UiA University of Agder

Master

Predicting bitcoin returns using artificial neural networks

An application of large datasets to Convolutional Neural Networks and Long Short-Term Memory based Artificial Neural Networks in finance.

DANIEL LINDESTAD

SUPERVISOR

Jochen Jungeilges

University of Agder, 2022

School of Business and Law

Department of Economics and Finance

Acknowledgements

This thesis is written as a part of the Master‘s degree in Business Administration with a specialization in Analytical Finance at the University of Agder. I would especially like to thank my supervisor, professor Jochen Jungeilges, who has been offering superbly valuable advice, feedback, and guidance throughout the process of writing this thesis, in addition to showing great patience when the work was at its most challenging, this thesis would not be possible without his guidance. Lastly, I would like to thank my fellow students, friends, and family for their support and helpful comments throughout the progression of this thesis, and the years before.

Abstract

Time series forecasting is one of the foremost challenges studied in finance. In this thesis various Convolutional Neural Network and Long Short Term Memory Artificial Neural Network models are used to predict Bitcoin returns. Previous literature has explored using data from Sentiment analysis of Social Media, and Blockchain information in isolation. This thesis seeks to combine the predictive power of earlier smaller models into a larger model that better utilizes a broader category of features in time series prediction. The resulting models are able to predict Bitcoin returns well, beating out simpler methods that do not utilize Artificial Neural Networks.

Keywords: Time series forecasting, Artificial neural networks, Forecasting, Bitcoin, Cryptocurrencies, Sentiment analysis.

JEL Classification: C45, C53.

Contents

List of Figures

List of Tables

1 Introduction

This master thesis explores machine learning predictions on large datasets. Machine learning has been able to develop comprehensive models that can make astonishing predictions in fields like text and speech recognition, and image classification. These models rely on large datasets to be able to create models with such great applications. When machine learning has been applied to finance, there has been a trend of not being able to fully utilize the depth and complexity of Neural Networks. This is because financial data is often very limited in frequency and span (Israel et al., 2020). When looking at returns one might only have one data point per day (or per month), if only a few features are available then little information can be extracted from this. This thesis attempts to remedy this limitation by broadening the feature selection available, by using a prediction target where data on fundamental factors are more easily available, including market perception, through sentiment analysis.

The use of Artificial Neural Networks (ANNs) in finance has been studied since the 1980s. Researchers have compared the performance of ANNs to conventional financial forecasting techniques and found that ANNs often outperform these methods. Some studies have found that neural network models can be more effective than classic statistical methods like regression and ARIMA at predicting price changes. The use of ANNs to predict prices and returns of Bitcoin and other cryptocurrencies has also been explored. Researchers have found that deep learning-based algorithms like LSTM can outperform traditional methods.

In the thesis, a rich dataset is gathered on historical Bitcoin prices, market sentiments through social media sentiment analysis, and various fundamental features of the underlying Bitcoin infrastructure, in the form of blockchain information. Several types of Artificial Neural Network models are trained, including Convolutional Neural networks, Long-Short Term

Memory Artificial Neural Networks, and combination models. Their performance metrics are characterized against the mean Bitcoin return. Several models outperform the simple mean return model, providing increased backing for the broad applications of Artificial Neural Networks for time-series prediction in finance.

The rest of the thesis is organized as follows. Chapter 2 present a review of the existing liteature relating to Artificial Neural Networks and their applications in finance. Chapter 3 covers prerequisite theory, including the basics of Artifcial Neural Networks, and the Efficient Market Hypothesis. Chapter 4 presents the data used in this thesis, the methods of collection, and the properties of the dataset. Chapter 5 details the methodology used when building the Artificial Neural Network models used in this thesis. Chapter 6 details the empirical results, and discusses them in light of previous literature and theory. Finally chapter 7 serves as the conclusion of the thesis.

2 Literature Review

The applications of Artificial Neural Networks to finance has been a topic of study since the end of the 1980s (White, 1988), (Jr. & Yoon, 1992). A number of authors have contrasted the outcomes of neural networks with those of conventional financial forecasting techniques. Canadian stock market returns were used to compare the average directional accuracy of ANNs with conventional least square regression and logistic regression by Olson and Mossman, 2003. The outcomes revealed that the ANN performed better than the best regression alternatives. Similar to this, Mostafa, 2010 carried out an empirical study to project closing price changes on the Kuwait Stock Exchange. To forecast the closing movements, they employed extended regression neural networks and multi-layer perceptron neural networks. The projections were also assessed using root mean square error (RMSE) and mean absolute error (MAE). The findings demonstrated that classic statistical methods like regression and ARIMA are not likely to perform as well as neural network models in predicting price changes.

Sitte and Sitte, 2000 studied how effective time delay neural networks (TDNN) are at predicting the S&P500, to measure the performance of their network they used root mean square error (RMSE). They found that the S&P500 TDNN predictions were same as a random walk prediction. They believed this to be a characteristic of the data, not the neural network. Rundo et al., 2019 did a large review of the most significant works in the field of machine learning applied to finance. They found the field was still evolving rapidly with new techniques continuing to be published, such as bat-neural network multi-agent system (BNNMAS), genetic algorithm-neural network (GANN), long short-term memory (LSTM) neural networks, among others. The surveyed studies conducted and reported in the article showed that deep learning-based algorithms such as LSTM outperformed traditional-based algorithms such as the ARIMA model. They also noted the significant success of sentiment analysis of both financial news and twitter data for building models that predict stock prices.

With the rise of Bitcoin and cryptocurrencies, an interest in using Artificial Neural Networks to predict both prices and returns has emerged (Spilak, 2018). Both Serafini et al., 2020 and Pano and Kashef, 2020 used VADER sentiment analysis for bitcoin pricing prediction. Serafini et al., 2020 used both a Recurrent Neural Network, and an Auto-Regressive Integrated Moving Average with eXogenous Input (ARIMAX) model to model how market behavior and sentiment impacted Bitcoin pricing. Both models were evaluated using Mean Square Error (MSE). Pano and Kashef, 2020 focused on finding what the optimal preprocessing strategy for BTC tweets to develop an accurate machine learning model that can predict bitcoin prices was. Pant et al., 2018 manually analyzed sentiments of tweets to train a Recurrent Neural Network (RNN) to predict Bitcoin prices.

Tripathi and Sharma, 2022 compared various types of LSTM networks (LSTM, Bi-directional LSTM, Convolutional Neural Network LSTM) on bitcoin price prediction, using Bayesian Optimization for hyperparameter tuning. They found that the Deep Artificial Neural Network model created using technical indicators as input data outperformed the other models, with record accuracy.

Jang and Lee, 2018 used a Bayesian Neural Network to predict bitcoin prices using fundamental blockchain data, including average block size, transactions per block, median confirmation time, hash rate, mining difficulty, number of confirmed transactions, and total amount of Bitcoin mined, finding this to improve the predictions of their model.

The paper Israel et al., 2020 discusses the potential uses of machine learning in finance, particularly in the field of return prediction. The authors provide an overview of machine learning and how it differs from traditional statistical methods. They also discuss the challenges of applying machine learning to finance and provide examples of successful uses of machine learning in finance. The authors conclude that machine learning is a natural evolution of quantitative tools in asset management, and not a revolutionary shift in the business model. Israel et al., 2020 argued that return prediction is a "small data" problem, because the amount of predictor variables does not change the class of problem to a “big data” problem, to do that one would need a large set of variables to try to estimate.

To summarize, various types of Artificial Neural Networks have been successfully been employed in previous research as a tool for forecasting financial time series. Previous literature on Bitcoin return prediction tend to only use historical prices and sentiments, or historical prices and blockchain data. This motivates the exploration of combining both of these data source into a more powerful model.

3 Theory

3.1 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models that are inspired by the structure and function of the human brain. These networks are composed of many interconnected processing nodes, which are called neurons. Each neuron receives input from other neurons, processes this input using a non-linear activation function, and then produces an output that is passed on to other neurons in the network. The main advantage of ANNs is that they are capable of learning from data, which allows them to make predictions or decisions based on previously unseen inputs. This ability to learn from data is what makes ANNs a powerful tool for many applications, such as image and speech recognition, natural language processing, and predictive modeling.

In an ANN, the connections between neurons are represented by weights, which are numerical values that determine the strength of the connection. During the learning process, the weights of the connections are adjusted based on the input data and the desired output, in order to improve the performance of the network. This process of adjusting the weights is known as training the network.

There are many different types of ANNs, which can vary in terms of their architecture and the learning algorithms that they use. Some of the most common types of ANNs include feedforward networks, convolutional networks, recurrent networks, and deep learning networks. Each of these types of ANNs has its own strengths and weaknesses, and is suited to different types of tasks and data.

3.1.1 The Perceptron

The perceptron is a type of artificial neural network that was developed in the 1950s by Frank Rosenblatt, a psychologist and computer scientist at the Cornell Aeronautical Laboratory. The perceptron is a simple model of a neuron in the human brain, and it is composed of a single layer of processing units, or neurons, that are connected to each other by weights (Rosenblatt, 1960).

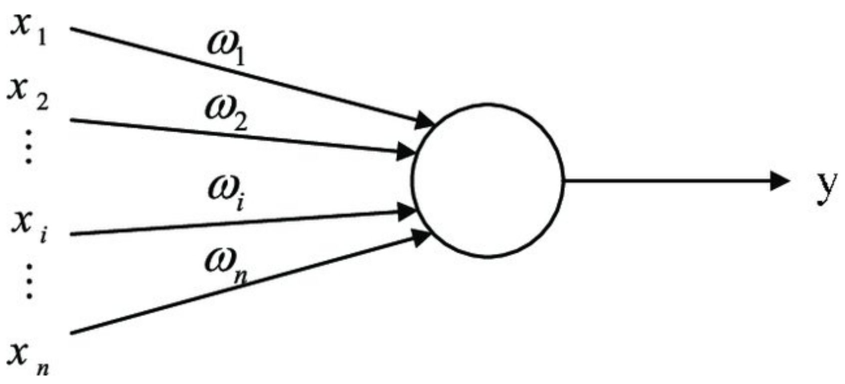


Figure 3.1: Structure of a single-layer perceptron (Shi, 2019).

Rosenblatt‘s original perceptron was designed to simulate the behavior of the visual system in the brain, and it was intended to be used for pattern recognition tasks. The perceptron was trained using a learning algorithm called the perceptron convergence procedure, which adjusted the weights of the network based on the input data and the desired output.

The perceptron was one of the first examples of a supervised learning algorithm, and it was seen as a promising step towards the development of intelligent machines that could learn from data. However, the perceptron had several limitations, and it was only able to solve linearly separable problems, which are problems in which the data can be divided into two classes by a single straight line (Minsky & Papert, 1969).

Despite its limitations, the perceptron sparked a great deal of interest and research in the field of artificial neural networks. In the 1960s, a number of researchers extended the perceptron model by adding additional layers of neurons, which allowed the network to solve more complex problems. These multi-layer perceptrons, as they were called, were the precursors to modern deep learning networks, which are now used for a wide range of applications.

Today, the perceptron is still used as a building block for many artificial neural networks, and it remains an important concept in the field of machine learning. Although it has been surpassed by more advanced algorithms, the perceptron continues to be a fundamental tool for understanding and working with neural networks.

3.1.2 Recurrent Neural Networks

Jordan, 1986 presented a first architecture as a superset of feedforward artificial neural networks that has one or more cycles. Each cycle makes it possible for a neuron to follow a path back to itself, allowing feedback of information. These cycles, or recurrent edges, allow the network‘s hidden units to see its own previous output so they give the network memory (Elman, 1990).

Recurrent neural networks (RNNs) are a type of neural network that can process sequential data, such as time series or natural language. Unlike traditional feedforward neural networks, which take a fixed-size input and produce a fixed-size output, RNNs can process a variable-length input and produce a variable-length output. This makes them well-suited for tasks such as language translation and speech recognition, where the input and output sequences can have different lengths.

RNNs were first proposed in the 1980s (Jordan, 1986), but it wasn't until the advent of deep learning and powerful computational resources in the 2010s that they became widely used. Early RNNs were relatively shallow, with only one or two hidden layers, and were not able to model long-range dependencies in the data. This led to the development of deeper RNNs, such as long short-term memory (LSTM) networks and gated recurrent units (GRUs), which are able to capture long-range dependencies more effectively.

RNNs are typically trained using a variant of backpropagation called backpropagation through time (BPTT). This involves unrolling the RNN in time and applying the chain rule to calculate the gradient of the loss function with respect to the network's weights. BPTT can be computationally expensive, especially for long sequences, and several techniques have been developed to improve its efficiency, such as truncated BPTT and reverse-mode differentiation (Werbos, 1990), (Frostig et al., 2021).

