},$$

and whose output lays between 0 and 1. Hence, a logistic regression predicts *probabilities* of a binary dependent variable given the vector of independent variables $\boldsymbol{x}_t$ (see Equation (3.2)).

The logistic regression model in (3.2) can be derived from an underlying latent variable model, econometrically specified as

$$y_t^* = \boldsymbol{w}^T\boldsymbol{x}_t + \varepsilon_t \quad \text{and} \quad y_t = \begin{cases} 1, & \text{if } y_t^* > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where the errors ε_t are standard logistically distributed.

In contrast to many other nonlinear machine learning algorithms like support vector machines or neural networks the interpretation of the logistic regression weights $\boldsymbol{w}$, numerically estimated via maximum likelihood estimation, is relatively straightforward. For economists the weight's sign indicates the direction of the causal effect of x_i on $\mathbb{P}[y = 1|\boldsymbol{x}]$. The exact causal effect of a marginal increase in the input parameter x_i to the probability predicted by the logistic

model is calculated as

$$\frac{\partial \mathbb{P}[y=1|\boldsymbol{x}]}{\partial x_i} = w_i \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{\left(1+e^{\boldsymbol{w}^T\boldsymbol{x}}\right)^2}.$$

Fully-Connected Neural Network

Neural networks and deep learning have recently found huge success and fame in academia as well as in industry applications. They achieved remarkable results in many fields, e.g. image classification, speech recognition, digital assistants like Siri from Apple, or superhuman Go playing engines like Google's AlphaGo. Day to day activities like translating a text or website with Google Translate is nowadays done by a neural network with a stack of seven large layers (see Chollet and Allaire (2018), p. 202) and thanks to autonomous driving powered by them we might not have to worry about driving cars in the future.

The basic idea behind neural networks is that various so called neurons are chained together in layers. A neuron consists of an *activation function* (equivalent to the previous mentioned link function), a prespecified amount of inputs and outputs, and a weight for each input. Each layer contains an also prespecified amount of neurons. The amount of layers and neurons form a first set of hyperparameters that each neural network possesses. Figure 3.1 shows a basic fully-connected neural network, also called multilayer perceptron (MLP), with only one hidden layer. Each blue dot represents a neuron and each neuron has as many weights assigned to it as there are neurons in the previous layer. A neuron j calculates the value it passes on, the output o_j, by

$$o_j = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right), \tag{3.3}$$

where $\varphi(\cdot)$ is the activation function, n the number of neurons in the previous layer, w_{kj} the weight of the output from neuron k, and o_k the output of neuron k that is located in the previous layer. For a neuron located in the input layer this o_k will be input x_k.

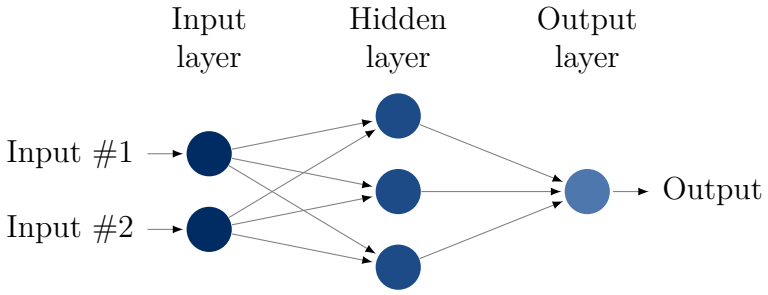

Figure 3.1: A simple neural network with one hidden layer.

The activation function $\varphi(\cdot)$ can theoretically differ for each neuron, however it is customary to set it to the same value for all hidden layers. Although, for the output layer it is important to choose a function that appropriately models the dependent variable's type (e.g. binary, continuous, etc.). For example, it does not make sense to choose an activation function for the output layer that returns binary values when a regression task is the problem at hand. Without an activation function the neural network would essentially be just a linear regression model, which would be incapable of learning nonlinear tasks (see Gupta (2017)). There exists a plethora of different activation functions, but we only use four different ones, namely the already known sigmoid function, equivalently formalised as

$$\varphi_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}},$$

and used for the output layer to model probabilities, the rectified linear unit (ReLU) function, defined as

$$\varphi_{\text{relu}}(x) = \max(0, x),$$

a variant of the ReLU, the leaky rectified linear unit (lReLU):

$$\varphi_{\text{lrelu}}(x) = \begin{cases} ax, & \text{if } x < 0, \\ x, & \text{otherwise,} \end{cases} \tag{3.4}$$

and a hyperbolic tangent (Tanh) function, defined as

$$\varphi_{\tanh}(x) = \frac{2}{1 + e^{-x}} - 1.$$

Figure 3.2, 3.3, 3.4, and 3.5 show how each of these activation functions map input data and illustrate their respective derivatives.

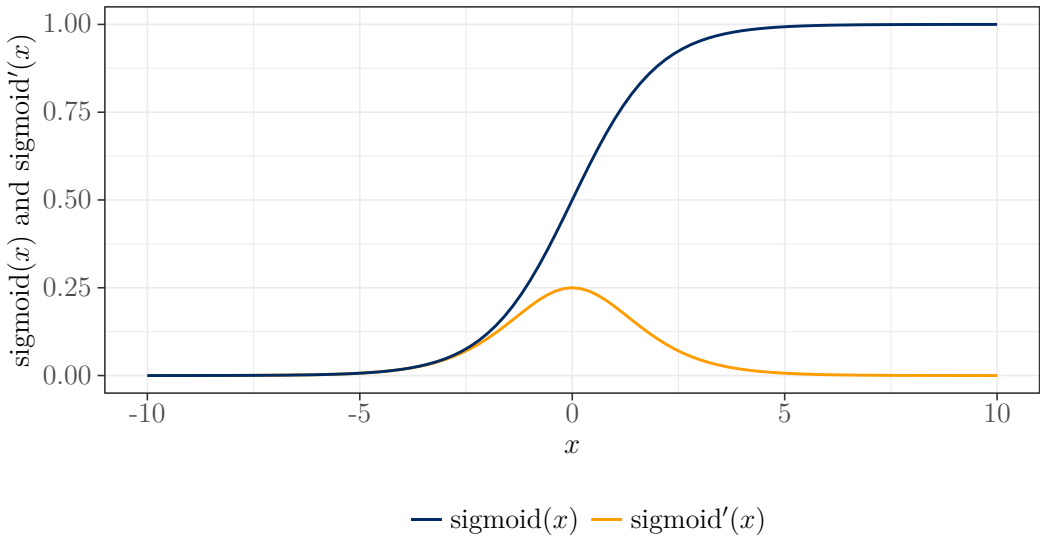

Figure 3.2: Sigmoid – function and first derivative.

Neural networks are, regardless of the recent hype, not a new subfield of machine learning. McCulloch and Pitts (1943) laid the foundation for their mathematical and logical background. However, until the 1980s there was no efficient way to train large neural networks. This changed when Rumelhart et al. (1986) described a "new learning procedure, back-propagation, for networks of neurone-like units" (see Rumelhart et al. (1986), p. 533). Backpropagation is a way to train a neural network using gradient descent optimisation for the weights of each neuron.

Basically, the process of backpropagation consists of two phases: propagation and weight update.

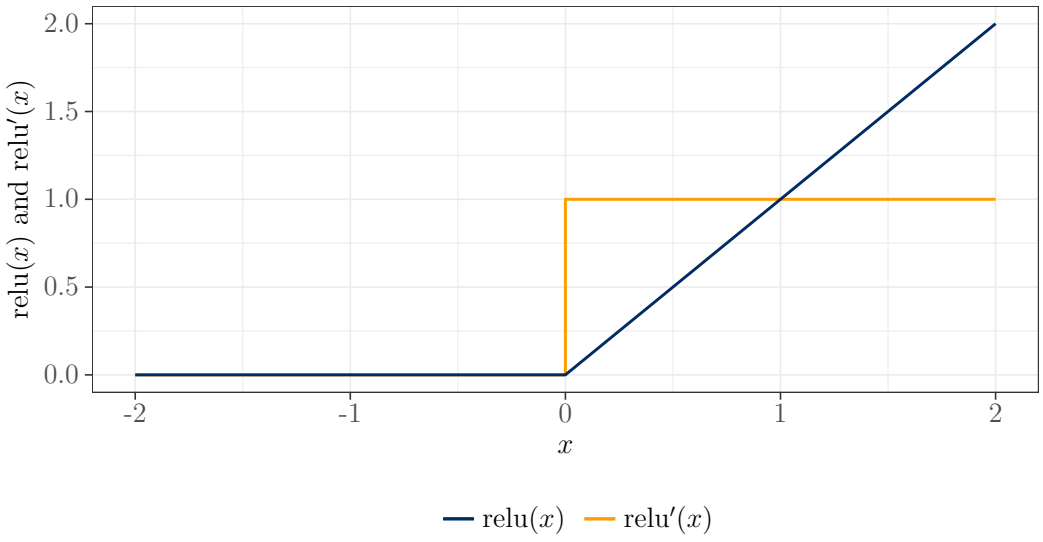

Figure 3.3: Rectified linear unit – function and first derivative.

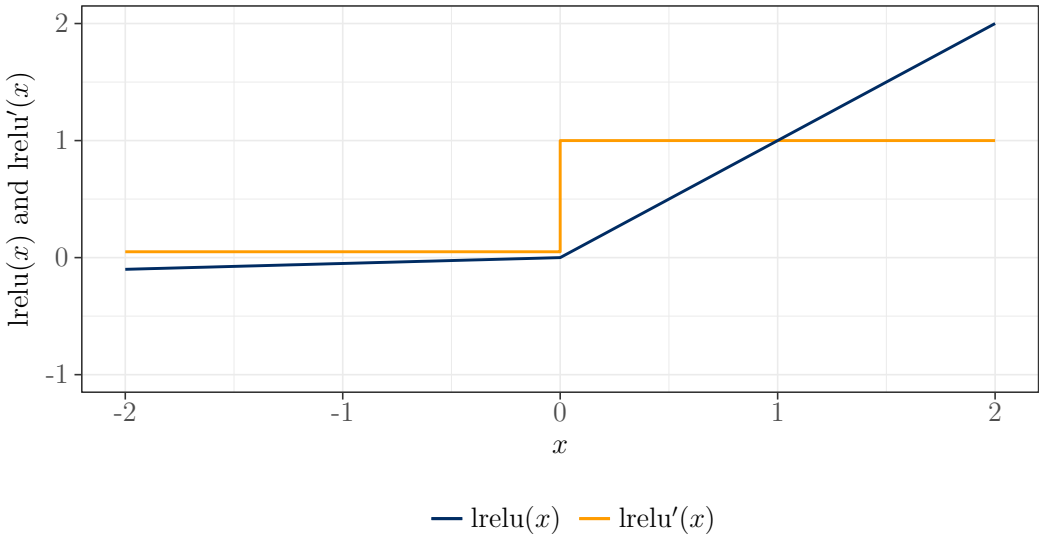

Figure 3.4: Leaky rectified linear unit with $a = 0.05$ – function and first derivative.

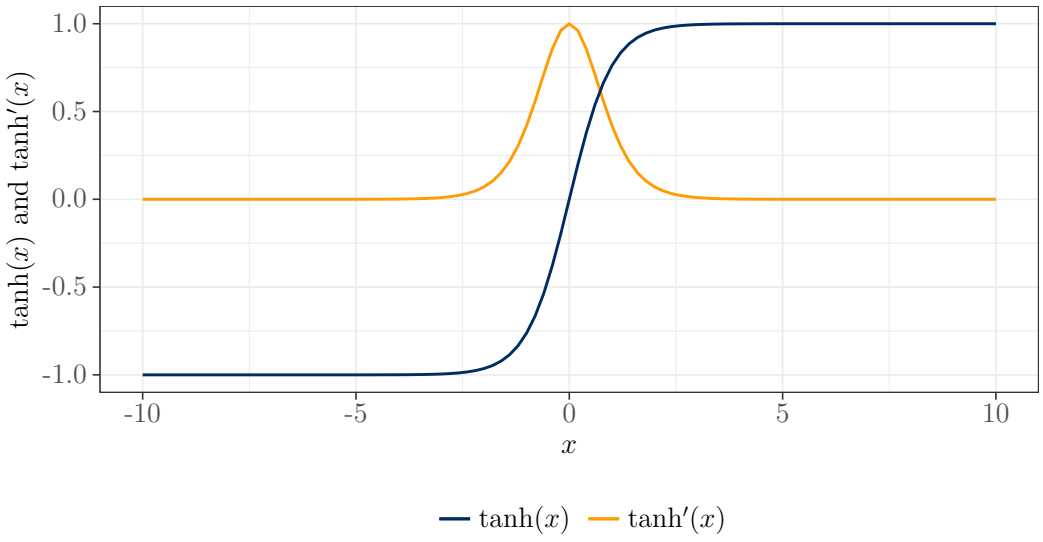

Figure 3.5: Hyperbolic tangent – function and first derivative.

Phase 1: Propagation

1. Output values are calculated by applying the weights to the input of each neuron, applying an activation function and passing the result to the next layer (the so called *forward pass*).
2. Calculation of the error term E_{total} with the loss measure ℓ.
3. Calculation of the partial derivatives of E_{total} with respect to each weight w_{kj}: $\frac{\partial E_{total}}{\partial w_{kj}}$.

Phase 2: Weight update

1. Each partial derivative $\frac{\partial E_{total}}{\partial w_{kj}}$ is multiplied by the learning rate η and then subtracted from its respective previous weight w_{kj} to generate the new, updated weight w_{kj}^{+}.

This process is repeated until the error term E_{total} reaches a certain threshold, a predefined number of steps is exceeded, or another stopping criteria is met.

An appropriate loss measure ℓ for a classification task like the one at hand is cross-entropy (see Friedman et al. (2009)). It describes how close the predicted distribution is to the true distribution and is defined for the binary case as:

$$\ell(p_t) = -\left(y_t \log(p_t) + (1 - y_t) \log(1 - p_t)\right),$$

where p_t is the predicted probability and y_t the actual label of observation t.