Despite their effectiveness, RNNs have several limitations. One of the main challenges is the vanishing gradient problem, where the gradient of the loss function with respect to the network's weights becomes very small during training, making it difficult for the network to learn. This can be mitigated using techniques such as gradient clipping and initialization of the network's weights. Another challenge is the inability of RNNs to handle long-range dependencies in the data. While deeper RNNs can alleviate this problem to some extent, they can still struggle to model very long sequences.

Overall, RNNs have proven to be a powerful tool for modeling sequential data and have been applied to a wide range of tasks, including language translation, speech recognition, and time series forecasting. While they have their limitations, research continues to improve their performance and address their challenges.

3.2 Efficient Market Hypothesis

The efficient market hypothesis (EMH) is a theory in finance that states that financial markets are efficient and that the prices of financial assets, such as stocks and bonds, reflect all available information. The EMH is based on the assumption that market participants are rational and have access to the same information (De Bondt & Thaler, 1985). The concept of market efficiency has a long history, dating back to the work of French mathematician Louis Bachelier in 1900 (Sewell, 2011). Bachelier's work, which focused on the behavior of stock prices, laid the foundation for the development of the EMH in the 1960s.

The EMH was formally introduced by Eugene Fama in his 1965 paper "The Behavior of Stock Market Prices" (Fama, 1965). In this paper, Fama proposed three different forms of market efficiency, which are now known as the weak form, semi-strong form, and strong form of the EMH.

The weak form of the EMH states that past prices and returns of a financial asset do not provide any information that can be used to predict its future prices and returns. In other words, technical analysis, which uses past price and volume data to make investment decisions, is not effective in outperforming the market. The semi-strong form of the EMH states that prices of financial assets reflect all publicly available information, such as financial statements and news releases. In this form, fundamental analysis, which uses information about a company's financial health and prospects to make investment decisions, is not effective in outperforming the market. The strong form of the EMH states that prices of financial assets reflect all information, including non-public information. In this form, insider trading, which involves the use of non-public information to make investment decisions, is not effective in outperforming the market.

The EMH has implications for investors, as it suggests that it is difficult to consistently outperform the market through individual investment decisions. Instead, investors are advised to diversify their portfolios and invest in a broad range of assets to reduce risk and maximize returns. Overall, the efficient market hypothesis is a widely-accepted theory in finance that has influenced the way that investors and financial analysts think about markets and investment decisions. Despite its popularity, the EMH has also been criticized for its assumptions and its ability to explain real-world market behavior.

One of the main criticisms of the EMH is that it assumes that market participants are rational and have access to the same information. However, in reality, investors may not always be rational and may not have equal access to information. This can lead to mispricings in the market and opportunities for investors to outperform the market. Another criticism of the EMH is that it does not fully explain the behavior of financial markets. For example, market bubbles, where prices of assets rise to unsustainable levels, and crashes, where prices of assets fall sharply, are not consistent with the predictions of the EMH. Despite these criticisms, the EMH continues to be an influential theory in finance, and its concepts are widely used by investors and financial analysts.

4 Data

This section details the collection process, of obtaining data used, and further processing that has been performed on the data. The data included has been chosen on the basis of being freely available, and being fundamentally relevant to Bitcoin pricing. All data is from the period January 2012 to November 2022. The data must be split into a training and testing period so that it can be used to train an Artificial Neural Network. A common split used (Geron, 2019) is the 80/20 split, which is also used in this thesis. The first 80% of the data is allocated to the training period, and the last 20% is allocated to the testing period. Each datapoint consists of the last 50 days worth of data, and the prediction target (label) is the next days log-return.

4.1 Historical Bitcoin Returns, Volume

The prediction target of the model is future Bitcoin log-returns. Historical daily Bitcoin prices and volume are from Bitcoinity, using prices from four exchanges: bitstamp, coinbase, gemini, and kraken. The initial data includes some outliers, in particular during exchange outages. The daily mean price is computed, and values that lie more than 2% outside the mean price are removed from the dataset (until at least two values exist at the time-step). A new daily mean price is calculated, which is used.

Subsequently, daily log-returns are calculated:

$$R_t = ln\left(\frac{P_t}{P_{t-1}}\right) \tag{4.1}$$

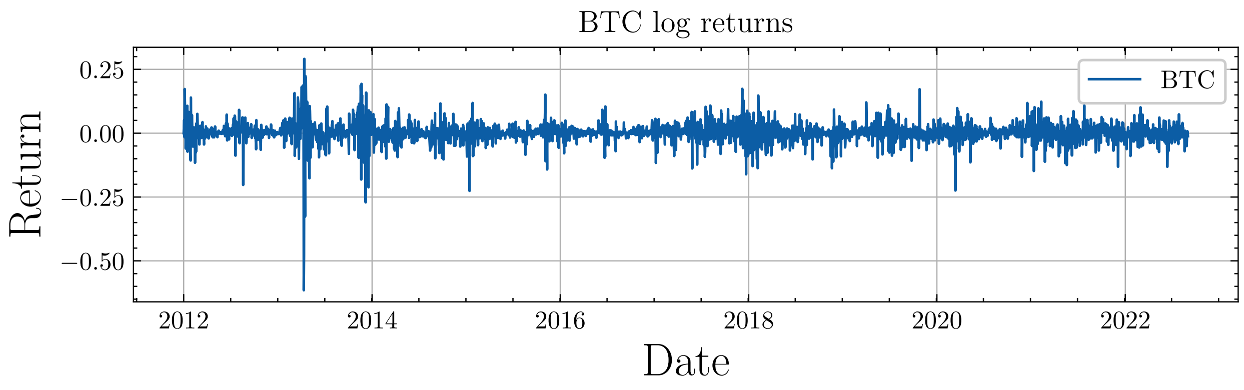


Figure 4.1: Daily log-returns for Bitcoin.

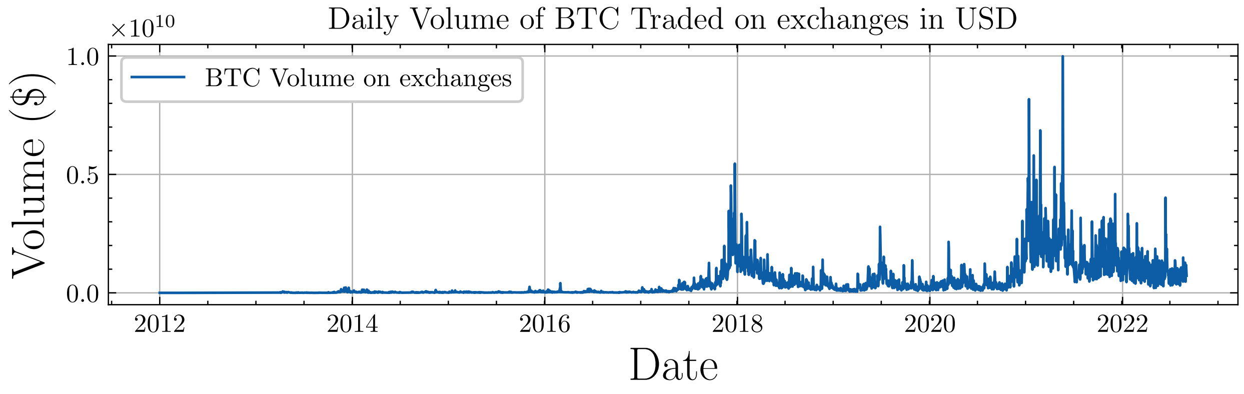


Figure 4.2: Daily USD volume of Bitcoin traded on exchanges.

4.2 Twitter sentiment data

Opinions expressed in social media can provide valuable insight into the market outlook on an asset. A number of papers have used sentiment analysis in their models with great success (Pano & Kashef, 2020), (Mohapatra et al., 2020), (Pant et al., 2018). A total of 237 million tweets from January 2012 to November 2022 were gathered on five keywords using the python library *snscrape*:

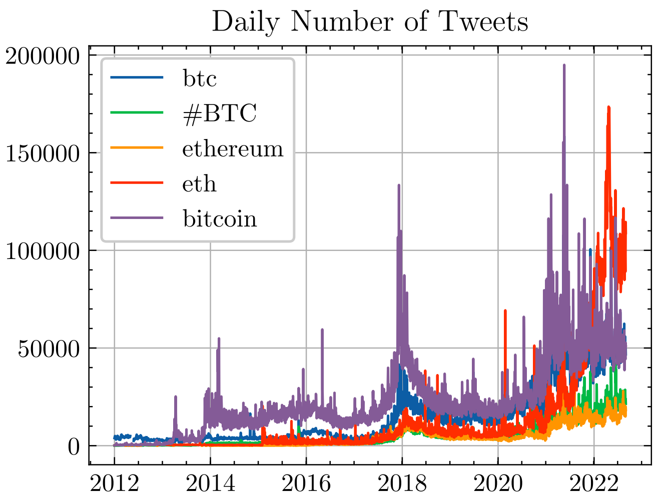


Figure 4.3: Daily number of tweets for each keyword.

Keyword	Number of tweets
btc	59,051,214
eth	50,038,897
#BTC	20,282,755
bitcoin	91,748,937
ethereum	16,233,656
Total	237,355,459

Table 4.1: Keywords and number of results retrieved.

VADER - Valence Aware Dictionary and Sentiment reasoner

The main challenge that comes with classifying social media sentiments are time and accuracy. Manual classification is extremely time-consuming. VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media (Hutto & Gilbert, 2015). It is fully open-sourced under the MIT License.

VADER uses a combination of natural language processing (NLP) and rule-based sentiment analysis to accurately identify sentiment in text. In other words, VADER uses a dictionary of sentiment-related words, along with a set of heuristics, to determine the overall sentiment of a piece of text. This approach allows VADER to accurately capture the nuances of sentiment that are often expressed in social media, including emoticons, emojis, and slang (Hutto

& Gilbert, 2015).

One of the key strengths of VADER is its ability to handle the ambiguity and subjectivity of sentiment. For example, consider the phrase “I am not unhappy.” This phrase could be interpreted in a variety of ways, depending on the context. VADER is able to handle this kind of ambiguity by using a combination of NLP and rule-based techniques to accurately identify the sentiment expressed in the phrase.

In terms of implementation, VADER uses a dictionary of sentiment-related words, along with a set of rules, to determine the overall sentiment of a piece of text. The dictionary includes words that are associated with positive or negative sentiment, as well as words that are associated with neutral sentiment. VADER also includes a set of rules that allow it to identify sentiment-related punctuation, such as exclamation points, and to adjust the sentiment score accordingly. VADER assigns a numeric value (polarity score) to the sentiment expressed in a string of text, such as a tweet, in the range of -1 to 1. A polarity score of -1 represents a maximally negative sentiment, a polarity score of 0 represents a neutral sentiment, and a polarity score of 1 represents a maximally positive sentiment.

Overall, VADER is a valuable tool for analyzing the sentiment of text, particularly in the context of social media. Its combination of NLP and rule-based techniques allows it to accurately identify sentiment, even in the face of ambiguity and subjectivity. As a result, VADER is a useful tool for researchers and practitioners who are interested in understanding the sentiment expressed in social media data.

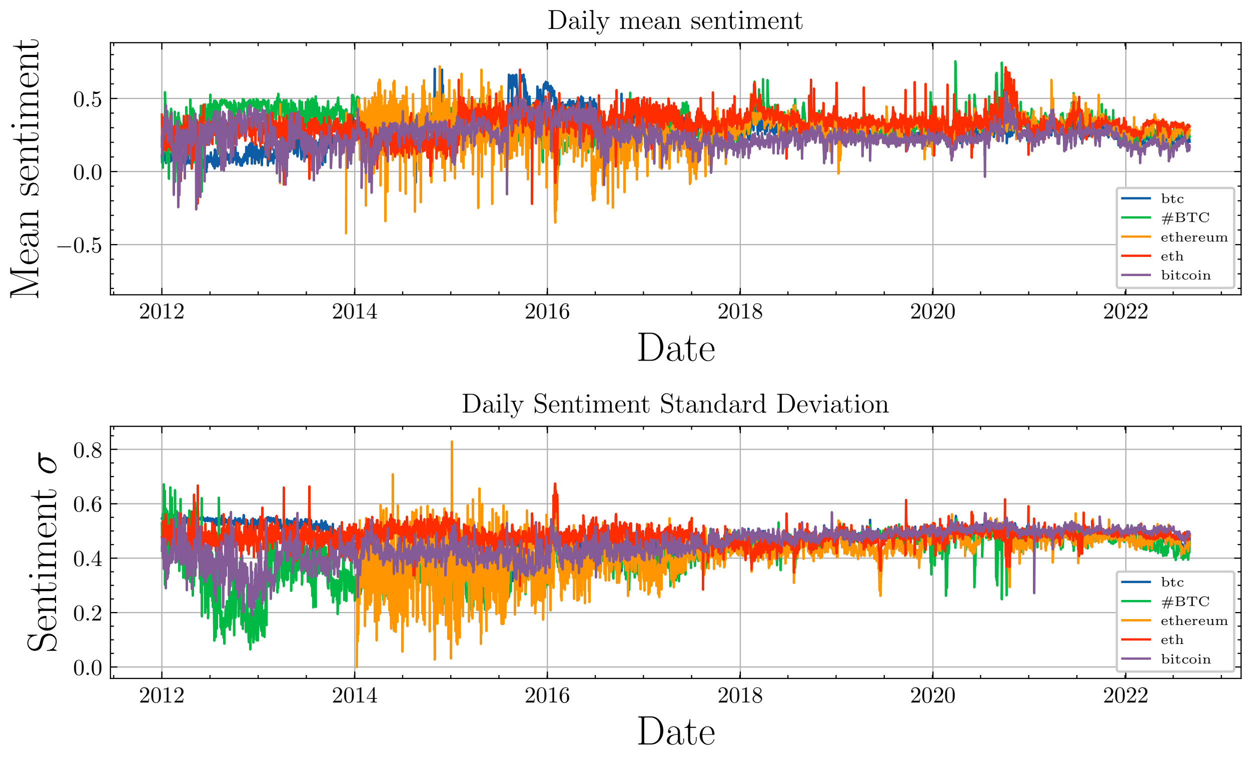


Figure 4.4: Daily sentiment μ and σ

For each day, the mean sentiment μ, and the standard deviation σ of sentiment, is computed for each keyword. When VADER is not able to classify a sentiment properly, it assigns a polarity score of 0. Therefore, to increase the signal to noise ratio, tweets that are given a polarity score of exactly 0 are excluded. A comparison of the sentiment data with- and without polarity scores of 0 is made available in the appendix.

4.3 Blockchain data

A blockchain is a distributed, decentralized, public ledger that records transactions on multiple computers. It allows for the secure and transparent storage of transaction data without the need for a central authority or intermediary. At its core, a blockchain is a chain of blocks that contain transaction data. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. This structure allows for the creation of a tamper-evident, append-only record of transactions (Narayanan et al., 2016).

The decentralized nature of a blockchain means that it is not controlled by any single entity and is instead maintained by a network of participating nodes. This distributed network ensures that the transaction data on the blockchain is not susceptible to tampering or revision. The use of cryptographic hashes and distributed network architecture allows for the implementation of consensus mechanisms that enable the network to agree on the current state of the blockchain. This ensures the integrity of the transaction data and allows for the creation of trust among parties involved in the transaction.

The information stored on the blockchain is useful when pricing the asset. The blockchain contains a complete and transparent record of all transactions on the bitcoin network. This allows for the analysis of transaction data, including the amount of bitcoin being transferred, and the addresses involved in the transaction. This information can be used to identify trends and patterns in the movement of bitcoin, which can be useful in forecasting its future value. The use of cryptographic hashes and distributed network architecture in the blockchain ensures the integrity and immutability of transaction data. This means that the data on the blockchain cannot be tampered with or altered, providing a reliable source of information for forecasting purposes. Furthermore, the adoption of bitcoin as a store of value and means of exchange can also impact its future returns. The increasing acceptance of bitcoin by businesses, institutions, and individuals as a legitimate asset class can drive its demand and value, leading to potential future returns for investors. Different metrics of the blockchain and Bitcoin network are included to capture this information: The number of transactions, blocksize, mining difficulty, and the network hashrate.

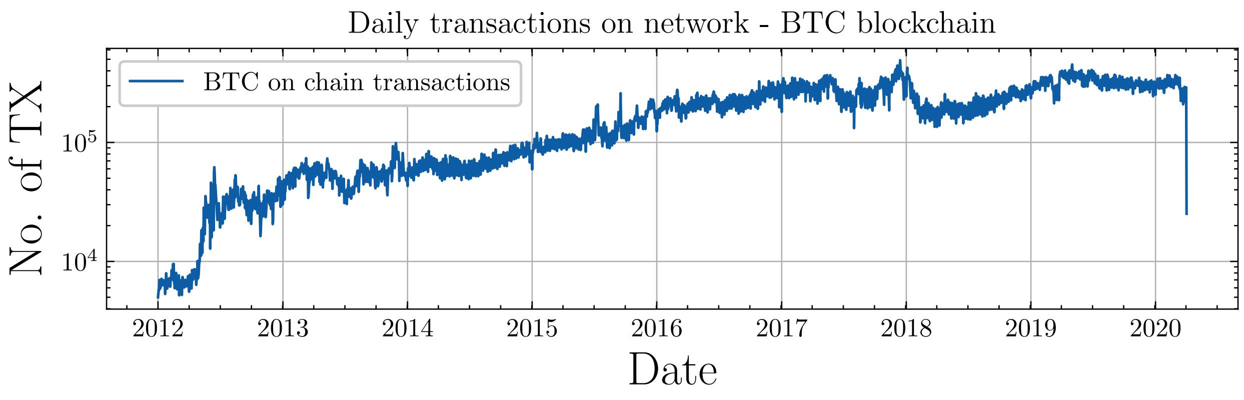


Figure 4.5: Number of daily transactions on-chain, log-scale.

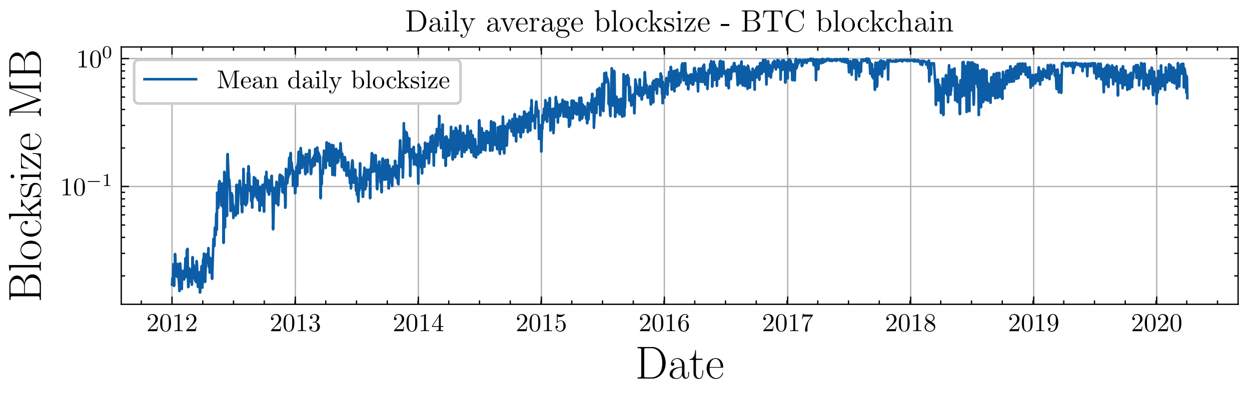


Figure 4.6: Average Blocksize in MB, log-scale.

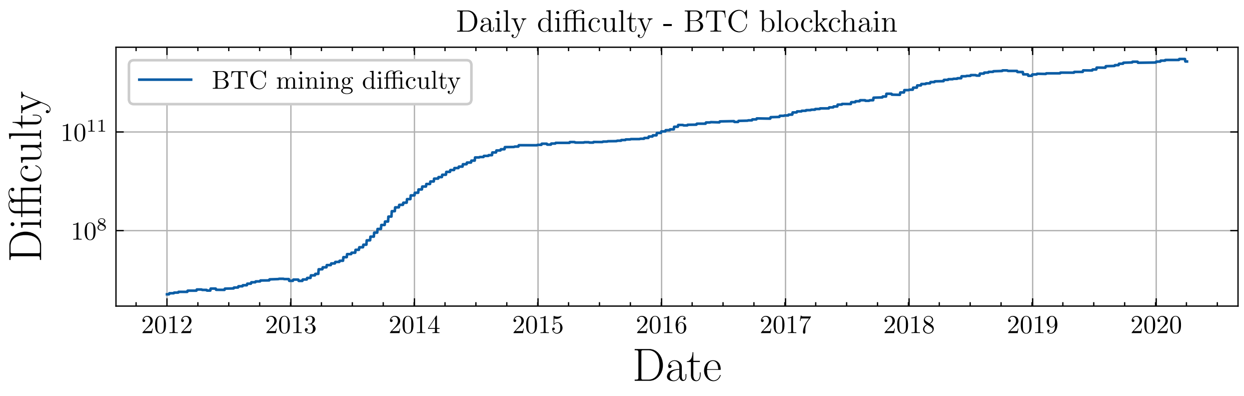


Figure 4.7: Daily Difficulty, log-scale.

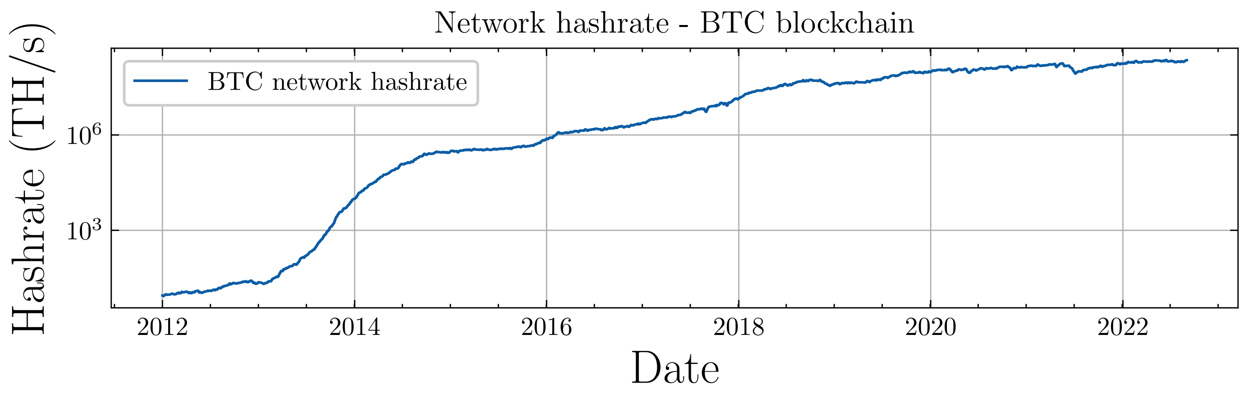


Figure 4.8: Daily mean network hash-rate, log-scale.

The number of transactions, blocksize, mining difficulty are sourced from Bitcoinity, while the hashrate is sourced from Blockchain.com.

4.4 Stationarity

In time series analysis, stationarity refers to the statistical properties of a time series, such as the mean and variance, that do not change over time. A time series is said to be stationary if its statistical properties are constant over time.

In other words, a stationary time series has a constant mean and variance, and its autocorrelation structure does not depend on the time at which the series is observed. This means that the time series can be modeled using a fixed set of parameters, which makes it easier to make predictions about future values in the series.

There are several methods for testing whether a time series is stationary, such as the Augmented Dickey-Fuller test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test (Kwiatkowski et al., 1992). These tests use statistical tests to determine whether the mean and variance of the time series are constant over time.

The Augmented Dickey-Fuller (ADF) test is a statistical test used to determine the presence of a unit root in a time series sample. A unit root is a feature of a time series in which the value of the series at a given point is the sum of its own previous value and a white noise error term. If a time series has a unit root, then it is said to be non-stationary, meaning that its statistical properties (such as its mean and variance) vary over time (Dickey & Fuller, 1979).

We start with the Dickey-Fuller test. Consider the following $AR(1)$ regression model:

$$y_t = \theta_{y_{t-1}} + \epsilon_t \tag{4.2}$$

The unit root null hypothesis against the stationary alternative corresponds to

$$H_0 : \theta = 1 \qquad against \qquad H_A : \theta < 1 \tag{4.3}$$

The model can also be stated as

$$\Delta y_t = (\theta - 1)\, y_{t-1} + \epsilon_t = \pi y_{t-1} + \epsilon_t \tag{4.4}$$

Here $\pi = \theta - 0$. The unit root hypothesis can be restated as

$$H_0 : \pi = 0 \qquad against \qquad H_A : \pi < 1 \tag{4.5}$$

The Dickey-Fuller test is the t-test for H_0:

$$\hat{r} = \frac{\hat{\theta} - 1}{SE(\hat{\theta})} = \frac{\hat{\pi}}{SE(\hat{\pi})} \tag{4.6}$$

The ADF test is an extension of the Dickey-Fuller test, which was developed in the 1970s as a way to test for the presence of a unit root in a time series. The ADF test is an improved version of the Dickey-Fuller test that uses additional information such as the lagged values of the time series in order to provide more accurate results.

The null hypothesis of the ADF test is that the time series has a unit root, while the alternative hypothesis is that the time series is stationary. To perform the ADF test, we first need to construct an augmented regression model of the time series, which includes the time series and its lagged values as independent variables, as well as an error term. This augmented regression model is then used to estimate the coefficients of the time series and its lagged values, which are used to calculate the ADF statistic.

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \pi_1 \Delta y_{t-1} + ... + \pi_{p-1} \Delta y_{t-p+1} + \epsilon_t \tag{4.7}$$

where the constant term α and the coefficient on a time trend β are included in a model of an autoregressive process with lag order p. When the values of α and β are both set to zero, the model represents a random walk, whereas setting only β to zero results in a random walk with a drift.