There are three different gradient descent variants: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, which differ in the way they update the weights and how much data they use to compute the gradient of the loss function (see Ruder (2016)). Whereas batch gradient descent will compute the gradient of the loss function for the entire training set, stochastic gradient descent performs a parameter update for each training observation. Mini-batch gradient descent is the mixture of both and updates for every mini-batch of n training examples.

Albeit, these methods can run into some problems. One problem is the tuning of the learning rate η. On the one hand, a too small learning rate drastically slows down the convergence. On the other hand, a too large learning rate can hinder convergence and cause the loss function to not reach a minimum. Another challenge is avoiding to get trapped in suboptimal local minima and saddle points when minimising the error. Dauphin et al. (2014) argue that "(a) non-convex error surfaces in high dimensional spaces generically suffer from a proliferation of saddle points, and (b) in contrast to conventional wisdom derived from low dimensional intuition, local minima with high error are exponentially rare in high dimensions" (see Dauphin et al. (2014), p. 8), which makes saddle points the much more problematic of the two for finding global minima in the usually very high dimensional loss functions of neural networks.

To tackle these problems we employ the ADAM (Adaptive Moment Estimation) algorithm, introduced by Kingma and Ba (2014). It computes adaptive learning rates for each weight, meaning that it performs large updates for infrequent- and smaller updates for frequent parameters, making it better

suited for sparse data. A momentum factor is also incorporated, which increases updates "for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions" (see Ruder (2016), p. 4). The algorithm stores an exponentially decaying average of past gradients $\boldsymbol{m}_t$ and past squared gradients $\boldsymbol{v}_t$ (note that all operations on vectors are element-wise in the following equations concerning the ADAM update rule):

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t, \tag{3.5}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2, \tag{3.6}$$

where $\boldsymbol{m}_t$ is an estimate for the mean and $\boldsymbol{v}_t$ for the uncentered variance of the gradients, and $\boldsymbol{g_t}$ is the gradient defined as:

$$\boldsymbol{g}_t = \nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}_{t-1}),$$

with $\ell(\cdot)$ being the loss function.

Kingma and Ba (2014) observed that $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are biased towards zero when initialised as a vector of zeroes. To control for this bias, they proposed using bias-corrected mean and variance estimates:

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t},$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}.$$

The vector of weights $\boldsymbol{w}$ is then updated by:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}\hat{\boldsymbol{m}}_t, \tag{3.7}$$

where ϵ is an infinitesimal value to avoid dividing by zero and η is the learning rate.

The authors have found "Adam to be robust and well-suited to a wide range of non-convex optimisation problems in the field machine learning" and that

"experiments confirm the analysis on the rate of convergence in convex problems" (see Kingma and Ba (2014), p.10).

When applied to multivariate time series (as in our case), multilayer perceptrons face the challenge of handling the inter-temporal relation of this type of data. In order to deal with this challenge we utilise the so called *moving window* approach, in which the MLP is trained to predict the label $\boldsymbol{y}$ with the explanatory variables $\boldsymbol{x}$ lagged by one time period up to the lookback l. For a simple example, consider the case of only two explanatory variables x_1 and x_2 and a lookback period of $l = 2$. The dependent variable y_t is then predicted by $x_{1,t-1}$, $x_{1,t-2}$, $x_{2,t-1}$, and $x_{2,t-2}$. Grudnitski and Osburn (1993) had considerable success with this method when forecasting S&P500 and gold future prices and were able to correctly predict the direction of these with an accuracy of 75% and 61%, respectively.

Recurrent Neural Network

The fully-connected neural networks we illuminated before have the characteristic that they have no memory, meaning that each input is processed independently and no state is kept in between. A recurrent neural network changes this. It processes sequences by iterating through its elements and keeping a *state* of what the network has already seen before. Therefore, a recurrent neural network is essentially a neural network with an internal loop, where the output of the previous observation is used in the calculation of the current output (see Figure 3.6).

Hence, this basic recurrent neural network conceptionally looks like Figure 3.7, where the output t is inserted as (hidden) state t together with the input $t + 1$ to compute the output $t + 1$. The very first state is initialised with an all zero vector, called the initial state (see Chollet and Allaire (2018)).

More formally, given the input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$, a recurrent

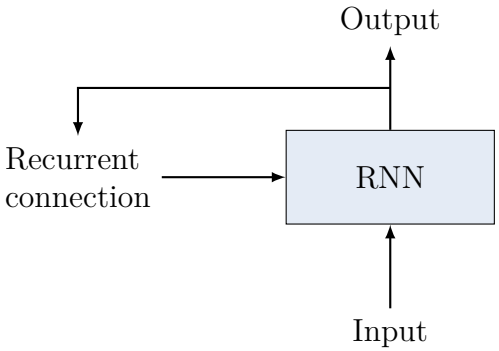

Figure 3.6: A recurrent network: a network with a loop (source: Chollet and Allaire (2018), Figure 6.7).

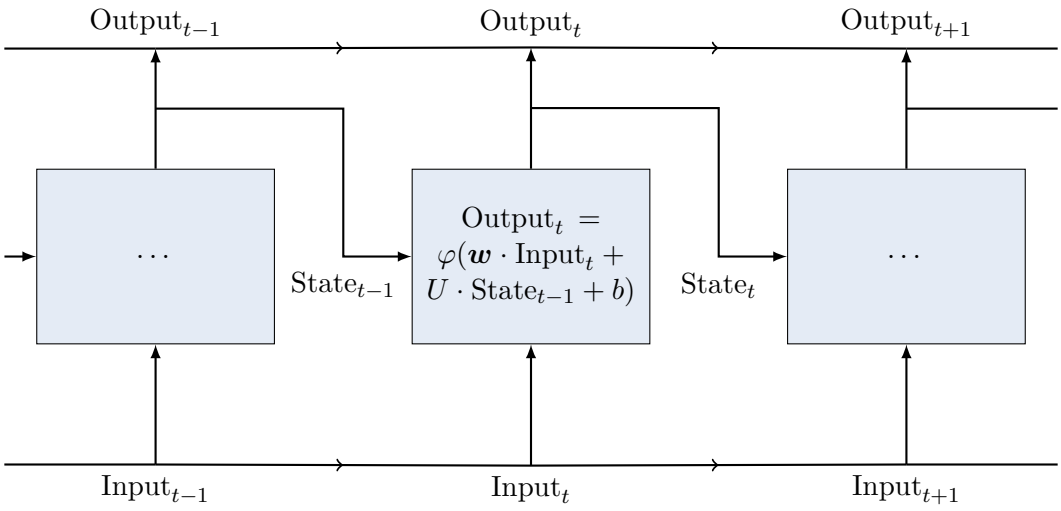

Figure 3.7: A simple recurrent neural network, unrolled over time (source: Chollet and Allaire (2018), Figure 6.8).

neural network updates its hidden state $\boldsymbol{h}_t$ by

$$\boldsymbol{h}_t = \begin{cases} 0, & \text{if } t = 0, \\ \varphi(\boldsymbol{w}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}), & \text{otherwise,} \end{cases} \tag{3.8}$$

where $\varphi(\cdot)$ is again the activation function as introduced in the previous chapter, $\boldsymbol{w}$ the weights for the input $\boldsymbol{x}$, $\boldsymbol{U}$ the trainable weights for the hidden

state and $\boldsymbol{b}$ a bias vector to account for possible constants or intercepts.

Unfortunately, Bengio et al. (1994), amongst others, observed that it is nearly impossible for a recurrent neural network to efficiently learn crucial information about inputs seen many timesteps ago. This is due to the *vanishing gradient problem*, which describes the observation that gradients tend to vanish, or, albeit more rarely, due to the *exploding gradient problem*, in which the gradient exponentially explodes. However, two models have been developed to counteract these effects. First, the long short-term memory model (LSTM) developed by Hochreiter and Schmidhuber (1997) and second, the gated recurrent unit (GRU), model proposed by Cho et al. (2014). For all our purposes it is sufficient to know that they both achieve the same goal of handling the two issues mentioned before, gradient vanishing and exploding, much better than the *vanilla* recurrent neural network (see Chollet and Allaire (2018)). Performance wise GRUs seem to be able to converge faster, are cheaper to run and handle small datasets better, but might lose some representational power when compared to LSTM (see Chung et al. (2014) and Chollet and Allaire (2018)). Thus, we use a GRU neural network in our analysis, as one of the main issues we are facing is the relatively small available dataset size due to the infancy of Bitcoin relative to other financial asset.

Figure 3.8 visualises the functionality of a gated recurrent unit. As Cho et al. (2014) describe in simple words, "the update gate z selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$" and "the reset gate r decides whether the previous hidden state is ignored".

More formally, the activation $h_{t,j}$ for a neuron j of a GRU neural network is a linear interpolation between the previous activation $h_{t-1,j}$ and a candidate activation $\tilde{h}_{t,j}$:

$$h_{t,j} = (1 - z_{t,j})h_{t-1,j} + z_{t,j}\tilde{h}_{t,j},$$

where an update gate $z_{t,j}$ controls the magnitude of the update of each unit's activation and is computed by

$$z_{t,j} = \varphi_{\text{sigmoid}}(w_z \boldsymbol{x}_t + U_z \boldsymbol{h}_{t-1}). \tag{3.9}$$

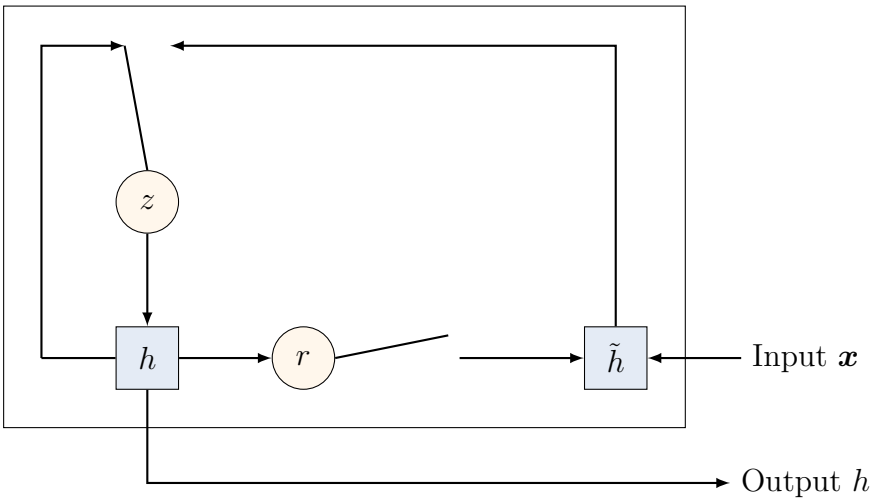

Figure 3.8: Illustration of a gated recurrent unit (source: Chung et al. (2014), Figure 1 (b)).

The candidate activation $\tilde{h}_{t,j}$ is calculated similarly to (3.8):

$$\tilde{h}_{t,j} = \varphi_{\tanh}(w\boldsymbol{x}_t + U(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1})),$$

where $\boldsymbol{r}_t$ is a set of reset gates and $\odot$ is an element-wise multiplication. This reset gate basically tells the neuron when to and how much of previous states should be included in the computation. Hence, when $r_{t,j}$ is close to zero, the neuron will not consider previously computed states. The reset gate $r_{t,j}$ is computed quite similarly to the update gate in Equation (3.9):

$$r_{t,j} = \varphi_{\text{sigmoid}}(w_r\boldsymbol{x}_t + U_r\boldsymbol{h}_{t-1}).$$

Convolutional Neural Network

Whereas dense fully-connected neural networks learn global patterns in their input feature space, convolutional neural networks (CNN) learn local patterns which make the patterns they learn translation invariant. This means that no matter where they find a certain pattern they will be able to recognise it. Take for example the task of classifying images into certain categories, which

is a typical field where convolutional neural networks shine. A fully-connected neural network would have to find a previously recognised pattern in the same location to classify it correctly, whereas a convolutional neural network can recognise it anywhere in the image. This leads to phenomenal performance in this area, as shown in many studies, for example Krizhevsky et al. (2012) or Simonyan and Zisserman (2014). As documented in Chollet and Allaire (2018), they also perform exceptionally well on time series forecasting, especially if the amount of data is limited.

But before we get into time series forecasting, let us focus on what a convolutional neural network actually is. A convolution layer takes a feature map, which can theoretically be a tensor of any rank, and outputs a tensor of response maps by applying a certain kernel repeatedly to the feature map. The kernel transforms the input by sliding a window of previously specified dimension over the input feature map, extracting this patch, and performing an element-wise product with the feature map window, followed by summing up the result, which yields a scalar. In the following this calculation is called element-wise dot product.

To better illustrate this, assume that we have a 2D input tensor of shape 4×4 only containing binary values (orange colour only for illustrative purposes):

$$T_{in} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & \mathbf{0} & \mathbf{1} & 0 \\ 1 & \mathbf{0} & \mathbf{0} & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and a kernel of size 3×3:

$$\kappa = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

The actual convolution is taking place when the element-wise dot product is

taken between the kernel and the input. Clearly, one cannot simply multiply T_{in} with κ, as the dimension of these do not match. But this is not what we want to achieve. The kernel will be applied to a window of the same size as the kernel and the only possible windows in T_{in} are the ones that have an orange coloured number at its centre. Therefore, the first element of the result Λ of the convolution is defined by:

$$\begin{aligned}\Lambda_{1,1} &= \begin{bmatrix} 0 & 1 & 1 \\ 0 & \mathbf{0} & \mathbf{1} \\ 1 & \mathbf{0} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \\ &= 2,\end{aligned}$$

the second element is:

$$\begin{aligned}\Lambda_{1,2} &= \begin{bmatrix} 1 & 1 & 1 \\ \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{0} & \mathbf{0} & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \\ &= 3,\end{aligned}$$

and so on. Hence, the final result for the convolution is:

$$\begin{aligned}\Lambda &= \begin{bmatrix} \Lambda_{1,1} & \Lambda_{1,2} \\ \Lambda_{2,1} & \Lambda_{2,2} \end{bmatrix} \\ &= \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}.\end{aligned}$$

In practice, many kernels are usually applied and the actual values of the kernels are learned from the data. The size and dimension of the output feature map, also called convolved feature, is dependent on

- **Kernel size** – A larger kernel results in a smaller convolved feature.
- **Depth** – Higher depth, meaning more kernels that are applied to the input, increases the amount of learnable patterns and thus, the size of the output feature map.
- **Stride** – It is possible to widen the steps between every kernel application to more than one. Naturally, this decreases the size of the output. This is rarely used in a convolution layer and the below described pooling is usually preferred.
- **Padding** – One could add zeroes around the border so that the filter is also applied to bordering elements, which increases the size of the output feature map.