The ADF test statistic is given by

$$DF_\tau = \frac{\hat{\gamma}}{\mathrm{SE}(\hat{\gamma})} \tag{4.8}$$

The ADF statistic is a t-statistic that measures the significance of the estimated coefficients in the augmented regression model. If the ADF statistic is greater than the critical value for the chosen significance level, then we can reject the null hypothesis and conclude that the time series is stationary. The following test statistics and p-values where computed:

Time series	Test statistic	p-value
btc_logreturn	-10.2504	**0.0000**
btc_volume	-2.7094	0.0724
blockchain_blocksize	-1.9500	0.3089
blockchain_transactions	-1.7804	0.3902
blockchain_difficulty	0.2266	0.9737
blockchain_hashrate	1.5435	0.9977
sentiments_btc	-3.2501	**0.0173**
sentiments_sd_btc	-5.5455	**0.0000**
sentiments_#BTC	-4.4787	**0.0002**
sentiments_sd_#BTC	-3.0432	**0.0310**
sentiments_ethereum	-3.2736	**0.0161**
sentiments_sd_ethereum	-2.2192	0.1994
sentiments_eth	-4.5441	**0.0002**
sentiments_sd_eth	-6.0128	**0.0000**
sentiments_bitcoin	-5.2523	**0.0000**
sentiments_sd_bitcoin	-3.2426	**0.0176**

Table 4.2: Augmented Dickey-Fuller test results. P-values significant at the 5%-level are highlighted.

We fail to reject H_0 on six variables, showing that a significant portion of the data is not stationary. Notably, the variable we are trying to predict (BTC log-returns) are shown to be stationary (we can reject H_0, and affirm the alternative hypothesis H_a, showing stationarity for the log-returns).

Recurrent neural networks (RNNs) are a type of artificial neural network that are designed to process sequential data, such as time series data. Unlike traditional feedforward neural networks, which have a fixed input and output size, RNNs have a "memory" that allows them to store information from previous time steps and use it to inform their predictions or decisions at the current time step.

One of the key advantages of RNNs is that they do not require the input time series to be stationary (Malhotra et al., 2015). Stationarity is a desirable property for time series data, because it simplifies the analysis and modeling of the data. However, many real-world time series are non-stationary, meaning that they have varying means and variances, and exhibit trends or seasonality. Non-stationary time series can be difficult to model using traditional methods, because the statistical properties of the data are constantly changing.

RNNs, on the other hand, are able to model non-stationary time series effectively, because they are able to capture the dynamics of the data and learn how the statistical properties of the series change over time (Malhotra et al., 2015). This ability to model non-stationary time series is particularly useful in applications such as financial forecasting, where the data may be subject to changes in market conditions or other factors (Gers et al., 2001). One popular type of RNN is the long short-term memory (LSTM) network, which is a variant of the basic RNN architecture that uses special units called LSTM cells to store and manipulate information from previous time steps. RNNs and LSTM networks are well-suited for modeling non-stationary time series data, because they are able to capture the dynamics of the data and learn how the statistical properties of the series change over time. This ability to model non-stationary data makes RNNs and LSTMs a powerful tool for time-series forecasting.

4.5 Descriptive Statistics

Descriptive statistics provides a set of tools and techniques for describing and summarizing data. These tools include measures of central tendency, such as the mean, median, and mode, which describe the center of the data distribution. Other measures, such as the standard deviation, variance, and interquartile range, describe the spread and dispersion of the data.

Data	Mean	Std	Iqr	Min	Max	Skewness	Kurtosis
btc_logreturn	0.002	0.038	0.028	-0.615	0.291	-1.784	30.794
btc_volume	4.22e+08	7.20e+08	5.14e+08	6.71e+04	9.99e+09	3.520	21.166
bc_blocksize	5.20e+05	2.83e+05	4.94e+05	1.48e+04	9.98e+05	-0.054	-1.079
bc_transactions	1.38e+05	1.17e+05	2.13e+05	5.00e+03	4.91e+05	0.568	-1.123
bc_difficulty	4.93e+12	6.01e+12	1.38e+13	1.16e+06	1.66e+13	0.676	-1.333
bc_hashrate	4.78e+07	6.54e+07	9.51e+07	8.58e+00	2.28e+08	1.163	0.023
sen_btc	0.266	0.109	0.102	-0.076	0.704	0.566	1.560
sen_sd_btc	0.466	0.064	0.062	0.000	0.559	-3.350	21.065
sen_#BTC	0.328	0.088	0.119	-0.139	0.755	-0.095	1.129
sen_sd_#BTC	0.408	0.087	0.119	0.000	0.672	-1.022	0.921
sen_ethereum	0.234	0.155	0.189	-0.765	0.803	-0.870	1.584
sen_sd_ethereum	0.342	0.180	0.170	0.000	0.842	-1.115	-0.279
sen_eth	0.322	0.083	0.088	-0.222	0.714	-0.089	2.779
sen_sd_eth	0.481	0.032	0.034	0.000	0.674	-0.692	17.268
sen_bitcoin	0.246	0.087	0.093	-0.260	0.546	-0.208	2.175
sen_sd_bitcoin	0.449	0.059	0.082	0.000	0.570	-1.525	5.882

Table 4.3: Descriptive statistics.

A normal distribution has a skewness equal to zero. Therefore, a positive skewness indicates a distribution that is skewed to the right, while a negative skewness indicates a distribution that is skewed to the left. A positive skewness may be an indication of more small losses and fewer large gains in the returns. In contrast, a negative skewness may indicate a distribution with longer or fatter tails. A normal distribution has a kurtosis of 3. We note that the Bitcoin log-returns are highly leptokurtic, with a kurtosis of 30.8. This means that the log-returns exhibit heavy tails.

4.6 Data scaling for use with Neural Networks

Data scaling is an important preprocessing step for training artificial neural networks. Neural networks are sensitive to the scale of the input data, and if the data is not properly scaled, the network may not be able to learn effectively (Geron, 2019).

One of the reasons why data scaling is important is that it helps to ensure that the network can learn from the data. If the data is not scaled, some input variables may have much larger values than others, which can cause the network to focus disproportionately on those variables. For example, if one input variable has values in the range 0-1 and another has values in the range 0-1000, the network may pay much more attention to the second variable, since it has much larger values. This can lead to suboptimal learning and may prevent the network from achieving good performance.

Data scaling also helps to improve the convergence of the network during training. The optimization algorithms used to train neural networks typically work by adjusting the network's weights in the direction of the gradient of the loss function. If the data is not scaled, the gradients of the loss function with respect to the weights may have different scales, which can cause the optimization algorithm to oscillate or move in suboptimal directions. By scaling the data, the gradients will have more consistent scales, which can help the optimization algorithm to converge more quickly and reliably (Djordjevic et al., 2022).

Another benefit of data scaling is that it can improve the generalization of the network. When a neural network is trained on a particular dataset, it tries to learn a set of weights that can be used to make accurate predictions on that dataset. However, the network's performance on unseen data (i.e. data that was not used during training) is not always as good as its performance on the training data. This is because the network may have learned patterns that are specific to the training data, and may not generalize well to new data. By scaling the data, the network can learn more robust and generalizable patterns, which can improve its performance on unseen data.

To optimize the performance of the models, the data is scaled using the *sklearn* python library to have a mean of 0 and a variance of 1. Each feature is independently scaled, and after model prediction, the prediction data is un-scaled using the inverse scaling operation.

5 Methodology

This section covers the method used to generate the various Artificial Neural Network models used in this thesis.

5.1 Gradient Descent

Gradient descent is an optimization algorithm that is commonly used in machine learning and deep learning. It is an iterative algorithm that starts with an initial estimate of the solution to a problem, and then iteratively improves the solution by taking small steps in the direction of the negative gradient of the objective function (Geron, 2019).

The objective function is a mathematical function that expresses the problem to be solved. For example, in a machine learning problem, the objective function might be the loss function, which measures the difference between the predicted output of a model and the true output. The goal of gradient descent is to find the values of the model parameters that minimize the loss function, and therefore produce the best possible predictions.

To understand how gradient descent works, it is helpful to consider a simple example. Suppose we have a function $f(x)$ that we want to minimize. We can use gradient descent to find the minimum of this function by starting with an initial guess of the solution, x_t, and then iteratively updating the solution according to the following rule:

$$x_{t+1} = x_t - \eta \times \nabla f(x_t) \tag{5.1}$$

Here, η is the learning rate, which determines the size of the steps that we take in the direction of the negative gradient. The gradient, $\nabla f(x)$, is a vector that points in the direction of the greatest increase in the function. By moving in the opposite direction, we can move downhill towards the minimum of the function.

In practice, we can repeat this process many times, using the updated solution from one iteration as the initial guess for the next iteration. This allows us to improve the solution iteratively, until we reach a point where the gradient is very small, indicating that we have reached the minimum of the function.

One of the key advantages of gradient descent is that it is an efficient algorithm that can handle large-scale optimization problems. By using a learning rate that is carefully chosen, we can ensure that the algorithm converges to the minimum of the function in a reasonable amount of time. Furthermore, gradient descent can be easily parallelized, allowing us to take advantage of modern computing architectures to solve large-scale optimization problems more efficiently.

5.2 Activation Function (Loss Function)

Activation functions are a fundamental component of neural networks, and are used to compute the output of a neuron given its input. An activation function (also referred to as a loss function) takes in a real-valued input, performs a mathematical operation on it, and produces a real-valued output. The output of the activation function is then passed on to the next layer of the neural network, where it is used to compute the output of the next set of neurons (Geron, 2019).

There are many different activation functions that can be used in a neural network, and the choice of activation function can have a significant impact on the performance of the network. Some commonly used activation functions include the sigmoid function, the hyperbolic tangent function, and the Rectified Linear Unit (ReLU) function.

The sigmoid function is a smooth, S-shaped function that maps any real-valued input to a value between 0 and 1:

$$F(x) = \frac{1}{1+e^{-x}} \tag{5.2}$$

It is often used in the output layer of a binary classification neural network, where it can be interpreted as a probability. The sigmoid function has the useful property that its derivative is easy to compute, which makes it efficient to use in training a neural network.

The hyperbolic tangent function, also known as tanh, is similar to the sigmoid function, but maps inputs to values between -1 and 1.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.3}$$

Like the sigmoid function, the tanh function is smooth and differentiable, which makes it easy to use in training a neural network. However, the tanh function has a slightly steeper slope than the sigmoid function, which can make it more effective at capturing subtle patterns in the data (Lewis et al., 2020).

The ReLU function is a simple, non-linear function that maps any input that is less than or equal to 0 to 0, and any input that is greater than 0 to the input itself:

$$f(x) = max(0, x) \tag{5.4}$$

The ReLU function is widely used in neural networks because it is computationally efficient and has been shown to improve the performance of neural networks on a wide range of tasks (Sussillo & Abbott, 2014).

Overall, activation functions are an important component of neural networks, and the choice of activation function can have a significant impact on the performance of the network. Different activation functions have different properties, and the appropriate choice of activation function will depend on the specific task and the characteristics of the data. The ReLU activation function is used for all models trained in this thesis.

5.3 Long Short-Term Memory (LSTM)

Long time lag problems are difficult for traditional recurrent neural networks using hidden units to learn due to a tendency for the gradients to vanish. Long Short Term Memory units (LSTM units) were introduced as a solution to this by addressing the vanishing gradients issue (Hochreiter & Schmidhuber, 1997).

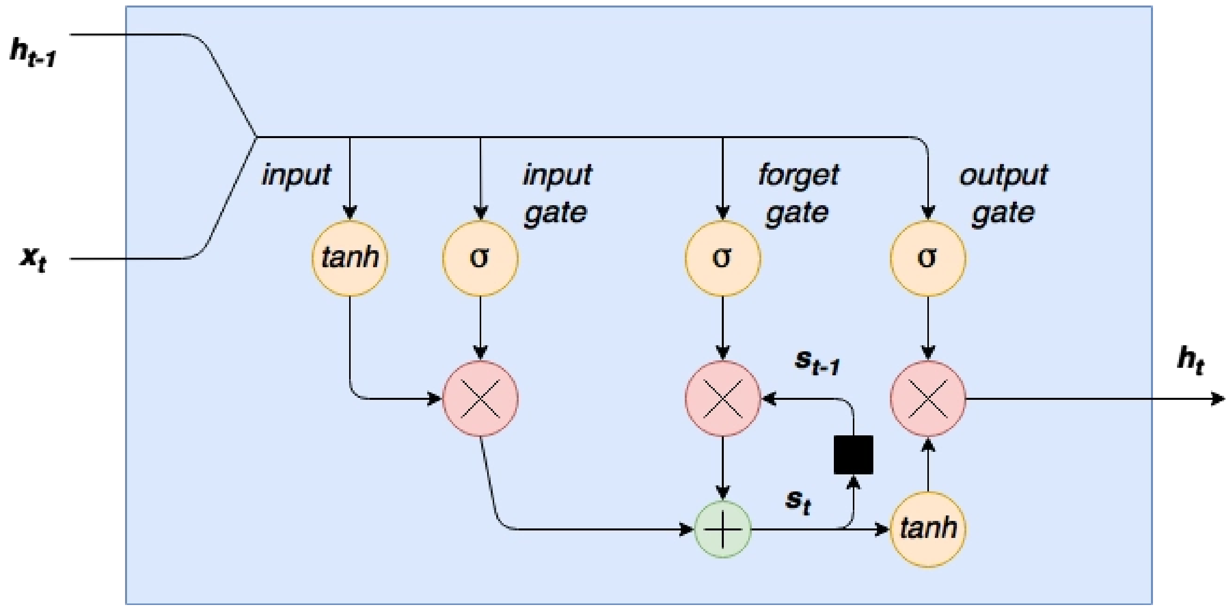


Figure 5.1: Structure of an LSTM unit (Kienzler, 2018).

LSTM units take an input x_t, are composed of an input gate i_t, a forget gate f_t, and an output gate o_t. The gates are used to add or remove information from the unit, they can be read from, written to, or erased at each time step, through explicit gating mechanisms. The unit also maintains a hidden state vector h_t and a memory cell s_t. The following equations express a forward pass of a single LSTM unit (Greff et al., 2017):

$$f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f \right) \tag{5.5}$$

$$i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i \right) \tag{5.6}$$

$$o_t = \sigma_g \left(W_o x_t + U_o h_{t-1} + b_o \right) \tag{5.7}$$

$$s_t = f_t \times s_{t-1} + i_t * \sigma_s \left(W_s x_t + U_s h_{t-1} + b_s \right) \tag{5.8}$$

$$h_t = o_t * \sigma_h \left(c_t \right) \tag{5.9}$$

The symbol $*$ denotes the element-wise product of the matrices.

5.4 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a type of neural network that is particularly well-suited to processing data that has a grid-like structure, such as an image. CNNs are composed of multiple layers of interconnected neurons, and are trained using a variant of the backpropagation algorithm (O'Shea & Nash, 2015).

One of the key features of CNNs is the use of convolutional layers, which are designed to automatically and adaptively learn spatial hierarchies of features. In a convolutional layer, the input is passed through a set of learnable filters, each of which is convolved with the input to produce a set of output maps. The filters are typically small spatially, but extend through the full depth of the input. The convolution operation is repeated across the entire input, with the filter moving in a sliding window fashion. This allows the convolutional layer to automatically learn spatial hierarchies of features, without the need for manual engineering (O'Shea & Nash, 2015).

Another key feature of CNNs is the use of pooling layers, which are designed to reduce the spatial size of the input, while retaining the most important information. This is typically achieved through the use of a pooling operation, such as max pooling, which selects the maximum value within a small window of the input. By reducing the spatial size of the input, pooling layers make the neural network more computationally efficient, while also helping to reduce overfitting.

Once the convolutional and pooling layers have extracted and condensed the relevant features from the input, the resulting feature maps are passed through a series of fully connected layers, which use the features to make predictions. The output of the fully connected layers is then passed through a loss function, which is used to compute the error between the predicted output and the true output. This error is then backpropagated through the network, and used to update the weights of the network in order to reduce the error (O'Shea & Nash, 2015).

Overall, CNNs are a powerful tool for processing data that has a grid-like structure, such as an image. The use of convolutional and pooling layers allows CNNs to automatically and adaptively learn spatial hierarchies of features, while the fully connected layers use these features to make predictions. This makes CNNs well-suited to a wide range of tasks, including image classification, object detection, and image segmentation.

CNNs are typically used to process data that has a grid-like structure, such as an image. However, CNNs can also be applied to time series data, which has a temporal structure rather than a spatial structure. One of the key reasons why CNNs can be applied to time series data is that the convolution operation, which is central to the functioning of CNNs, is a mathematical operation that can be applied to any type of data, regardless of its structure. In the case of time series data, the convolution operation is applied along the temporal axis, rather than the spatial axis. This allows CNNs to automatically and adaptively learn temporal hierarchies of features, just as they would learn spatial hierarchies of features in the case of image data (Velastegui et al., 2020).

Another reason why CNNs can be applied to time series data is that the convolutional and pooling layers in a CNN are shift-invariant, meaning that they produce the same output regardless of the position of the input within the temporal sequence. This property is particularly useful for time series data, where the order of the data points is important. By using shift-invariant layers, CNNs are able to capture the temporal dynamics of the data, without the need for manual engineering.

Furthermore, CNNs are able to handle variable-length time series data, which is common in many real-world applications. By using padding, CNNs can process time series of different lengths, and still produce a consistent output. This allows CNNs to be applied to a wide range of time series data, without the need for data preprocessing or manual feature engineering.

Overall, CNNs can be applied to time series data because the convolution operation is a general mathematical operation that can be applied to any type of data, regardless of its structure. The shift-invariance of the convolutional and pooling layers allows CNNs to capture the temporal dynamics of the data, and the ability to handle variable-length inputs makes them well-suited to a wide range of time series data.

5.5 Combination Networks

5.5.1 CNN-LSTM

Convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are both types of neural network that are widely used in machine learning and deep learning. CNNs are typically used to process data that has a grid-like structure, such as an image, while LSTM networks are used to process data that has a temporal structure, such as a time series.

Combining CNNs and LSTM networks can be particularly useful for tasks involving financial time series data, where the goal is to make predictions about the future behavior of financial markets. Financial time series data has both spatial and temporal structure, and can be effectively processed by combining CNNs and LSTM networks (Tripathi & Sharma, 2022).

For example, consider a stock prediction task, where the goal is to predict the future price of a stock based on historical data. A CNN could be used to process the historical data, and to extract spatial features such as patterns, trends, and anomalies. These spatial features could then be passed to an LSTM network, which would process the sequence of spatial features over time, and use them to make predictions about the future price of the stock.

Another way in which CNNs and LSTM networks can be combined for financial time series data is by using a CNN to extract spatial features from the input data, and then using these features as the input to an LSTM network. This approach allows the CNN to capture the spatial patterns in the data, and to produce a condensed representation of the input that is suitable for processing by an LSTM network. This can be particularly useful when dealing with large-scale datasets, where the use of a CNN can make the overall model more computationally efficient (Tripathi & Sharma, 2022).

Overall, combining CNNs and LSTM networks can be a powerful approach for tasks involving financial time series data. The ability of CNNs to extract spatial features from the data, and the ability of LSTM networks to process temporal sequences, make these two types of network well-suited to predicting the future behavior of financial markets. By combining CNNs and LSTM networks, we can build models that are able to capture both the spatial and temporal patterns in the data, and make more accurate predictions.

5.5.2 Multi-Head-CNN-LSTM (MH-CNN-LSTM)

Multi-head convolutional neural networks (CNNs) and long short-term memory (LSTM) networks are a type of neural network architecture that combines the strengths of both CNNs and LSTM networks. In a multi-head CNN-LSTM network, the input data is processed by multiple parallel CNNs, each of which extracts different spatial features from the data. The output of the CNNs is then passed to an LSTM network, which processes the temporal sequence of spatial features and uses them to make predictions (Mo et al., 2020).

This type of architecture can be particularly useful for tasks involving financial time series data, where the goal is to make predictions about the future behavior of financial markets. Financial time series data has both spatial and temporal structure, and can be effectively processed by combining multiple CNNs and an LSTM network.

For example, consider a stock prediction task, where the goal is to predict the future price of a stock based on historical data. A multi-head CNN-LSTM network could be used to process the historical data, with each CNN extracting different spatial features from the data. These features could include patterns, trends, and anomalies, and could be extracted at different scales and resolutions. The output of the CNNs would then be passed to an LSTM network, which would process the sequence of spatial features over time, and use them to make predictions about the future price of the stock.

One of the key advantages of this type of architecture is that it allows the CNNs and the LSTM network to learn complementary spatial and temporal features, respectively. By extracting multiple spatial features from the input data, the CNNs are able to capture a rich and diverse set of patterns and trends. The LSTM network can then use these features to make more accurate predictions, by taking into account the temporal dynamics of the data (Mo et al., 2020).

Furthermore, multi-head CNN-LSTM networks are able to handle large-scale datasets, and can make efficient use of modern computing architectures. The use of multiple parallel CNNs allows the network to process the data in a distributed manner, making it possible to scale the network to large datasets. This makes multi-head CNN-LSTM networks a valuable tool for tasks involving financial time series data, where the amount of data can be very large.

5.6 Tuning Hyper Parameters

Hyperparameters are configurations that are external to a model, and are not estimated with data (Brownlee, 2017). Setting these parameters correct are essential to good model prediction, as such several techniques have been developed to improve beyond manual network tuning, such as Grid-Search, Randomized-Search. These very computationally expensive, and not feasible in a multi-dimensional search space on real-world data (Tripathi & Sharma, 2022).

For all but one model, hyper-parameters were set by using commonly successful settings. To explore if a gain in performance could be had with hyperparameter optimization, one LSTM model was trained with estimated optimal hyperparameters:

To estimate the optimal hyperparameters, a Bayesian optimization technique was employed (Snoek et al., 2012). Bayesian Optimization uses previously calculated hyperparameter values to guide the search for optimal parameters. The following hyperparameters were optimized: learning rate, number of LSTM layers, number of LSTM units (neurons) per layer, batch size, and dropout rate. The method of optimization was such: A search space for each hyperparameter was defined (see table 5.1). A Bayesian Optimization was performed for 400 iterations, each model being trained for 15 epochs. The ten best performing sets of hyperparameters were saved for further assessment.

The reason for not estimating hyperparameters for all models, and for more iterations was the compute-time required. Estimating hyper-parameters for all models for 2000 iterations was estimated to take over 2 years of compute-time, and was deemed unfeasible.

Hyper-parameter	Type	Bounds	Values	Description
Learning rate	Range	[1e-6, 1e-3]	-	Learning rate to adjust the weights of the network
Number of layers	Range	[1, 5]	-	Number of LSTM layers in the network
Number of LSTM units per layer	Range	[5, 512]	-	Number of neurons to be used in a layer
Batch size	Choice	-	[8, 16, 32, 64, 128]	Batch Size or the count of samples to be used in one pass
Dropout rate	Range	[0, 0.30]	-	Dropout regularization rate to ignore randomly selected neurons

Table 5.1: Hyperparameters optimized.

Initial early experiments had shown several models which exhibited epochwise double decent. This can occasionally be observed in neural networks in cases where the number of parameters (features) that are estimated is large (Stephenson & Lee, 2021). The ten best sets of hyperparameters were further trained for 1100 epochs, to assess whether they exhibited epochwise double decent.

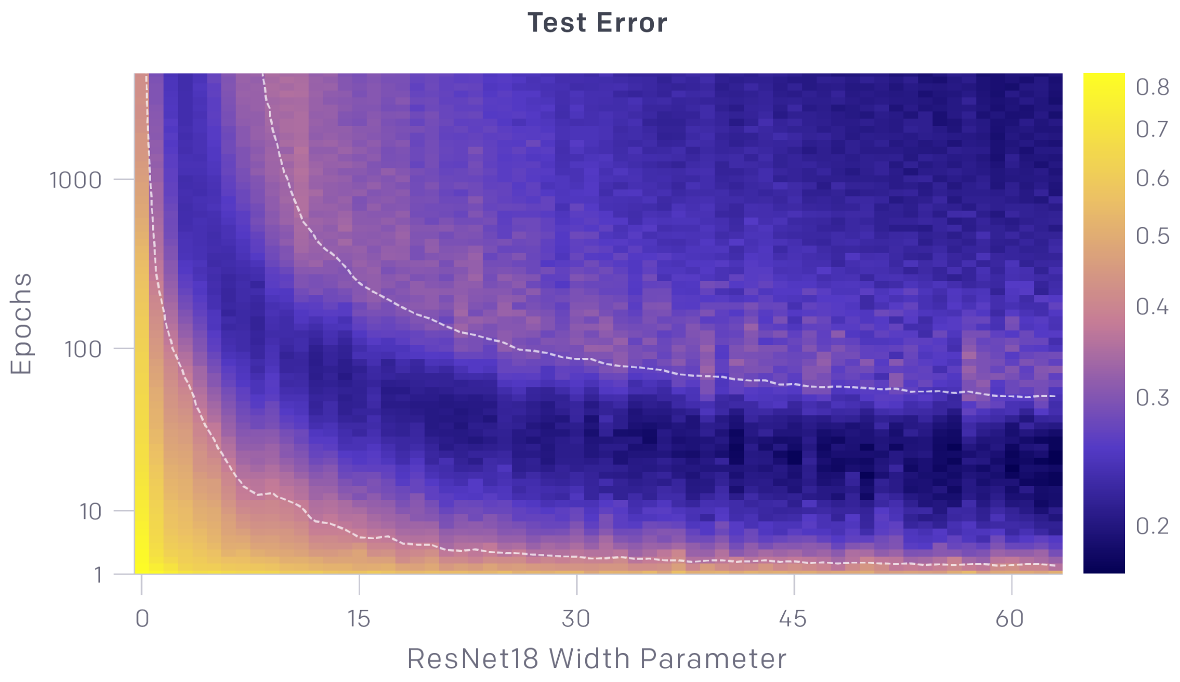


Figure 5.2: A case of epochwise double descent (Nakkiran et al., 2019). Initial training data indicates overfitting, but further training reverses this overfitting.