Another way of aggressively reducing the size of feature maps is to implement a pooling operation after each convolution layer. Pooling consists of extracting windows from the input feature map and applying a hardcoded function to it, often taking the maximum, average, or sum of the window. Hence, it is conceptionally quite similar to a convolution layer, but the learned linear transformation, called convolution kernel, is exchanged for an unchanging function. Furthermore, pooling is usually done with an added stride having the same size as the length of the window to downsample the feature map and being able to learn patterns larger than the window size (see Chollet and Allaire (2018)). Moreover, without adding pooling layers the amount of total weights are growing immensely with the quantity of kernels, which leads to intense overfitting (see Chollet and Allaire (2018)).

The main difference between using convolutional neural networks for tasks like image classification and for time series forecasting is the dimension of the convolution layer. Whereas it is customary to use a 2D convolution for the first case, one would naturally use a 1D convolutional neural network to process time series data (see Chollet and Allaire (2018)).

3.1.2 Classical Methods

Autoregressive Moving Average Model (ARMA)

A simple and cheap, in terms of computing power, way to model a stationary time series that exhibits autocorrelation on a linear level like many economic variables do, is to fit an *Autoregressive Moving Average*, or short ARMA, model to it. Since we assured in the previous Chapter *Data* that our time series is stationary, differencing is no longer necessary and thus, we drop the integrated part, "I", of the more general ARIMA model. Hence, we arrive at the compact ARMA(p,q) model, which can be formalised as

$$\begin{aligned} y_t &= \beta_0 + \beta_1 y_{t-1} + ... + \beta_p y_{t-p} + \varepsilon_t + \alpha_1 \varepsilon_{t-1} + ... + \alpha_q \varepsilon_{t-q} \\ &= \beta_0 + \varepsilon_t + \sum_{i=1}^{p} \beta_i y_{t-i} + \sum_{i=1}^{q} \alpha_i + \varepsilon_{t-i}, \end{aligned}$$

where y_t is the variable to be explained at time t, β_0 the intercept, ε_t the error term at time t, β_i the coefficients of the ith autoregressive part (AR component), and α_i the coefficients of the ith moving average part (MA component).

To find the optimal lag structure p and q one can either use the AIC, short for *Akaike information criterion* (see Akaike (1974)), the BIC, short for *Bayesian information criterion* (see Schwarz (1978)), or a combination of them. Another option is to look at the correlogram of the autocorrelation- and partial autocorrelation function, often simply shortened to ACF and PACF respectively. The autocorrelation is the linear dependence of a time series on its previous lags, whereas the partial autocorrelation is the linear dependence of a time series on a specific lag after removing the autocorrelation of other lags. Theoretically, for an ARMA(p,0) the ACF tails of gradually and the PACF cuts of after p lags, whereas for an ARMA(0,q) it is the opposite: the ACF cuts of after q lags and the PACF tails of gradually. If an ARMA(p,q) is present both ACF and PACF converge gradually (see Tsay (2010) for this behaviour). The plotted ACF and PACF of a stylised ARMA(1,0) with $\beta_1 = 0.75$ can be seen in Figure 3.9 and 3.10.

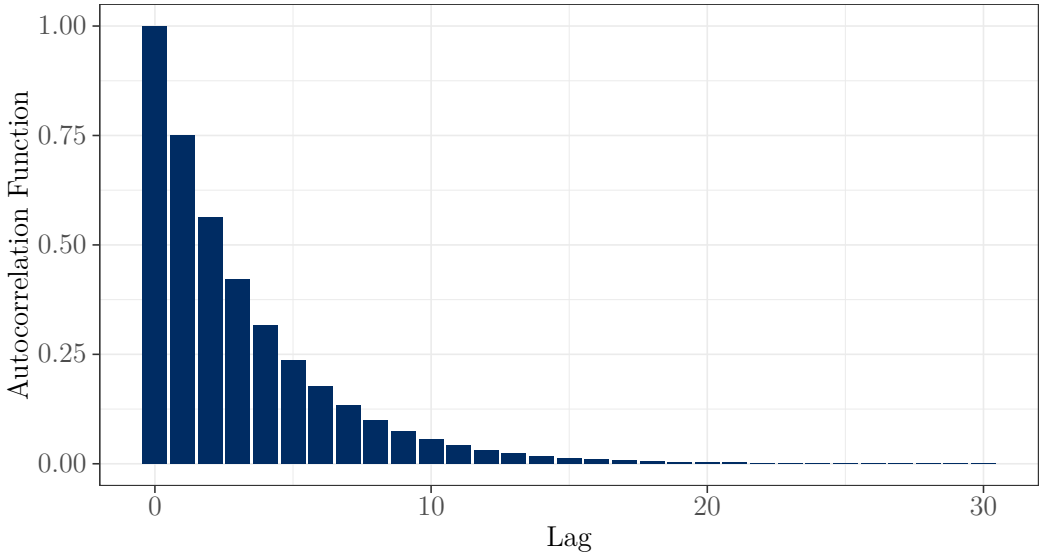

Figure 3.9: Autocorrelation function for an ARMA(1,0) process with $\beta_1 = 0.75$.

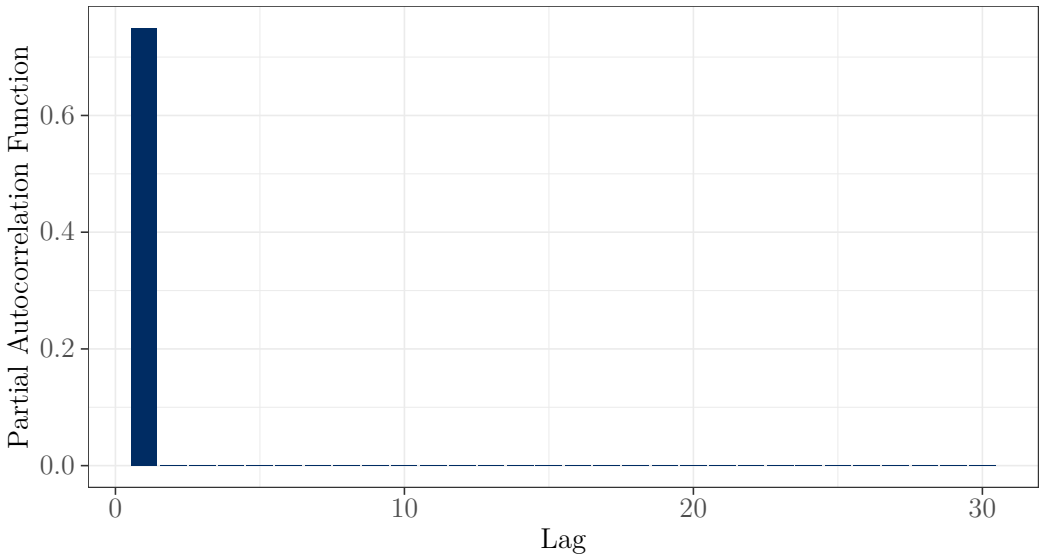

Figure 3.10: Partial autocorrelation function for an ARMA(1,0) process with $\beta_1 = 0.75$.

Trend Following Strategy (Momentum)

Trend following strategies for other asset classes than cryptocurrencies had considerable success in the past and continue to perform well in today's markets. The idea behind trend following strategies is simple: go with the trend, i.e. if the market is trending up, buy and if the market is trending down, sell. How to determine this trend can be as simple as summing over recent returns or as difficult as applying a complex long, medium or short term transformation to the returns. In this work we will only consider the most basic trend following strategy: time series momentum. This should not mean that other, more complex strategies are worse in predicting cryptocurrency returns. However, finding the optimal trend following strategy for a certain cryptocurrency, say Bitcoin, would probably fill another thesis of this extent completely and thus, is clearly beyond the scope of this work.

If a trader follows the time series momentum, he will go long markets that exhibited positive recent returns and short those with negative recent returns. To arrive at these returns and to form the trading signal (long or short) one takes the sum over a previously specified period and sees whether they are positive or not. A positive (negative) sum implies a *buy* (*sell*) signal. Expressing this in a more formal way for a lookback of l periods and a return r_t in period t the trading signal in period T would be

$$S_{T,\text{Momentum}} = \begin{cases} 1 & \text{if } \sum_{t=T-l}^{T-1} r_t > 0, \\ -1 & \text{else.} \end{cases}$$

According to Moskowitz et al. (2012), this strategy has been successful on average for nearly all equity index futures, fixed income futures, commodity futures and currency forwards since 1985. Hurst et al. (2017) have gone further and investigated this strategy on 29 commodities, 11 equity indices, 15 bond markets and 12 currency pairs from, if available, 1880 to 2013. They found that over this period longer than a century momentum strategies have performed considerably and consistently well. This would suggest that "trends

are pervasive features of global markets" (see Hurst et al. (2017), page 25).

In Pedersen (2015) a possible reason for this persistence is given by an initial underreaction to news and delayed overreaction. This initial underreaction is caused by various behavioural tendencies and market frictions (names taken from Pedersen (2015)):

1. **Anchor and insufficient adjustment**. Historical data might influence people too much and they do not change their views when new information emerge (see Tversky and Kahneman (1974) and Barberis et al. (1998)).

2. **The disposition effect**. People tend to sell winners too early to realise their gains and keep losers because realising those losses is discomforting. The first creates downward price pressure slowing upward adjustment and vice versa (see Shefrin and Statman (1985) and Frazzini (2006)).

3. **Non profit seeking activities**. The activities of central banks and investors who rebalance to certain strategic asset allocation (e.g. 60% stocks and 40% bonds) fall under this category and will work counteractive to the trend (see Silber (1994)).

4. **Frictions and slow moving capital**. Market frictions, delayed responses by significant market participants and slow moving capital could also slow down price discovery (see Mitchell et al. (2007) and Duffie (2010)).

The delayed overreaction may again be caused by different reasons:

1. **Herding and feedback trading**. After prices having moved in the same direction for some time, a few traders and analysts may believe that this trend is there to stay (see, amongst others, Bikhchandani et al. (1992) and Welch (2000)).

2. **Confirmation bias and representativeness**. People tend to look for information that is in line with what they already believe to be true

and hence, they think that recent price movements are an indicator for movements in the future. This could lead traders to buy assets that have gone up recently and sell those which have fallen (see Wason (1960), Tversky and Kahneman (1974) and Daniel et al. (1998)).

3. **Fund flows and risk management**. As investors likely pull money from underperforming fund managers these managers are forced to reduce their positions in underperforming assets, whereas outperforming fund managers will receive additional inflows and will invest those in their outperforming positions. This adds selling pressure to the underperforming assets and buying pressure to the ones outperforming. Additionally, there are risk-management schemes that imply to sell in down markets and buy in up markets (see Daniel et al. (1998)).

Both underreaction and overreaction are illustrated in Figure 3.11.

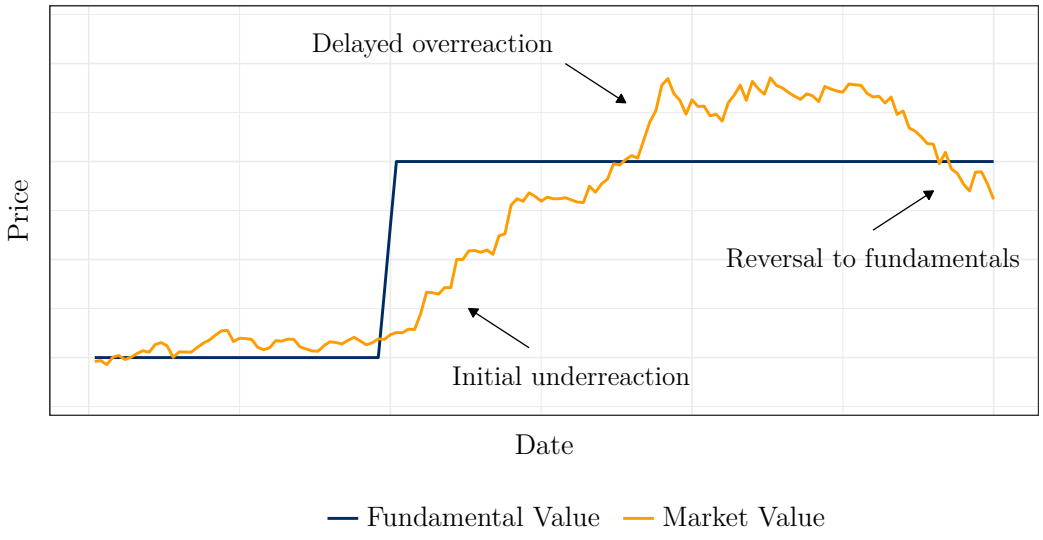

Figure 3.11: Stylised Momentum (inspired by Pedersen (2015), Figure 12.1).

3.2 Bitcoin Trend Forecasting and Performance Evaluation

The following two subsections describe our methodology and implementation of predicting the one-day-ahead Bitcoin price trend and its utilisation in developing an automated trading strategy. Specifically, the first subsection explains the concept of the complex model selection process, which is employed to determine the ideal feature set, lag structure, hyperparameter setup and machine learning algorithm, yielding the highest prediction accuracy on independent validation sets. The second Subsection illustrates how these Bitcoin trend predictions are used to define an automated trading strategy. Furthermore, we introduce certain benchmarks and performance measures to compare and evaluate the resulting strategies.

3.2.1 Model Selection

When confronted with a great variety of potential input features, machine learning algorithms and model hyperparameters, a structured model selection approach is needed to identify the optimal configuration for the prediction task in question. In our case this model selection framework needs to determine the following meta-parameters such that the chosen combination yields the best forecasting performance:

- Set of input features, $\boldsymbol{x}$, used to predict the Bitcoin trend.
- Lag structure or lookback period of this input feature set.
- Choice of the machine learning algorithm.
- Algorithm specific hyperparameters like
  - Dropout rate, to fight overfitting.
  - Amount of hidden layers and units (neurons) in the neural network architecture.
  - Kernel filter size and quantity in the convolutional neural network architecture.

– Tunable parameters of the ADAM backpropagation algorithm used during model training.

Before turning to the analysis and actual selection of these free parameters, let us first introduce the two non-competing, but rather interacting methods for model selection discussed in the time series literature, namely the so called *filter* and *wrapper* approach. Filters are exclusively used for feature selection, including the choice of the feature's lag structure. They make use of linear statistics like autocorrelation analysis or stepwise regression to determine relevant predictors or lags and thereby, reduce the search space of potential parameters. Due to their linearity and independence of specific prediction algorithms, filters are computationally very efficient and fast. However, the same properties also cause reduced flexibility and filtering accuracy, since nonlinearities and algorithm specific biases cannot be captured (see Crone and Kourentzes (2010)).

In contrast, the wrapper approach incorporates the algorithm's intrinsic characteristics and biases. In fact, it performs a grid search on a range of feature-, model-, and hyperparameter combinations to compute their predictions and compare them based on the resulting forecasting accuracies. Hence, depending on the machine learning algorithm at hand, the wrapper approach considers nonlinear relationships in its evaluation and selects not only features, but all possible meta-parameters. That is why "wrappers are often recognised as a superior alternative for feature evaluation in supervised learning problems" (Crone and Kourentzes (2010), p.1925). Nevertheless, they suffer from their computational inefficieny, since implementing a grid-search requires looping through all potential meta-parameter combinations (see the above list), which increase exponentially in the amount of tunable characteristics. Therefore, the application of wrappers is limited by the available computational power and training time.

As a consequence, we follow Crone and Kourentzes (2010) and utilise both, the filter and wrapper approach, to benefit from their respective advantages. To initially reduce the search space of free parameters we leverage the autocorrelation (ACF) and partial autocorrelation function (PACF) of each predictor

to get a general idea of the lookback period (lag structure) for our feature set. For illustration purposes Figure 3.12 and 3.13 plot the ACF and PACF of the Bitcoin log return based on its closing price, respectively.

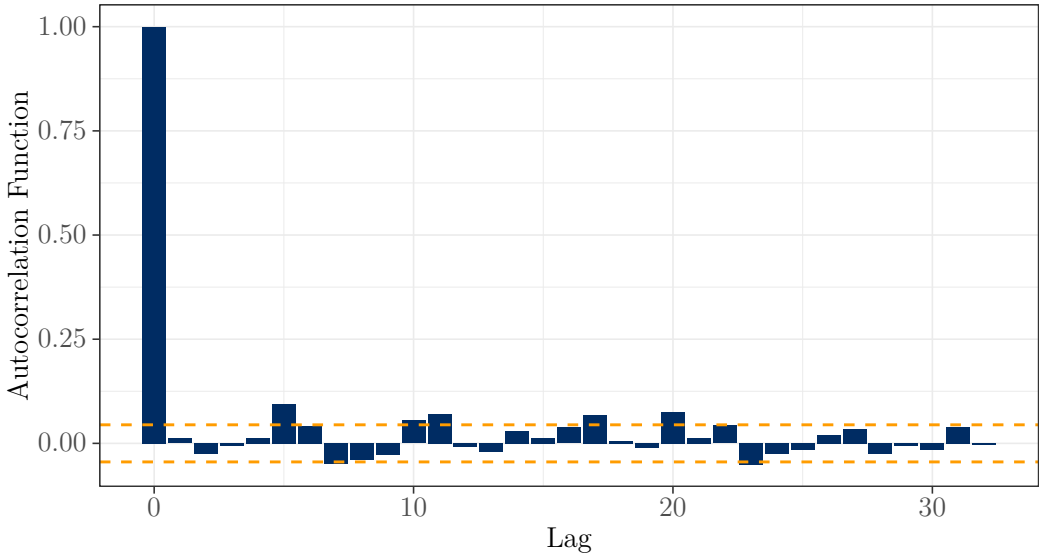

Figure 3.12: ACF of Bitcoin log return based on its closing price.

Taking a look at the ACF in Figure 3.12 it can be observed that almost no linear dependence in the time series is present. However, this behaviour might not be unexpected for financial assets, since it is generally considered very hard to deploy past performance as an indicator for future outcomes. Furthermore, note that potential nonlinearities are not captured by the ACF. In addition, the PACF plot in Table 3.13 displays a range of significant lags in the partial autocorrelation of Bitcoin log returns. Taking into account both plots and their potential shortfalls, we decide to incorporate and analyse two lookback periods of 14 and 28 days in the subsequent wrapper approach.

On top of that, due to the previously stated time and computational limitations, we exclude certain meta-parameters from tuning via grid-search. One of these parameters is the dropout rate, which we include into each neural network architecture by adding an extra dropout layer to efficiently reduce

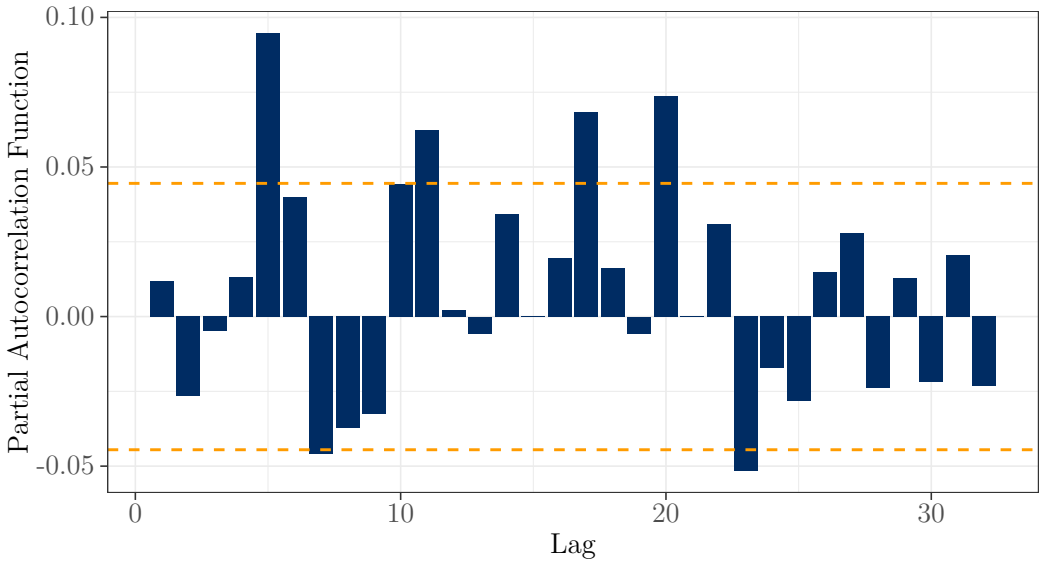

Figure 3.13: PACF of Bitcoin log return based on its closing price.

the amount of overfitting. Traditional overfitting can be defined as flexibly training an algorithm so that it fully memorises the training data (zero training error), but fails to generalise well on unseen test data (high test error). It especially occurs when training a complex high dimensional predictor on too few training observations. The problem gets even more severe in our case when employing sophisticated algorithms like fully-connected-, GRU- and convolutional neural networks with many weight parameters to estimate. Even though other counter measures exist, Chollet and Allaire (2018) argue that dropout, developed by Srivastava et al. (2014), is "one of the most effective and most commonly used regularisation techniques for neural networks" (see Chollet and Allaire (2018), p. 100). Basically, after dropout is applied to a layer, the output vector o_j (see Equation (3.3)) has a fraction of zero entries distributed at random. This fraction represents the dropout rate and following the findings in Srivastava et al. (2014), who describe that it is optimally chosen between 0.2 and 0.6, we set it to a fixed value of 0.4.

For similar reasons we set the values for the ADAM backpropagation al-

gorithm (see Equations (3.5), (3.6), and (3.7)) to the ones proposed by Kingma and Ba (2014): 0.9 for β_1, 0.999 for β_2, 0.0001 for η, and 10^{-8} for ϵ. However, Kingma and Ba (2014) also tested a broad range of other hyperparameter values and concluded that "Adam performed equal or better than RMSProp [unpublished, alternative learning rate method[1]], regardless of the hyperparameter setting" (see Kingma and Ba (2014), p.8).

Lastly, we determine that all neural network architectures, except the convolutional neural network, consist of only one hidden layer of neurons. For the CNN we implement three hidden layers, namely two convolution and a final fully-connected layer. Within that setup we use 16 and 8 kernels respectively (for the first and second convolution) with a kernel length of 6 each. To introduce nonlinearity, we apply the leaky ReLU activation function[2] to each convolution layer, which improves the performance significantly as documented in Xu et al. (2015).

After fixing these meta-parameters our grid-search is reduced to a more reasonable amount of tunable parameters. In particular we define the following search space, which consists of:

- 4 different machine learning algorithms, namely the logistic regression, fully-connected neural network, GRU neural network and convolutional neural network.
- 32 unique feature combinations, employing different setups based on the five categories introduced in the *Data* Chapter. The complete list of all selected combinations can be seen in Appendix B.
- 2 lookback periods of 14 and 28 days, determined by the ACF and PACF filter approach.
- 2 values, 16 and 32, for the amount of hidden neurons used in all neural network models.

[1]Proposed by Geoffrey Hinton in Lecture 6e of his Coursera Class.
[2]See Equation (3.4) and Figure 3.4 in Subsection 3.1.1 within this Chapter.

Hence, we evaluate all $32 \cdot 2 \cdot 2 = 128$ parameter combinations for each of the above listed algorithms based on their respective validation accuracies. When facing cross-sectional data this is usually done by applying cross-validation to avoid overfitting on the training data and to choose an optimal model configuration that generalises well to new unseen data (see, amongst others, Abu-Mostafa et al. (2012)). While cross-validating, the training dataset is randomly shuffled, and divided in K equally sized folds. The k^{th} fold is set apart for validation, and the model is trained on the remaining $K-1$ folds and validated on the observations set aside before. This is repeated for each $k = 1, \ldots, K$ fold until every data point is used for validation exactly once. The cross-validation accuracy is then computed by averaging over the validation accuracies of these folds.

In our work this method is problematic, as randomly shuffling time series data eliminates the time dependence of observations, which is what is used in many forecasting approaches, including ours. To still be able to determine the remaining hyperparameters we apply a 4-fold expanding window validation ($K = 4$) as described in Hyndman and Athanasopoulos (2014) and illustrated in Figure 3.14. The procedure for this validation approach is as follows:

1. Select the observations of the first k folds, and train the model on this data subset. Compute the accuracy on a validation set containing the samples of the next $(k + 1)$ fold.
    - Repeat the above for $k = 1, \ldots, K$ and store the computed accuracies.
2. Calculate the forecast accuracy by averaging over each validation accuracy computed in step 1.

A further reason for the validity of this approach is grounded in its direct derivation from how one would evaluate a time series model in the real world. If, at first, data is only available scarcely one has to work with what is at hand. As soon as more observations are available the model can be retrained

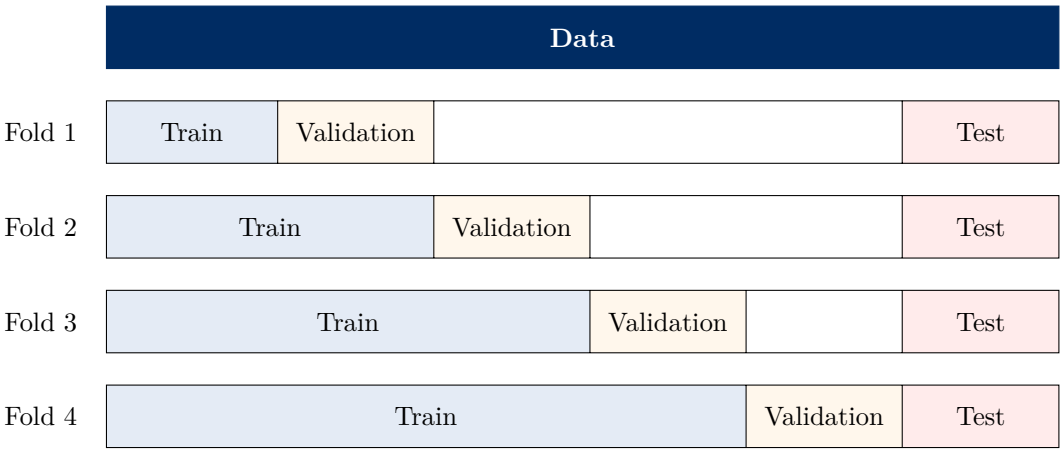

Figure 3.14: Illustration of 4-fold expanding window validation for time series data.

and tested on new data. Thus, the expanding window validation error reflects what would have been observed if the strategy was applied in the past.

Our training and validation set for this expanding window method is composed of observations from January 1st 2013 until May 31st 2017. Data for the test set, composed of observations ranging from June 1st 2017 to April 19th 2018, was shelved for later use and was not included in the validation stage (more on testing and strategy evaluation in the following Subsection). By splitting the validation and test data in mid-2017, we ensure that the immense Bitcoin price explosion (see Figure 2.1) is reflected in both, training and unseen test data, which likely increases the models' generalisation ability.

Even though we reduced the size of our grid-search already, looping though all 128 parameter combinations while applying the 4-fold expanding window validation requires an enormous amount of time. We employed a deep learning instance from Amazon Web Service to run this model selection process. The instance we chose is the one recommended in Chollet and Allaire (2018), titled p2.xlarge, which is equipped with one Nvidia Tesla K80, 4 virtual CPUs, and 61 gigabyte of RAM. Utilising the `R` package `keras` (see Allaire and Chollet (2018)), which is an interface to the deep learning library `tensorflow` (see

Abadi et al. (2015)), the total initialisation, training, and validation process still took 136.8 hours to complete on this setup.