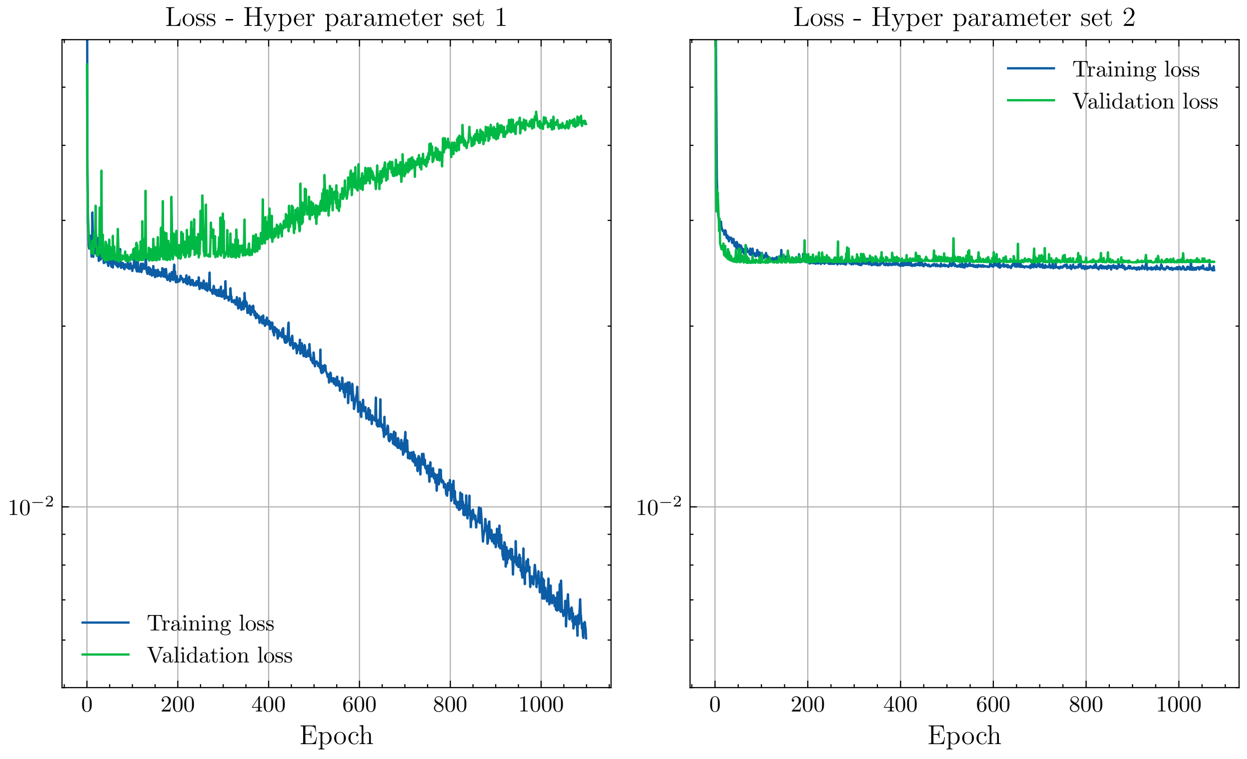


Figure 5.3: Loss function for the best and second-best hyperparameters

None of the sets of optimal hyperparameters showed signs of epochwise double descent.The first set of optimal hyperparameters exhibited fast overfitting, and was therefore disgarded, the second set was used for further study. All hyperparameters and loss functions can be found in the appendix.

Hyper-parameter	1	2	3	4	5	6	7	8	9	10
Learning rate	6.7e-4	2.0e-5	9.7e-6	1.7e-5	4.7e-5	1.4e-5	7.0e-4	2.8e-5	4.4e-5	5.3e-4
N. layers	1	5	5	5	5	4	5	5	5	5
N. units	150	158	40	333	42	149	254	182	490	395
Batch size	8	8	8	8	16	16	128	8	8	128
Dropout rate	0.00	0.06	0.06	0.14	0.14	0.28	0.26	0.30	0.30	0.30

Table 5.2: Hyperparameters optimization results. From best to worst.

5.7 Measuring performance

The chosen set of hyperparameters were trained with an early stopping function, stopping training after predictive performance on the validation data did not improve for 15 epochs. To measure performance, three performance metrics were used, mean absolute error (MAE), mean square error (MSE), and root mean square error (RMSE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{5.10}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{5.11}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \tag{5.12}$$

Where y_i is the ith observation of y, and $\hat{y}_i$ is the ith predicted value of y. The model performance was compared with the mean logreturn of the training period, and naïve approach where all returns are estimated as 0.

6 Results and Discussion

Five models were trained, including one hyperparameter optimized LSTM model (LSTM1). The hyperparameter optimized LSTM model ultimately performed worse than the model with hyperparameters set by convention, showing the limitation of the hyperparameter estimation techniques employed in this thesis, likely coming down to a lack of compute time, as discussed in the previous chapter.

Metric	**CNN**	LSTM1	LSTM2	**CNN-LSTM**	MH-CNN-LSTM	Naive	Mean return
MAE	.6718	.6783	.6777	.6763	.6871	.6796	.6772
MSE	.4527	.4613	.4605	.4587	.4742	.4631	.4598
RMSE	.6728	.6792	.6786	.6773	.6886	.6805	.6781

Table 6.1: Performance metrics results. Models that outperformed the mean return method in bold.

While the hyperparameter optimized LSTM model (LSTM1) was able to beat the naïve approach, it did not perform better than the mean return model.

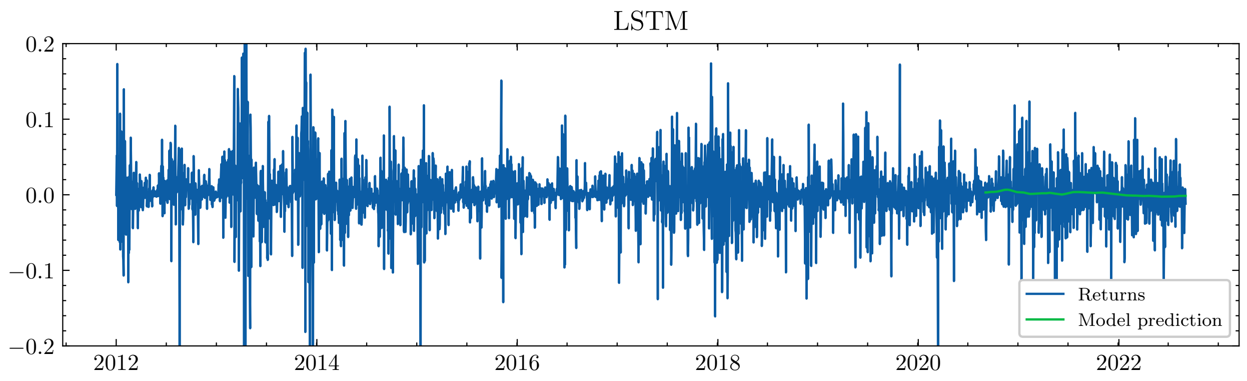


Figure 6.1: LSTM1 (Hyperparameter optimized): Target prediction vs actual prediction. Hyperparameter optimized LSTM network.

The LSTM2 model, where hyperparameters were set by the author, did predict better than the naive approach, and the hyperparameter optimized LSTM model, LSTM1, but was not able to predict returns better than the simple mean return.

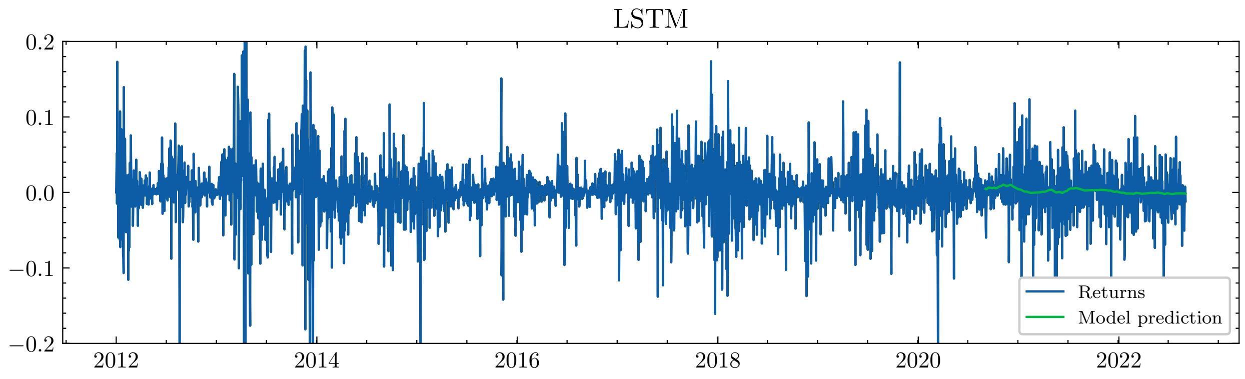


Figure 6.2: LSTM2: Target prediction vs actual prediction.

The CNN model performed surprisingly well, considering its relative simplicity, exhibiting detailed and rapid changes in the predicted value, while still remaining relatively good performance.

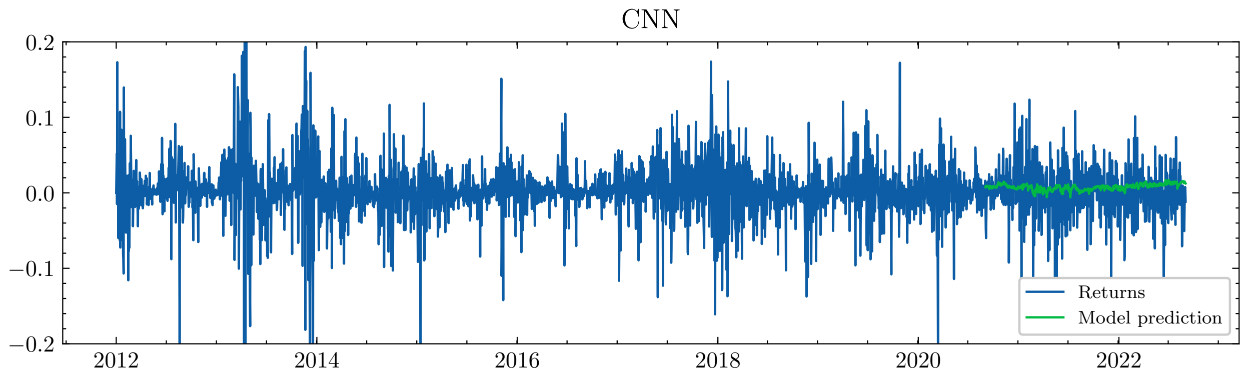


Figure 6.3: CNN: Target prediction vs actual prediction

The CNN-LSTM model provided an interesting combination of features of both the CNN model and the LSTM models. It exhibited larger changes, similar to the CNN model, however they are much more smoothed out compared to the CNN model. It seems this model is good predicting the trend of the returns.

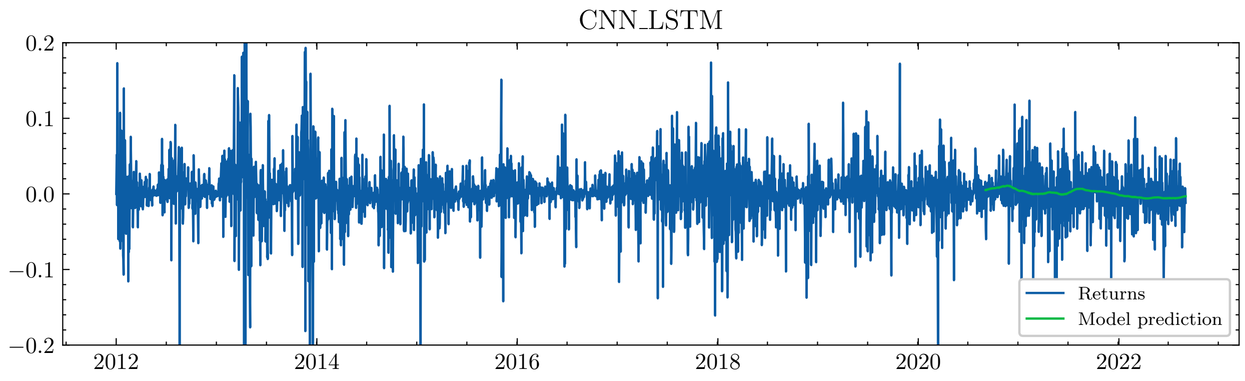


Figure 6.4: CNN-LSTM: Target prediction vs actual prediction

The Multi-Head-CNN-LSTM model exhibited strong signs of epochwise double descent. It is likely that this model can be improved again by significantly increasing the training period, although this requires significant time investment due to the size and complexity of the model. The model was the worst performing model, it is however, likely insufficiently trained to maximize its performance.