3.2.2 Trading and Strategy Evaluation

Based on the model selection approach described above, we utilise the best performing models m with their respective hyperparameter combinations to predict the test set probabilities for the one-day-ahead Bitcoin price trend. These probabilities, denoted as $p_{t,m}$, are used to derive the predicted label $\hat{y}_{t,m}$, representing the forecasted Bitcoin trend. Formally, $\hat{y}_{t,m}$ is defined as:

$$\hat{y}_{t,m} = \begin{cases} 0, & \text{if } p_{t,m} < 0.5 - \omega, \\ 1, & \text{if } p_{t,m} > 0.5 + \omega, \\ \texttt{NA}, & \text{otherwise}, \end{cases} \tag{3.10}$$

where only probabilities are utilised that exceed a certain threshold controlled by the confidence parameter ω, similar to Amjad and Shah (2017). In our analysis, we set $\omega = 0.1$, which appears to be a reasonable trade-off between a higher signal certainty and a not dramatically reduced signal quantity.

Employing these predicted labels, we calculate a first measure of performance, the test accuracy of model m's forecast:

$$Acc_m = 1 - \frac{\hat{y}_m}{T - n_{\texttt{NA}}},$$

with

$$\hat{y}_m = \sum_{t=1}^{T} \begin{cases} 0, & \text{if } \hat{y}_{t,m} = \texttt{NA}, \\ |y_{t,\text{actual}} - \hat{y}_{t,m}|, & \text{otherwise}, \end{cases}$$

and $n_{\texttt{NA}}$ as the count of `NA`s in $\hat{y}_{t,m}$. Moreover, $y_{t,\text{actual}}$ is the actually observed Bitcoin trend at time t and T denotes the size of the test set.

Building on the predicted signal $\hat{y}_{t,m}$ from Equation (3.10) we create a return series r_m by simply going long in Bitcoin whenever the model predicts a 1 (= up trend), short whenever it predicts a 0 (= down trend), and hold cash whenever

it is unsure ($\hat{y}_{t,m} = \texttt{NA}$). More formally, the return from this long-short strategy implied by model m is:

$$r_{t,m} = \begin{cases} -r_{t,B}, & \text{if } \hat{y}_{t,m} = 0, \\ r_{t,B}, & \text{if } \hat{y}_{t,m} = 1, \\ 0, & \text{if } \hat{y}_{t,m} = \texttt{NA}, \end{cases} \tag{3.11}$$

where $r_{t,B}$ is the regular (non-logarithmic) return of Bitcoin at time t.

Besides the four machine learning algorithms we also employ two classical benchmark models, namely ARMA and momentum. We implement an ARMA(2,2) model via the `forecast` `R` package from Hyndman and Khandakar (2008), which determines the optimal lag structure by the small sample size corrected version of the Akaike Information Criteria (AICc). The momentum feature is constructed with a lookback l of 14 days, as described in Chapter 2. For both benchmarks Equation (3.11) simplifies to:

$$r_{t,m} = \begin{cases} -r_{t,B}, & \text{if } \hat{y}_{t,m} = 0, \\ r_{t,B}, & \text{if } \hat{y}_{t,m} = 1, \end{cases}$$

because no confidence parameter ω is applied.

It is noteworthy that in practice it is impossible to observe the closing price and trade on it at the same time. Therefore, in our trading strategy we implement the observation and signal creation process based on the closing price and the actual trading on the opening price of the following day.

The return $r_{t,m}$ is used in our second performance measure: the Sharpe ratio, developed by Sharpe (1966). We compute the annualised sharp ratio as:

$$S_m = \frac{\mathbb{E}[r_m - r_f]}{\sigma_m} \cdot \sqrt{365}, \tag{3.12}$$

where r_f is the daily risk-free rate[3], σ_m the daily volatility of the strategy given by model m and $\sqrt{365}$ the annualisation factor. The Sharpe ratio indicates

[3] As a proxy for the risk-free rate we use the daily federal funds rate.

how well an investor is compensated for the risk involved in a particular asset.

A third and last performance measure we will look at is Jensen's alpha. Introduced by Jensen (1968), it describes the abnormal return of an asset or portfolio above the theoretical expected return. This theoretical return is usually taken from a model describing a market return, like the capital asset pricing model, abbreviated to CAPM. According to the CAPM, every expected return of security i traded on the market should lie on the security market line (see, amongst others, Albrecht and Maurer (2008)):

$$\mathbb{E}[r_i] = r_f + \beta_i \left(\mathbb{E}[r_M] - r_f\right), \tag{3.13}$$

where r_M is the return of the market portfolio and r_f again the daily federal funds rate proxying the risk-free rate. The alpha of security i is then defined by the abnormal return not explained by Equation (3.13):

$$\alpha_i = \mathbb{E}[r_i] - \left[r_f + \beta_i \left(\mathbb{E}[r_M] - r_f\right)\right]. \tag{3.14}$$

In practice, this alpha is the intercept of a regression of the realised return r_i on $r_f + \beta_i (r_M - r_f)$. According to Nobel laureate Eugene Fama and the Efficient Market Hypothesis, introduced in its current form in Fama (1970) and which has been studied to great extend in many papers (see, amongst others, Basu (1977), Malkiel (2005), or Fama and French (2012)), financial markets are so efficient that repeatedly and consistently earning a positive alpha should be impossible. Nevertheless, famous hedge funds like Renaissance Technologies with their mysterious Medallion Fund earned significantly abnormal returns in the past, beating benchmark indices like the S&P500 consistently in a 13 year period between 2001 and 2013[4]. Hence, financial markets rather seem to be efficiently inefficient, meaning that enough active investors are compensated for their costs, but additional active investing is discouraged (see Pedersen (2015)).

One issue arises when calculating Equation (3.14) for the strategies in this

[4]See https://themarketmogul.com/easy-renaissance-technologys-medallion-fund/, retrieved 11/05/2018.

work: How to define the market? On the one hand, just taking a typical benchmark for equity strategies like the MSCI World Index is surely not enough, since this only accounts for the equity market. On the other hand, constructing a market capitalisation weighted portfolio only consisting of cryptocurrencies would not reflect the diversification and hedging potential of Bitcoin, documented in Brière et al. (2015). To account for these aspects we use both market portfolios just mentioned. One that reflects the global equity market capitalisation, namely the MSCI World Index, and one that represent the market of cryptocurrencies, for which we are using the CRIX, short for CRyptocurrency IndeX, developed and maintained by the Ladislaus von Bortkiewicz Chair of Statistics at Humboldt University Berlin. This market cap weighted index consists only of liquid cryptocurrencies exceeding a certain market capitalisation (see Trimborn and Härdle (2016)). The respective alphas for the model m will be called $\alpha_{m,\text{MSCI}}$ and $\alpha_{m,\text{CRIX}}$.

Chapter 4

Results

In this chapter we explore the results from our analysis of Bitcoin price trends with the methods previously described. Whereas the first section compares prediction accuracies between strategies, the second part analyses their respective returns based on the Sharpe ratio and Jensen's alpha as typical performance measures.

4.1 Prediction Accuracy

Using the combined filter and wrapper approach for model selection documented in the last chapter, we determine the optimal configuration for each of the four machine learning algorithms previously described in the *Theory* Section. Table 4.1 displays the best performing features for each algorithm. Interestingly, the expanding window validation yields an almost identical feature selection for three out of four models, namely for the logistic regression, fully-connected neural network, and the convolutional neural network. One could conclude that these features hold some predictive power for the one-day-ahead forecast of the Bitcoin price trend. In contrast, the optimal configuration for the GRU neural network consists of only the Bitcoin return. This might point in the opposite direction of other variables not helping at all in explaining the next-day return. Another plausible explanation is that the algorithm was unable to learn constructive patterns from these additional features due to the relative small size of the training set. Nonetheless, up to this point

it is rather unclear and might be algorithm dependent whether more or less predictors are needed for good forecasting performance.

Model	Features chosen
Logistic regression	`btc close return`, `btc open return`, `btc high return`, `btc low return`, `btc volume`, `eth close return`, `xrp close return`, `ltc close return`
Fully-connected neural network	`btc close return`, `btc open return`, `btc high return`, `btc low return`, `btc volume`, `eth close return`, `xrp close return`, `ltc close return`
GRU neural network	`btc close return`
Convolutional Neural Network	`btc close return`, `btc open return`, `btc high return`, `btc low return`, `btc volume`, `eth close return`, `xrp close return`, `ltc close return`, `momentum`

Table 4.1: Chosen features via the model selection approach documented in Chapter 3, Subsection *Model Selection*.

During the same model selection process we also pin down the two remaining hyperparameters, the lookback period and the amount of neurons in the hidden layer, visualised in Table 4.2. On top of the four algorithms and their corresponding selected features, Table 4.2 also shows the set of hyperparameters for two additional feature configurations. The first, named *Autoregressive*, is the straightforward case of only choosing lagged values of the Bitcoin return as explanatory variable (coinciding with what was chosen for the GRU neural network via grid-search), whereas the second configuration uses all features available, which we call the *kitchen sink combination*, and is likely prone to overfit the training set due to its great dimensionality (see Barreto and Howland (2005)). The set of hyperparameters of these supplementary configurations were chosen based on the expanding window validation accuracy of the fully-connected neural network. We add these two models to have a

benchmark for our neural networks to see if the additional or left out features actually improve the models' performance.

Model	Lookback period	Neuron count
Logistic regression	14	–
Fully-connected neural network	28	16
GRU neural network	28	32
Convolutional neural network	14	32
Autoregressive	28	32
All features	28	16

Table 4.2: Chosen Lookback period & Neuron count via the model selection approach documented in Chapter 3, Subsection *Model Selection*.

Employing the features and hyperparameters documented for each algorithm in Table 4.1 and 4.2, we trained our models with their respective optimal configuration on the whole training set ranging from the start of 2013 until the 31st of May, 2017. Afterwards, we evaluated their performance on the test set. The resulting training and test accuracies of the combinations chosen during the model selection process as well as the accuracies of the two classical benchmark models, ARMA and momentum, are reported in Table 4.3. To make the comparison easier we also added a row describing the accuracy of simply holding Bitcoin during the training and test period, i.e. these values reflect the mean of the Bitcoin trend during these periods.

Looking at these accuracies a few interesting observations can be made:

- The logistic regression performs remarkably well for such a computationally cheap approach, beating the fully-connected neural network as well as the simple long strategy by a 5 percentage points higher test accuracy.

- Regardless of the implemented dropout rate, the fully-connected neural network seems to struggle with immense overfitting, as the training accuracy is 100%, but the model is unable to generalise and even beat the

Model	Training accuracy	Test accuracy
Logistic regression	0.69	0.57
Fully-connected neural network	1.00	0.52
GRU neural network	0.70	0.62
Convolutional neural network	0.79	0.59
ARMA(2,2)	0.54	0.52
Momentum	0.60	0.62
Long Bitcoin	0.55	0.52

Table 4.3: Training and test accuracies of the combinations selected by the filter and wrapper approach, the accuracies of the classical methods, and the accuracy of going long all the time.

long strategy's test accuracy.

- The GRU neural network and hence, the one relying only on the Bitcoin return and nothing else, generalises the best and achieves the leading test accuracy of 62% among all machine learning approaches.
- As a relatively close second the convolutional neural network offers a cheaper to compute alternative to the more expansive GRU neural network.
- As could have been expected, the univariate ARMA is inadequate in predicting the Bitcoin return, which is likely a result of its simplicity and unrealistic assumptions, e.g. that only linear relations exist in the data and that the volatility is homoskedastic.
- An intriguing surprise is presented by the performance of the momentum strategy, offering a substitute for the best performing neural network at a fraction of the computational power needed.
- The excellent performance of the GRU neural network and the momentum strategy could hint towards the previously mentioned idea that

additional features do not help in predicting the Bitcoin price and are thus not relevant price drivers.

The accuracies for the two benchmark feature combinations, autoregressive and kitchen sink, are shown in Table 4.4 and 4.5 (trained and tested on the same data as above). They can be viewed as sanity checks to identify whether adding features to the models actually helps explaining the Bitcoin trend or whether overfitting occurs when too many features are added. Both checks seem to be fulfilled. On the one side, the test accuracy actually increases for all models[1] when the best performing features, selected by the expanding window validation, are added (compare Table 4.3 and 4.4). On the other side, the training accuracy hits 100% coupled with a lower test accuracy when all features are included (compare Table 4.3 and 4.5).

Model	Training accuracy	Test accuracy
Logistic regression	0.65	0.53
Fully-connected neural network	0.91	0.51
GRU neural network	0.70	0.62
Convolutional neural network	0.89	0.49

Table 4.4: Training and test accuracies of the Autoregressive models (the only feature is the Bitcoin return).

4.2 Strategy Returns

In this section we turn our attention to the returns resulting from each strategy discussed before and evaluate the performance measures described in the previous chapter. Table 4.6 reports the daily mean and volatility of these regular (non-logarithmic) returns on the test set. Here, one can see the effect of the confidence parameter ω, as implemented in Equation 3.10. Without this parameter, all models should have a similar relatively high standard deviation

[1]Except for the GRU model, since it represents the optimal configuration already.

Model	Training accuracy	Test accuracy
Logistic regression	1.00	0.52
Fully-connected neural network	1.00	0.48
GRU neural network	1.00	0.49
Convolutional neural network	1.00	0.48

Table 4.5: Training and test accuracies of the kitchen sink models (all features are incorporated).

because all return series are based on the same underlying asset, do not include leverage and would be achieved by trading every period. However, ω ensures that in uncertain periods, where the resulting thresholds ($0.5+\omega$ and $0.5-\omega$) are neither exceeded nor undercut, no trading occurs, which intuitively reduces the strategies' volatility. Looking at Table 4.6 it can be seen that all algorithms but the fully-connected neural network exhibit a reasonably low volatility and high mean return, indicating the successful usage of ω, causing less frequent trading but higher certainty in their predictions. In contrast, the high volatility of the fully-connected neural network shows that the model traded more frequently (almost as frequent as the benchmark models), suggesting its supposedly high prediction certainty. Albeit, the meagre mean return proves the model wrong, which is another piece of evidence for its present overfitting.