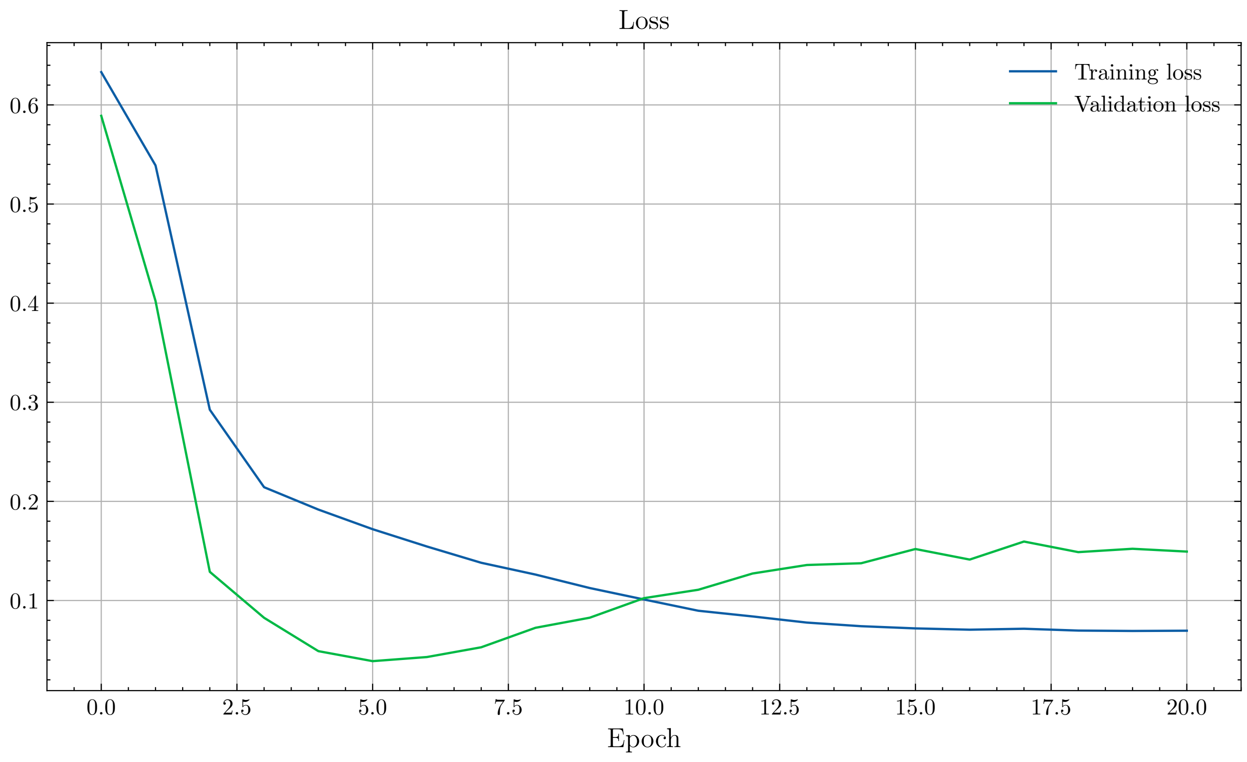


Figure 6.5: MH-CNN-LSTM: Loss vs. Epoch during training.

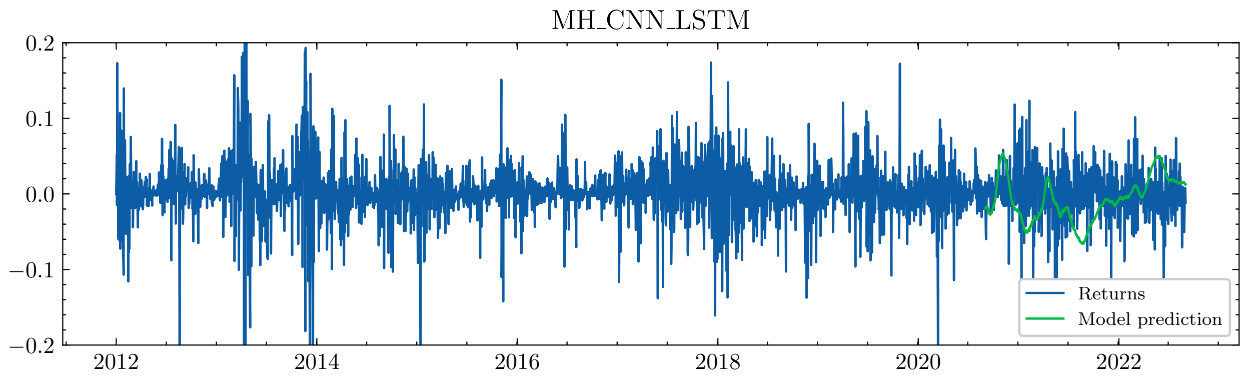


Figure 6.6: MH-CNN-LSTM: Target prediction vs actual prediction

A tendency of the models that underperformed, such as LSTM1, were to predict very close to the mean, with little seasonality, it appears these models were not able to learn significant information from the training set, and as such simply attempted to estimate the mean return with some minor variation. We can see this trend of estimating the mean, and then reducing the variance in the predictions across the time-series in the following 3D-plot:

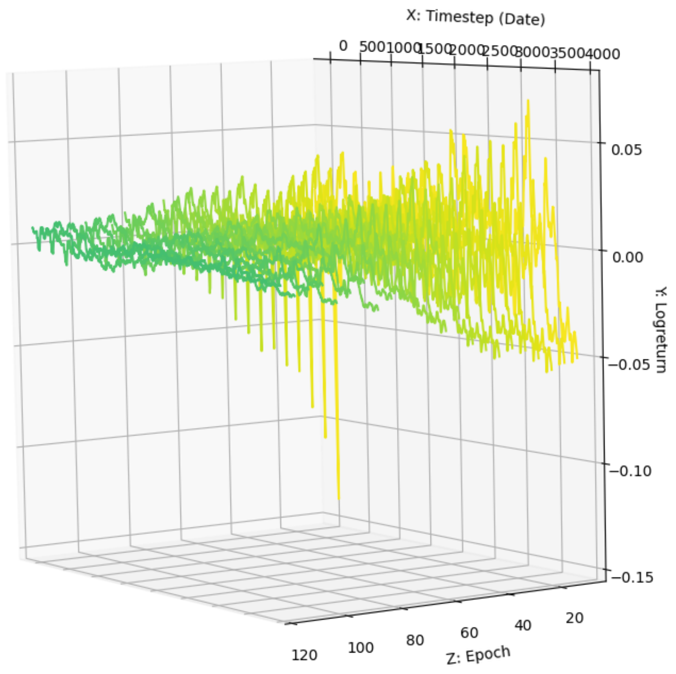


Figure 6.7: LSTM1: Evolution of prediction during training.

Note the orientation of the axis, lines further left are deeper into the training. In contrast, models, such as CNN that were able to gain information from the data did not exhibit this feature:

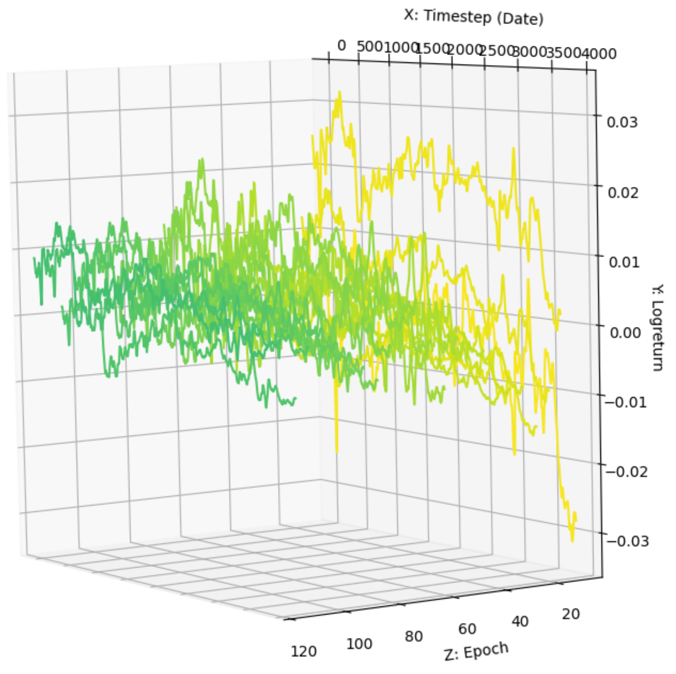


Figure 6.8: CNN: Evolution of prediction during training.

These findings are interesting in light of the Efficient Market Hypothesis (EMH). This research adds to the literature challenging the notions put forth i the EMH, by demonstrating that artificial neural networks (ANNs) can be used to make more accurate predictions of time series data, such as Bitcoin returns, blockchain data, and social media sentiment.

Five ANN models were trained on time series data, including a convolutional neural network (CNN) model and a CNN-long short-term memory (LSTM) model. The results showed that both the CNN and CNN-LSTM models outperformed a model based on the mean return of the data. This finding suggests that the data contains information that can be used to make more accurate predictions than the EMH would predict.

One possible explanation that could lead to such a scenarios is that the data used in the study contains information that is not publicly available, and therefore cannot be fully reflected in asset prices. However, the data used in this study has been purely from publicly available sources, and as such this does not apply. Further research where the models are tested as trading strategies could be used to see if the models truly beat the market.

Furthermore, this thesis challenges the notions put forth by Israel et al., 2020, who argued that return prediction is a “small data” problem, because the amount of predictor variables does not change the class of problem to a “big data” problem, to do that one would need a large set of variables to try to estimate. In this thesis, we do exactly this by applying Artificial Neural Networks to a broader set of features, to leverage the ability of ANNs to infer relations among large datasets. It may be beneficial for investors and firms to attempt to expand the breadth and depth of data that is collected, so that they also can leverage ANN models in traditional markets such as stock exchanges or bond markets.

7 Conclusions

This thesis had as a goal to explore the ability of various Artificial Neural Networks to model financial data, by leveraging a large dataset. Several models, including CNN and CNN-LSTM were able to outperform a simple mean return, and the other models. When machine learning has been applied to finance, there has been a trend of not being able to fully utilize the depth and complexity of Neural Networks. This is because financial data is often very limited in frequency and span (Israel et al., 2020). This thesis was able to overcome some of the limitations suggested by Israel et al., 2020, by leveraging a richer dataset than what is commonly used. Two networks, CNN and CNN-LSTM performed exceptionally well, beating the expected return model.

Artificial neural networks (ANNs) are a type of machine learning algorithm that are modeled after the structure and function of the human brain. In finance, ANNs are used for a variety of purposes, including identifying market trends, analyzing financial data, and making predictions about future market movements. ANNs are particularly useful in finance because they can process large amounts of data quickly and accurately, and can learn and adapt to new information over time. Despite their many advantages, ANNs also have limitations, such as a lack of transparency in their decision-making process, which can make them difficult to interpret and explain to non-experts. Overall, ANNs are a valuable tool for financial professionals, but they should be used carefully and in conjunction with other methods of analysis.