Evaluating these returns over time by investing 1 USD at the start of the test set period, makes a compelling comparison between the models and their respective strategies easy. To account for the different strategy return volatilities caused by the confidence parameter ω, we normalise the standard deviation of every return series to 1% per day. Figure 4.1 illustrates the development of 1 USD invested in the four strategies based on the machine learning algorithms and the Bitcoin long as a benchmark. As expected from Table 4.3, the GRU neural network ends up on top, but with the convolutional neural network as a close second. Even though the accuracies of the other two algorithms were at least on par with the one from the Bitcoin long strategy, they are unable to benefit from these accuracies here. This is also a concern for the two "win-

Model	Mean	Standard deviation
Logistic regression	0.0030	0.0452
Fully-connected neural network	0.0001	0.0548
GRU neural network	0.0051	0.0308
Convolutional neural network	0.0060	0.0433
ARMA	0.0035	0.0573
Momentum	0.0128	0.0559

Table 4.6: Daily mean and standard deviation of the returns on the test set for each model.

ning" models, as they are only able to pull away when the Bitcoin price took a downturn in early 2018.

In Figure 4.2 the same comparison is conducted for the two classical methods, ARMA and momentum. On the one hand, this plot reveals that the already intriguing accuracy of the momentum strategy turns into a phenomenal return series, beating the Bitcoin long benchmark by a safe distance. On the other hand, the accrued return of the ARMA model is as previously expected unable to deliver a performance surpassing the benchmark.

Finally, Figure 4.3[2] depicts the two best performing models to make the already apparent difference clearer. Despite the GRU neural network and momentum having similar accuracies, the momentum strategy is able to outshine the much more computationally expansive machine learning model by quite a margin.

A similar observation can be made from Table 4.7, which documents the Sharpe ratios calculated by Equation (3.12). Unsurprisingly, the momentum strategy again dominates among all models and the fully-connected neural network is unable to achieve a Sharpe ratio much greater than zero due to its small mean return.

[2]Figure C.1 in the Appendix C illustrates the same strategies with non adjusted returns as shown in Figure 4.3 to show that not using volatility adjusted returns when plotting the returns of trading strategies distorts the difference between the models.

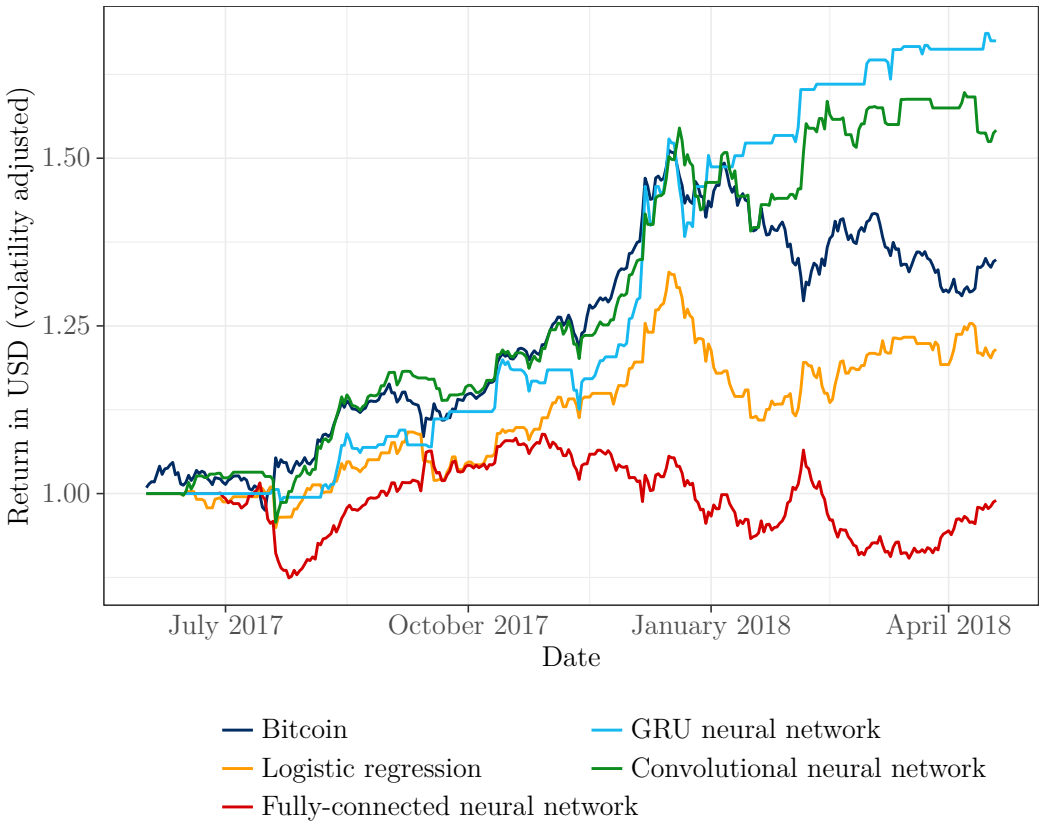

Figure 4.1: Development of the investment of 1 USD for the four machine learning approaches. Bitcoin price development as a benchmark.

The last performance measure analysed in this work is Jensen's alpha. Table 4.8 reports the CAPM regression coefficients (see Equations (3.13) and (3.14)) and their respective standard errors (in parentheses) and significance levels. As expected from the previous results, the GRU and convolutional neural network along with the momentum strategy achieve remarkable results. All three have a positive, significant alpha with regards to both, the MSCI World Index (α_{MSCI}) and the CRIX (α_{CRIX}). Furthermore, their beta with respect to the global equity market, β_{MSCI}, is insignificant, indicating distinct hedging potential. Another interesting observation is that the β_{CRIX} is significant for all machine learning models, implying that at least parts of their return can be

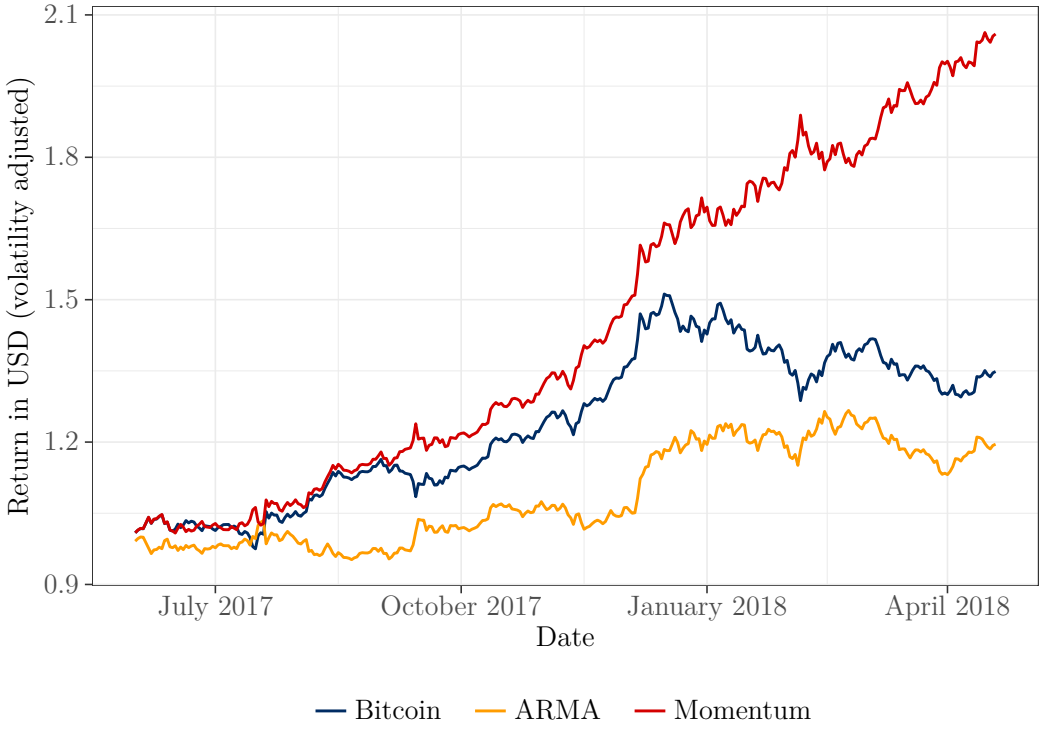

Figure 4.2: Development of the investment of 1 USD for the two classical methods, ARMA and momentum. Bitcoin price development as a benchmark.

explained by this cryptocurrency index, which might cause troubles for these strategies when the whole cryptocurrency market crashes. This illustrates another strength of the momentum strategy: there appears to be no linear relationship between the cryptocurrency market and the strategy, indicating that profits can likely still be made during a falling market.

Considering Figure 4.3 and Table 4.6, 4.7, and 4.8 a clear winner of this analysis can be seen: the momentum strategy. Despite it being a rather cheap to compute model it was able to beat the much more expansive machine learning models by quite a comfortable margin. Possible reasons for this result are manifold:

- Whereas the momentum strategy essentially only requires data from the

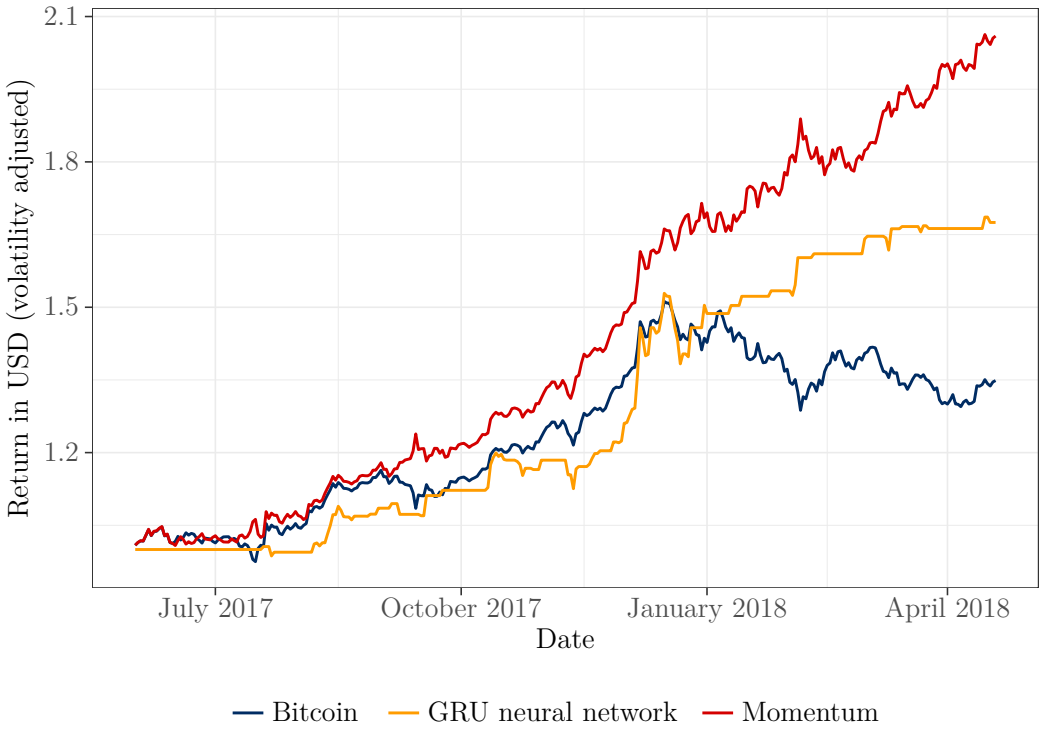

Figure 4.3: Development of the investment of 1 USD for the two best performing methods, GRU neural network and momentum. Bitcoin price development as a benchmark.

lookback period to compute a reliable signal, machine learning algorithms tend to perform badly when the amount of training data is insufficient. We suspect that, given enough data, neural networks are able to surpass such a relatively simple trend following strategy, as the momentum characteristics themselves could be learned and further be optimised from the data. More data would also empower these algorithms to leverage possible patterns present in other variables that the momentum strategy has no access to.

- The existence of significant positive returns of the momentum strategy hints towards the fact that certain behavioural mechanics exist in the

Model	Sharpe ratio
Logistic regression	1.23
Fully-connected neural network	0.02
GRU neural network	3.13
Convolutional neural network	2.64
ARMA	1.14
Momentum	4.36
Bitcoin long	1.85

Table 4.7: Annualised Sharpe ratios based on the returns for each model and the daily federal fund rate as risk-free asset.

Bitcoin market[3]. It is quite likely that a good chunk of these behavioural tendencies can be perfectly explained by social media data, for example in the form of a news sentiment analysis. Even though we analysed these in a separate feature configuration, this examination suffered once again from the fact that our dataset is rather small. Burdening it with also making sense of more than 500 added features (the combined headline vectors), in addition to predicting the Bitcoin trend, simply leads to extreme overfitting of the training data.

- Owing to the confidence parameter ω, there are periods where the machine learning algorithms hold cash instead of a long or short position in Bitcoin. It might be a feasible approach to combine the different strategies such that the algorithms keep trading using the momentum signal for these periods of uncertainty. At the very least, holding the risk-free asset would improve the performance without adding additional volatility[4].

[3]See the lists on initial underreaction and delayed overreaction in Subsection *Trend Following Strategy (Momentum)*, Chapter 3 for possible explanations for these behavioural mechanics.

[4]This is only true in theory, since in reality there is no such thing as a truly risk-free rate. All

Model	α_{MSCI}	β_{MSCI}	α_{CRIX}	β_{CRIX}
Logistic regression	0.0035	−1.1867***	0.0023	0.1119**
	(0.0025)	(0.4358)	(0.0025)	(0.0454)
Fully-connected neural network	0.0004	−0.6680	−0.0007	0.1507***
	(0.0031)	(0.5338)	(0.0030)	(0.0550)
GRU neural network	0.0052***	−0.2793	0.0040**	0.1911***
	(0.0017)	(0.3001)	(0.0016)	(0.0294)
Convolutional neural network	0.0063***	−0.6231	0.0052**	0.1465***
	(0.0024)	(0.4209)	(0.0024)	(0.0432)
ARMA	0.0036	−0.4237	0.0030	0.0798
	(0.0032)	(0.5585)	(0.0032)	(0.0580)
Momentum	0.0128***	−0.0047	0.0127***	0.0056
	(0.0031)	(0.5458)	(0.0031)	(0.0568)
Bitcoin long	0.0051	0.9089	0.0023	0.6211***
	(0.0032)	(0.5550)	(0.0026)	(0.0465)
Note:				*p<0.1; **p<0.05; ***p<0.01

Table 4.8: Alpha (daily) and beta values for each model. The daily federal fund rate was used as the risk-free asset. Standard errors in parenthesis.

- We only validated and tuned a certain limited range of hyperparameters. Maybe the neural networks we looked at were to shallow and a much deeper structure[5] is required to find patterns that generalise better. Or in general, other combinations of hyperparameters we did not test, e.g. a longer lookback period, a different certainty threshold, or a longer or shorter training time is needed. However, the main obstacle hindering us

available proxies, including the federal funds rate, exhibit additional risks, e.g. overnight counterparty credit risks. However, analysing the advantages and disadvantages of these proxies would fill another entire thesis, and thus are ignored in this work.

[5]The *deepness* of a neural network describes how many layers and neurons were used.

from testing for all of these possible combinations is our lack of available computational power, which could easily be solved by any large research institution or company.

- To mimic the way a momentum strategy "learns", i.e. only using the newest data, our machine learning algorithms could be retrained prior to predicting every test label using all observations that occurred before. This would require enormous amounts of processing power, as for every timestep in the test set new training is necessary. Despite this flaw, this method is quite applicable in practice, considering that training a neural network only once a day is not too demanding for professional investors.

Nevertheless, we think that the performance of the GRU and convolutional neural network is still tremendously good, even if a bit overshadowed by the momentum strategy.

Chapter 5

Conclusion

In this thesis we study to which extent modern machine learning algorithms and classical time series methods are capable of predicting the one-day-ahead price trend of the popular Bitcoin cryptocurrency. More specifically, we address this forecasting task by implementing four machine learning classifiers. These include the logistic regression model and three neural network architectures, namely the fully-connected-, GRU-, and convolutional neural network. While the former algorithm belongs to the class of computationally efficient generalised linear models, the latter three represent state of the art deep learning methods, able to learn highly nonlinear patterns. We compare their results with the prediction performances of the classical ARMA model and the trend following momentum strategy, which are equally popular in their respective fields of time series econometrics and finance.

Taking into account that an algorithm's forecasting accuracy inevitably depends on the amount and quality of training data, we analyse a carefully selected set of 25 features, which we suppose to have predictive power with regards to the Bitcoin price. These features include blockchain related market forces of supply and demand, global macroeconomic and financial development indicators and competing cryptocurrency market price data. Furthermore, we utilise attractiveness measures like Google Trends and Wikipedia Pageviews and additionally use natural language processing to create a Google News feature, which ideally captures the market sentiment concerning Bitcoin.

Due to the great amount of potential feature combinations and tunable hyperparameter values we employ a combined filter and wrapper approach to

identify the optimal model configuration for each algorithm. Our training period ranges from the 1st of January 2013 to the 31st of May 2017. To avoid overfitting on the training set we apply 4-fold expanding window validation inside the wrapper's grid-search. In contrast to cross-validation, this validation approach is suitable for time series data because it considers the data's intertemporal dependencies. The best performing configurations of each algorithm in the model selection process are then trained on the whole training set and evaluated on the test set, ranging from the 1st of June 2017 to the 19th of April 2018.

We find that the best GRU setup achieves a remarkable test accuracy of 62% in predicting the next-day Bitcoin price trend, followed by the convolutional neural network and logistic regression, which attain 59% and 57%, respectively. With a training accuracy of 100% but a test accuracy of only 52%, the fully-connected neural network ranks last and does not generalise well to unseen data due to overfitting. The ARMA model performs equally poor and thus, is like the fully-connected network not able to surpass the simple strategy of going long in Bitcoin, which also yields 52% prediction accuracy. However, all these results are excelled by the computationally cheap and intriguingly simple momentum strategy with a lookback of 14 days. It equalises the GRU network's test accuracy of 62%, but achieves better performance with regards to its annualised Sharpe ratio and Jensen's alpha.

This outcome was a bit unexpected and is particularly remarkable considering the simplicity of momentum. Although, the observation that the best GRU configuration includes only the autoregressive Bitcoin return as explanatory variable, hints to the possibility of simpler model setups having better chances in successfully predicting the price trend. Another potential explanation for the relative weak performance of the sophisticated neural networks compared to momentum might be the lack of sufficiently large training data. We suspect that, given enough data, neural networks should be able to surpass simple trend following strategies, as the momentum characteristics themselves could be learned and more patterns from other variables could be leveraged.

On top of the above results, when reflecting on our work, we are aware

of certain more or less severe limitations concerning this thesis, which leave plenty of space for future studies, e.g. as part of a Ph.D. A first and obvious simplification is the exclusion of any transaction costs, making any reported returns infeasible to reproduce in the real world. On the one hand, these transaction costs can be of financial nature, i.e. the ones occurring when actually buying Bitcoins, like the transaction fee, which is mostly comprised of the revenue the Bitcoin miners make, the bid-ask spread, or for large orders the market impact cost (see also Pedersen (2015)). On the other hand, these costs can also be of more practical nature, e.g. the time it takes for the transaction to be validated. Theoretically, machine learning algorithms are able to implement transaction costs into their optimisation problem, but this would increase the complexity of the task at hand greatly, as a completely different objective, i.e. the minimisation of trades made, has to be included.

As documented in Kim et al. (2016), we may achieve better results with our machine learning approaches if we do not predict the one-day-ahead Bitcoin trend but the trend at a later point in time (e.g. greater forecast horizon). Whatever the case may be, finding an optimal forecast horizon would add another tunable parameter to the grid-search, which increases the required training time.

Furthermore, it might be feasible to use a higher granularity for the input data, as variables like the Bitcoin price are readily available as tick data. However, this causes issues for variables unrelated to cryptocurrencies, e.g. the MSCI World Index or Google News data, as these have either a closing time or simply do not occur as often. Moreover, working with tick data introduces a plethora of other problems like handling values *ticking* at different times and the question of what to do when no ticks take place. An option that possibly solves these issues is to change the way data is processed in the models. We use a method called *batch learning*, meaning that data is processed in chunks and reused many times in the training process. In contrast, *online machine learning* iterates over the complete dataset only once and uses every observation independently. This cuts short the training time immensely and would make using a higher granularity more reasonable.

Similar to this limitation, one could also include additional features and test for their predictive power, like the analysis of Twitter data, as done in Matta et al. (2015), or other macroeconomic indicators. Although, finding and testing for these determinants could and did fill whole research papers, as discussed in Chapter 1, Section *Related Literature*.

Since we only tested two classes of machine learning methods, namely a generalised linear model (logistic regression) and several neural network architectures, it might be that other well known algorithms like *random forests* or *support vector machines* are able to better predict the Bitcoin trend. Evidence for the exceptional performance of a random forest applied to the forecasting of the Bitcoin trend was found in Amjad and Shah (2017), which could indicate that this algorithm is able to surpass our GRU neural network or the momentum strategy.

Lastly, one could also introduce a model that *averages* over the predictions of previously selected models and thus, tries to get the best out of all worlds. A method to achieve such an *ensemble* model is *Bayesian model averaging*, described in Hoeting et al. (1999), which also accounts for model uncertainty. Wright (2008) had considerable success using this method to forecast exchange rates and Avramov (2002) also observed favourable outcomes when applying it to the prediction of stock returns.

All in all, we think that future studies should and, given the increasing popularity of cryptocurrencies, will be conducted with regards to the determinants and possible predictors of Bitcoin and other coins. Nevertheless, in our opinion the results and interpretations presented in this thesis shed some light on what drives the returns of the new and exciting asset Bitcoin. They demonstrate that there exist strategies resulting in substantial positive performances, opening doors to untapped hedging potential and remarkable returns.

Bibliography

Literature

[1] Abu-Mostafa, Yaser S, Magdon-Ismail, Malik and Lin, Hsuan-Tien. *Learning from data*. Vol. 4. AMLBook New York, NY, USA, 2012.

[2] Akaike, Hirotugu. "A new look at the statistical model identification". In: *IEEE transactions on automatic control* 19.6 (1974), pp. 716–723.

[3] Albrecht, Peter and Maurer, Raimond. *Investment-und Risikomanagement*. Schäffer Poeschel, 2008.

[4] Amjad, Muhammad and Shah, Devavrat. "Trading Bitcoin and Online Time Series Prediction". In: *NIPS 2016 Time Series Workshop*. 2017, pp. 1–15.

[5] Avramov, Doron. "Stock return predictability and model uncertainty". In: *Journal of Financial Economics* 64.3 (2002), pp. 423–458.

[6] Barberis, Nicholas, Shleifer, Andrei and Vishny, Robert. "A model of investor sentiment". In: *Journal of financial economics* 49.3 (1998), pp. 307–343.

[7] Barreto, Humberto and Howland, Frank. *Introductory econometrics: using Monte Carlo simulation with Microsoft excel*. Cambridge University Press, 2005.

[8] Basu, Sanjoy. "Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis". In: *The journal of Finance* 32.3 (1977), pp. 663–682.

[9] Bengio, Yoshua, Simard, Patrice and Frasconi, Paolo. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[10] Bikhchandani, Sushil, Hirshleifer, David and Welch, Ivo. "A theory of fads, fashion, custom, and cultural change as informational cascades". In: *Journal of political Economy* 100.5 (1992), pp. 992–1026.

[11] Brière, Marie, Oosterlinck, Kim and Szafarz, Ariane. "Virtual currency, tangible return: Portfolio diversification with bitcoin". In: *Journal of Asset Management* 16.6 (2015), pp. 365–373.

[12] Cho, Kyunghyun et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[13] Chollet, François and Allaire, J.J. *Deep Learning with R*. Manning Publications, 2018.

[14] Chung, Junyoung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[15] Ciaian, Pavel, Rajcaniova, Miroslava and Kancs, d'Artis. "The economics of BitCoin price formation". In: *Applied Economics* 48.19 (2016), pp. 1799–1815.

[16] Cox, David R. "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), pp. 215–242.

[17] Daniel, Kent D, Hirshleifer, David A and Subrahmanyam, Avanidhar. "A theory of overconfidence, self-attribution, and security market under-and over-reactions". In: *Journal of Finance* 53 (1998), 1839–1885.

[18] Dauphin, Yann N et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[19] Dickey, David A and Fuller, Wayne A. "Distribution of the estimators for autoregressive time series with a unit root". In: *Journal of the American statistical association* 74.366a (1979), pp. 427–431.

[20] Dougherty, Christopher. *Introduction to econometrics*. Oxford University, 2002.

[21] Duffie, Darrell. "Presidential address: asset price dynamics with slow-moving capital". In: *The Journal of finance* 65.4 (2010), pp. 1237–1267.

[22] Fama, Eugene F. "Efficient capital markets: A review of theory and empirical work". In: *The journal of Finance* 25.2 (1970), pp. 383–417.

[23] Fama, Eugene F and French, Kenneth R. "Size, value, and momentum in international stock returns". In: *Journal of financial economics* 105.3 (2012), pp. 457–472.

[24] Frazzini, Andrea. "The disposition effect and underreaction to news". In: *The Journal of Finance* 61.4 (2006), pp. 2017–2046.

[25] Friedman, Jerome, Hastie, Trevor and Tibshirani, Robert. *The elements of statistical learning*. Second. Springer series in statistics New York, 2009.

[26] Garcia, David and Schweitzer, Frank. "Social signals and algorithmic trading of Bitcoin". In: *Royal Society open science* 2.9 (2015), p. 150288.

[27] Garcia, David et al. "The digital traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin economy". In: *Journal of the Royal Society Interface* 11.99 (2014), p. 20140623.

[28] Greaves, Alex and Au, Benjamin. *Using the Bitcoin Transaction Graph to Predict the Price of Bitcoin*. 2015.

[29] Grudnitski, Gary and Osburn, Larry. "Forecasting S&P and gold futures prices: An application of neural networks". In: *Journal of Futures Markets* 13.6 (1993), pp. 631–643.

[30] Gupta, Dishashree. *Fundamentals of Deep Learning – Activation Functions and When to Use Them?* 23rd Oct. 2017. URL: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/ (visited on 25/04/2018).

[31] Harris, Zellig S. "Distributional structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[32] Hochreiter, Sepp and Schmidhuber, Jürgen. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[33] Hoeting, Jennifer A et al. "Bayesian model averaging: a tutorial". In: *Statistical science* (1999), pp. 382–401.

[34] Hong, KiHoon. "Bitcoin as an alternative investment vehicle". In: *Information Technology and Management* 18.4 (2017), pp. 265–275.

[35] Hurst, Brian, Ooi, Yao Hua and Pedersen, Lasse Heje. "A Century of Evidence on Trend-Following Investing". In: *The Journal of Portfolio Management* 44.1 (2017), pp. 15–29.

[36] Hyndman, Rob J and Athanasopoulos, George. *Forecasting: principles and practice*. OTexts, 2014.

[37] Jang, Huisu and Lee, Jaewook. "An Empirical Study on Modeling and Prediction of Bitcoin Prices With Bayesian Neural Networks Based on Blockchain Information". In: *IEEE Access* 6 (2018), pp. 5427–5437.

[38] Jensen, Michael C. "The performance of mutual funds in the period 1945–1964". In: *The Journal of finance* 23.2 (1968), pp. 389–416.

[39] Kallenberg, Michiel et al. "Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring". In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1322–1331.

[40] Kim, Young Bin et al. "Predicting fluctuations in cryptocurrency transactions based on user comments and replies". In: *PloS one* 11.8 (2016), pp. 1–17.

[41] Kingma, Diederik P. and Ba, Jimmy. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[42] Kristoufek, Ladislav. "BitCoin meets Google Trends and Wikipedia: Quantifying the relationship between phenomena of the Internet era". In: *Scientific reports* 3 (2013), p. 3415.

[43] Kristoufek, Ladislav. "What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis". In: *PloS one* 10.4 (2015), e0123923.

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

[45] Madan, Isaac, Saluja, Shaurya and Zhao, Aojia. *Automated Bitcoin Trading via Machine Learning Algorithms*. 2015.

[46] Malkiel, Burton G. "Reflections on the efficient market hypothesis: 30 years later". In: *Financial Review* 40.1 (2005), pp. 1–9.

[47] Matta, Martina, Lunesu, Ilaria and Marchesi, Michele. "Bitcoin Spread Prediction Using Social and Web Search Media." In: *UMAP Workshops*. 2015.

[48] McCulloch, Warren S and Pitts, Walter. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[49] McNally, Sean. *Predicting the price of Bitcoin using Machine Learning*. 2016.

[50] Mitchell, Mark, Pedersen, Lasse Heje and Pulvino, Todd. "Slow moving capital". In: *American Economic Review* 97.2 (2007), pp. 215–220.

[51] Moskowitz, Tobias J, Ooi, Yao Hua and Pedersen, Lasse Heje. "Time series momentum". In: *Journal of financial economics* 104.2 (2012), pp. 228–250.

[52] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).

[53] Pedersen, Lasse Heje. *Efficiently inefficient: how smart money invests and market prices are determined*. Princeton University Press, 2015.

[54] Rohrbach, Janick, Suremann, Siran and Osterrieder, Joerg. *Momentum and Trend Following Trading Strategies for Currencies Revisited-Combining Academia and Industry*. 2017.

[55] Ruder, Sebastian. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[56] Rumelhart, David E, Hinton, Geoffrey E and Williams, Ronald J. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), p. 533.

[57] Schwarz, Gideon et al. "Estimating the dimension of a model". In: *The annals of statistics* 6.2 (1978), pp. 461–464.

[58] Shah, Devavrat and Zhang, Kang. "Bayesian regression and Bitcoin". In: *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*. IEEE. 2014, pp. 409–414.

[59] Sharpe, William F. "Mutual fund performance". In: *The Journal of business* 39.1 (1966), pp. 119–138.

[60] Shefrin, Hersh and Statman, Meir. "The disposition to sell winners too early and ride losers too long: Theory and evidence". In: *The Journal of finance* 40.3 (1985), pp. 777–790.

[61] Silber, William L. "Technical trading: when it works and when it doesn't". In: *The Journal of Derivatives* 1.3 (1994), pp. 39–44.

[62] Simonyan, Karen and Zisserman, Andrew. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[63] Srivastava, Nitish et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[64] Torres, Douglas Garcia and Qiu, Hongliang. "Applying Recurrent Neural Networks for Multivariate Time Series Forecasting of Volatile Financial Data". In: (2018).

[65] Trimborn, Simon and Härdle, Wolfgang K. "CRIX an Index for blockchain based Currencies". In: *SFB 649 Discussion Paper* (2016).

[66] Tsanas, Athanasios et al. "Novel speech signal processing algorithms for high-accuracy classification of Parkinson's disease". In: *IEEE Transactions on biomedical engineering* 59.5 (2012), pp. 1264–1271.

[67] Tsay, Ruey S. *Analysis of financial time series*. Third. Vol. 543. John Wiley & Sons, 2010.

[68] Turing, Alan Mathison. "Computing machinery and intelligence". In: *Mind* 59.236 (1950), pp. 433–460.

[69] Tversky, Amos and Kahneman, Daniel. "Judgment under uncertainty: Heuristics and biases". In: *science* 185.4157 (1974), pp. 1124–1131.

[70] Wason, Peter C. "On the failure to eliminate hypotheses in a conceptual task". In: *Quarterly journal of experimental psychology* 12.3 (1960), pp. 129–140.

[71] Welch, Ivo. "Herding among security analysts". In: *Journal of Financial economics* 58.3 (2000), pp. 369–396.

[72] Wright, Jonathan H. "Bayesian model averaging and exchange rate forecasts". In: *Journal of Econometrics* 146.2 (2008), pp. 329–341.

[73] Xu, Bing et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

R-Packages

[74] Abadi, Martín et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[75] Allaire, JJ and Chollet, François. *keras: R Interface to 'Keras'*. R package version 2.1.4. 2018. URL: https://CRAN.R-project.org/package=keras.

[76] Crone, Sven F and Kourentzes, Nikolaos. "Feature selection for time series prediction–A combined filter and wrapper approach for neural networks". In: *Neurocomputing* 73.10-12 (2010), pp. 1923–1936.

[77] Dahl, David B. *xtable: Export Tables to LaTeX or HTML*. R package version 1.8-2. 2016. URL: https://CRAN.R-project.org/package=xtable.

[78] Feinerer, Ingo, Hornik, Kurt and Meyer, David. "Text Mining Infrastructure in R". In: *Journal of Statistical Software* 25.5 (2008), pp. 1–54. URL: http://www.jstatsoft.org/v25/i05/.

[79] Hyndman, Rob J and Khandakar, Yeasmin. "Automatic time series forecasting: the forecast package for R". In: *Journal of Statistical Software* 26.3 (2008), pp. 1–22. URL: http://www.jstatsoft.org/article/view/v027i03.

[80] Keyes, Oliver and Lewis, Jeremiah. *pageviews: An API Client for Wikimedia Traffic Data*. R package version 0.3.0. 2016. URL: https://CRAN.R-project.org/package=pageviews.

[81] Massicotte, Philippe and Eddelbuettel, Dirk. *gtrendsR: Perform and Display Google Trends Queries*. R package version 1.4.1. 2018. URL: https://CRAN.R-project.org/package=gtrendsR.

[82] Mikolov, Tomas et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[83] Ooms, Jeroen. "The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects". In: *arXiv:1403.2805 [stat.CO]* (2014). URL: https://arxiv.org/abs/1403.2805.

[84] Pennington, Jeffrey, Socher, Richard and Manning, Christopher. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[85] Pfaff, Bernhard. "VAR, SVAR and SVEC Models: Implementation Within R Package vars". In: *Journal of Statistical Software* 27.4 (2008). URL: http://www.jstatsoft.org/v27/i04/.

[86] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2017. URL: https://www.R-project.org/.

[87] Raymond McTaggart, Gergely Daroczi and Clement Leung. *Quandl: API Wrapper for Quandl.com*. R package version 2.8.0. 2016. URL: https://CRAN.R-project.org/package=Quandl.

[88] Rinker, Tyler W. *textstem: Tools for stemming and lemmatizing text*. version 0.1.3. Buffalo, New York, 2017. URL: http://github.com/trinker/textstem.

[89] Rinker, Tyler W. *textclean: Text Cleaning Tools*. version 0.6.3. Buffalo, New York, 2018. URL: https://github.com/trinker/textclean.

[90] Ryan, Jeffrey A. and Ulrich, Joshua M. *quantmod: Quantitative Financial Modelling Framework*. R package version 0.4-12. 2017. URL: https://CRAN.R-project.org/package=quantmod.

[91] Ryan, Jeffrey A. and Ulrich, Joshua M. *xts: eXtensible Time Series*. R package version 0.10-1. 2017. URL: https://CRAN.R-project.org/package=xts.

[92] Selivanov, Dmitriy and Wang, Qing. *text2vec: Modern Text Mining Framework for R*. R package version 0.5.1. 2018. URL: https://CRAN.R-project.org/package=text2vec.

[93] Wickham, Hadley. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN: 978-0-387-98140-6. URL: http://ggplot2.org.

[94] Wickham, Hadley. *rvest: Easily Harvest (Scrape) Web Pages*. R package version 0.3.2. 2016. URL: https://CRAN.R-project.org/package=rvest.

Appendix

A Hash Function

A.1 Definition

A hash function is a mathematical algorithm that maps input data of variable size (e.g. a text message) to a fixed-size alphanumeric output value (a hash). Its use for the blockchain and cryptography in general results from five main properties:

1. A hash is **deterministic** given the input data, which means that the same message will always result in the same hash.
2. A hash is **unique**, which means that different input data will always result in different hashes.
3. A hash **changes significantly** (no visible correlations between old and new hash), when making only minor alterations to the message
4. A hash is **fast to compute**.
5. Given a hash, it is **infeasible to compute** the corresponding message, except by trying all possible messages (brute-force attack).

A.2 Example

The Bitcoin payment system uses the common SHA-256 (Secure Hash Algorithm). The following hash function example using SHA-256 visualises property 3. and 5. of the previous definition. Clearly no relation between both

hash values can be seen.

SHA-256(“Hello World”)
=d87774ec4a1052afb269355d6151cbd39946d3fe16716ff5bec4a7a631c6a7a8,
SHA-256(“Hello World.”)
=1e67e76d642e07e24a34ec0069212ef437e0b6e11101ed2f64434b5fea35e3fe.

B List of Feature Combinations used in Grid-Search

For a comprehensive overview of all 32 feature combinations we assign each individual predictor a number from 1 to 25.

btc close price = 1	xrp close price = 8	btc miner revenue = 15	msci price = 22
btc open price = 2	ltc close price = 9	btc cost per tx = 16	usd/eur rate = 23
btc high price = 3	btc addresses = 10	btc block size = 17	vix = 24
btc low price = 4	btc tx volume = 11	google trends = 18	google news = 25.
btc volume = 5	btc number tx = 12	wiki views = 19	
momentum = 6	btc tx conf time = 13	fed funds rate = 20	
eth close price = 7	btc hash rate = 14	gold price = 21	

Subsequently, Table B.1 shows a grid, where the rows represent all unique feature combinations and included features are marked as “x”.

Combination	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	x																								
2	x	x	x	x	x																				
3	x	x	x	x	x		x	x	x																
4	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x								
5	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x							
6	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x						
7	x	x	x	x	x					x	x	x	x	x	x	x	x								
8	x	x	x	x	x					x	x	x	x	x	x	x	x	x							
9	x	x	x	x	x					x	x	x	x	x	x	x	x	x	x						
10	x	x	x	x	x													x							
11	x	x	x	x	x													x	x						
12	x	x	x	x	x		x	x	x									x							
13	x	x	x	x	x		x	x	x									x	x						
14	x	x	x	x	x		x	x	x	x	x	x	x												
15	x	x	x	x	x		x	x	x	x	x	x	x					x							
16	x	x	x	x	x		x	x	x	x	x	x	x					x	x						
17	x	x	x	x	x					x	x	x	x												
18	x	x	x	x	x					x	x	x	x					x							
19	x	x	x	x	x					x	x	x	x					x	x						
20	x	x	x	x	x		x	x	x					x	x	x	x								
21	x	x	x	x	x		x	x	x					x	x	x	x	x							
22	x	x	x	x	x		x	x	x					x	x	x	x	x	x						
23	x	x	x	x	x									x	x	x	x								
24	x	x	x	x	x									x	x	x	x	x							
25	x	x	x	x	x									x	x	x	x	x	x						
26	x	x	x	x	x	x																			
27	x	x	x	x	x	x	x	x	x																
28	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x								
29	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x							
30	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						
31	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
32	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table B.1: Systematic overview about the setup of all 32 feature combinations analysed in the wrapper approach.

C Additional Graphics

To show that not using volatility adjusted returns distorts the difference between the models when plotting the returns of trading strategies, Figure C.1 illustrates the same strategies with non adjusted returns as shown in Figure 4.3.

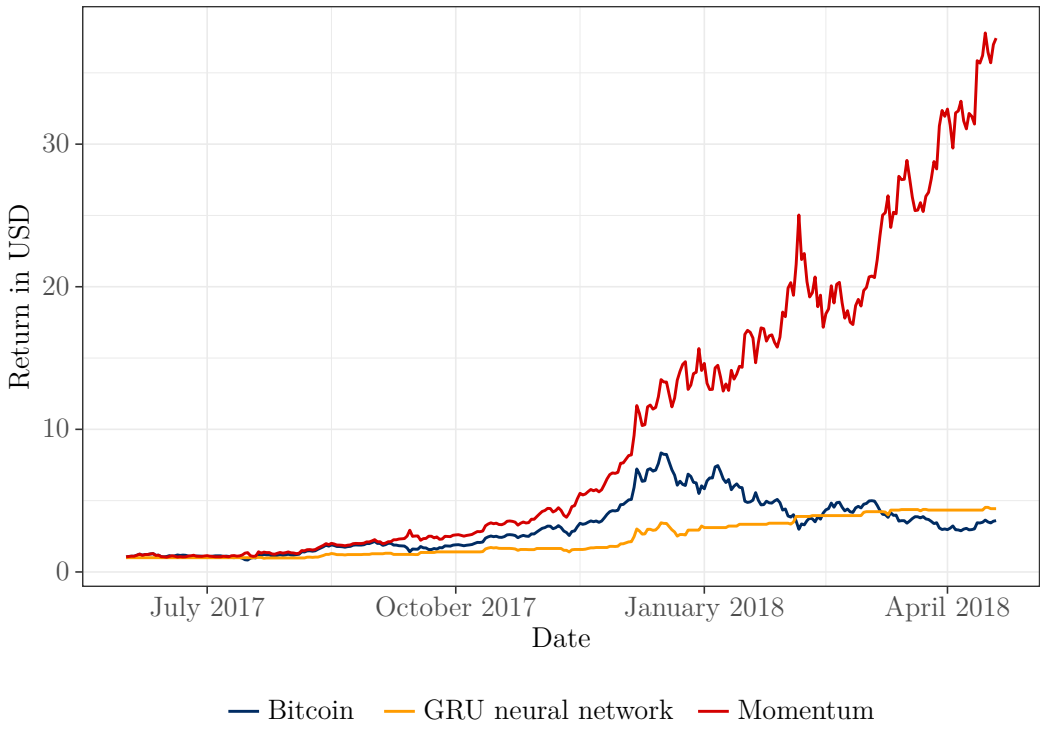

Figure C.1: Development of the investment of 1 USD into the two best performing methods, GRU neural network and momentum. Bitcoin price development as a benchmark. Not volatility adjusted.