A Appendix

A.1 Python Code

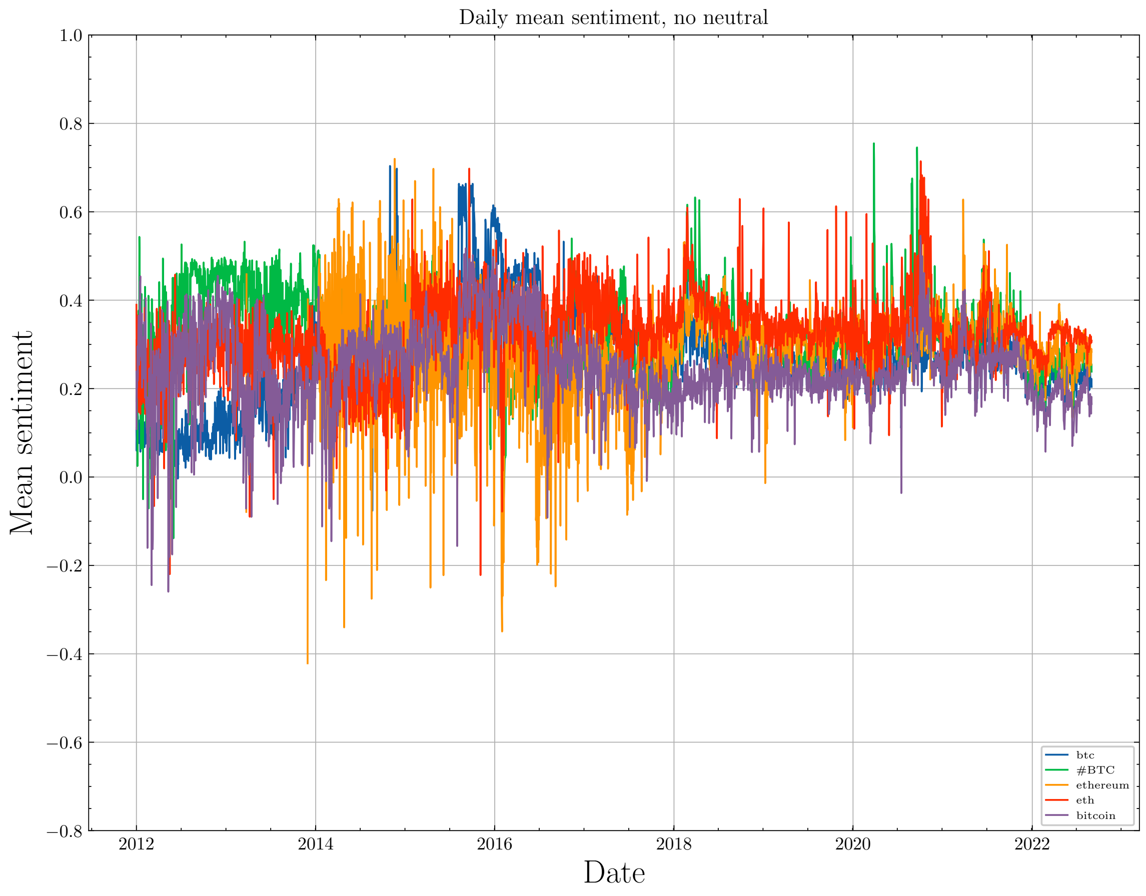
This thesis has been implemented using Python 3.10.8. The machine learning is implemented in Tensorflow 2.9.1. The machine used was running Windows 10, with 32GB of system memory, and an RTX 3070 graphics card.

The full codebase for the project is made publicly available on GitHub:

GitHub/btc-return-prediction-LSTM-masters-thesis

GitHub/btc-return-prediction-CNN-masters-thesis

A.2 Sentiment data with and without neutral sentiments



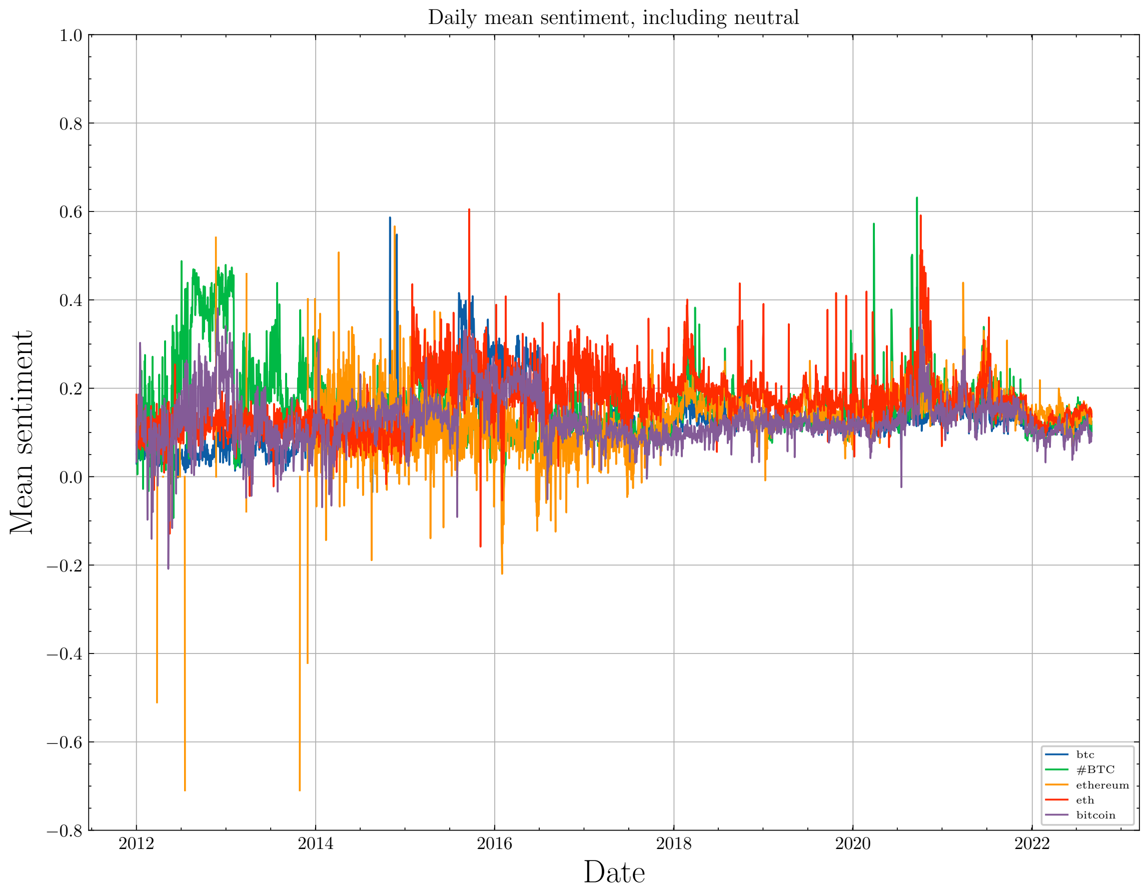
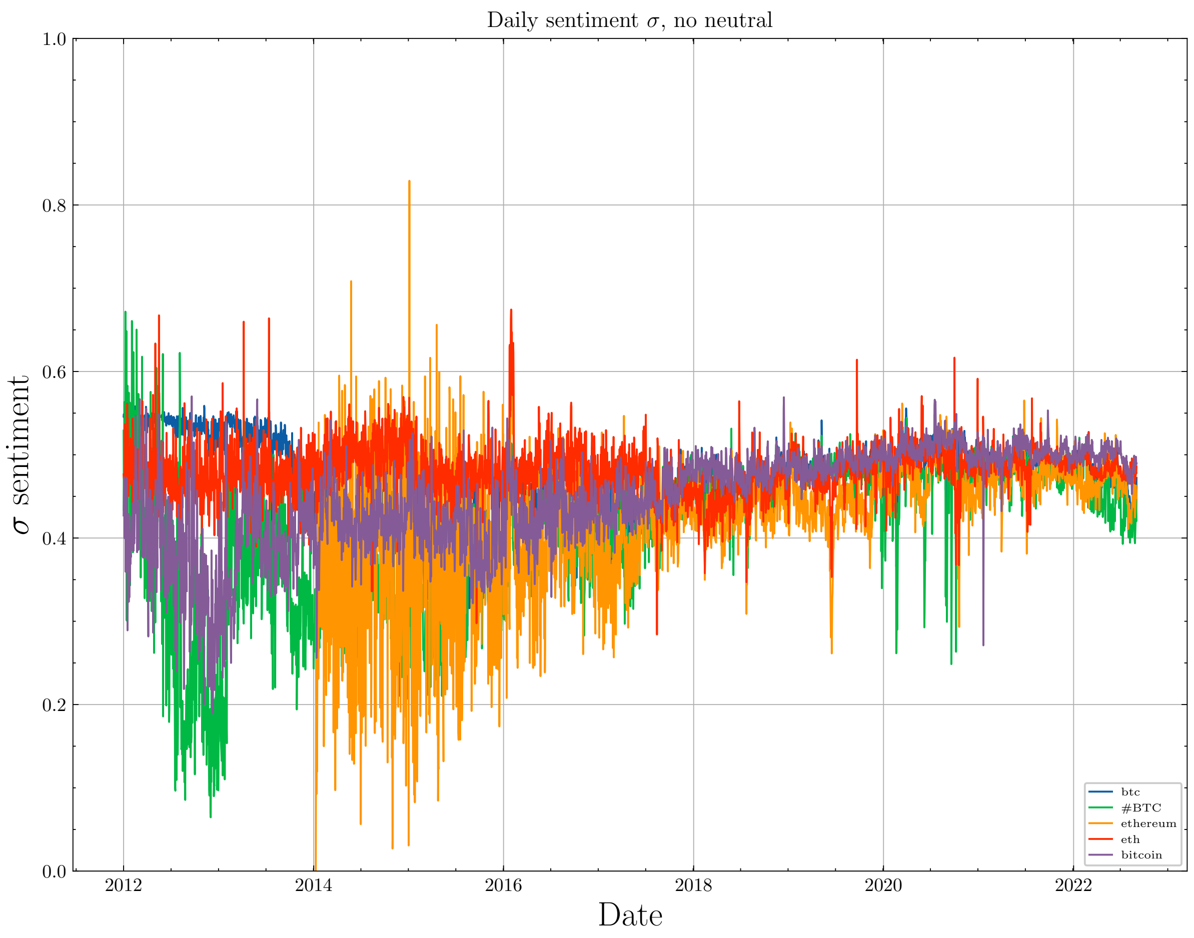


Figure A.1: Daily mean sentiment values, comparing including and excluding *polarity* $= 0$



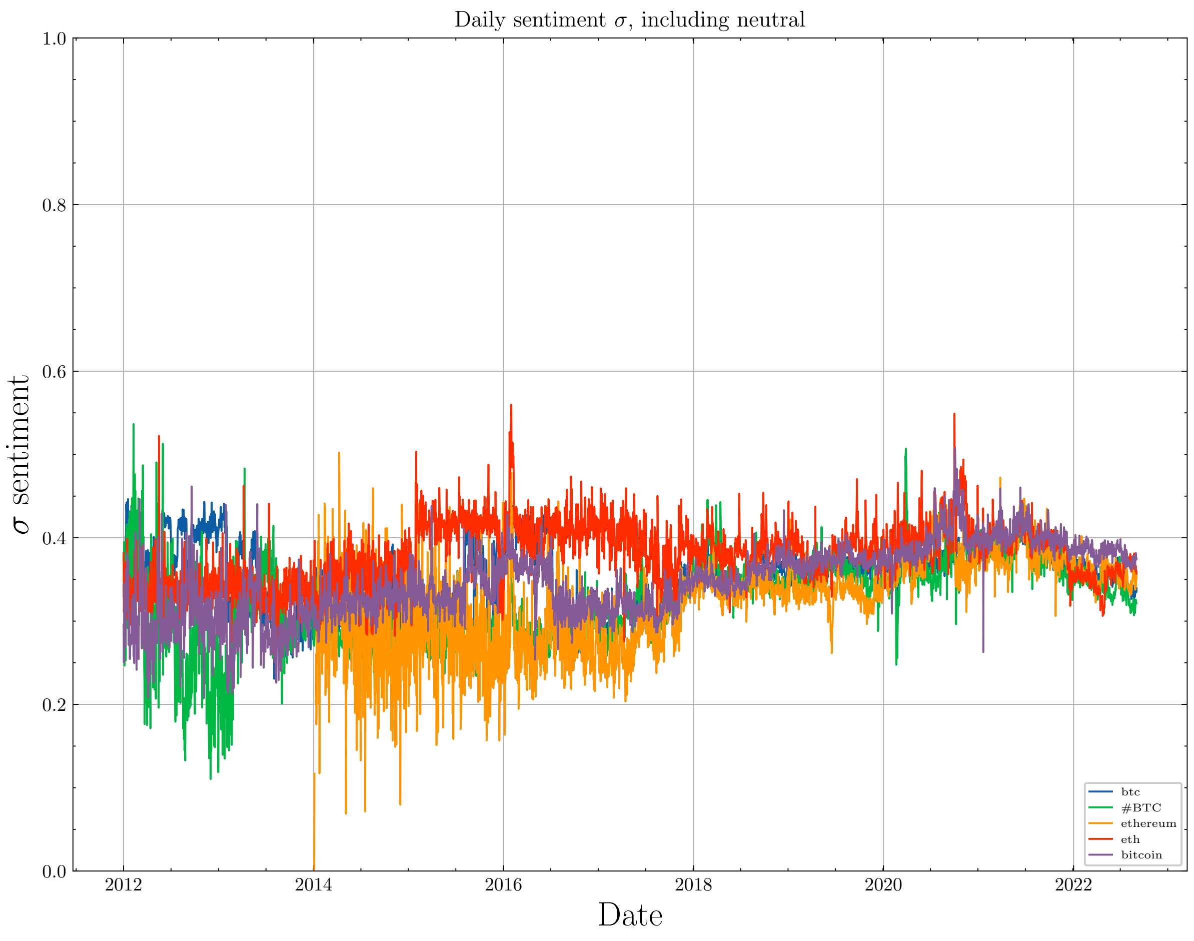
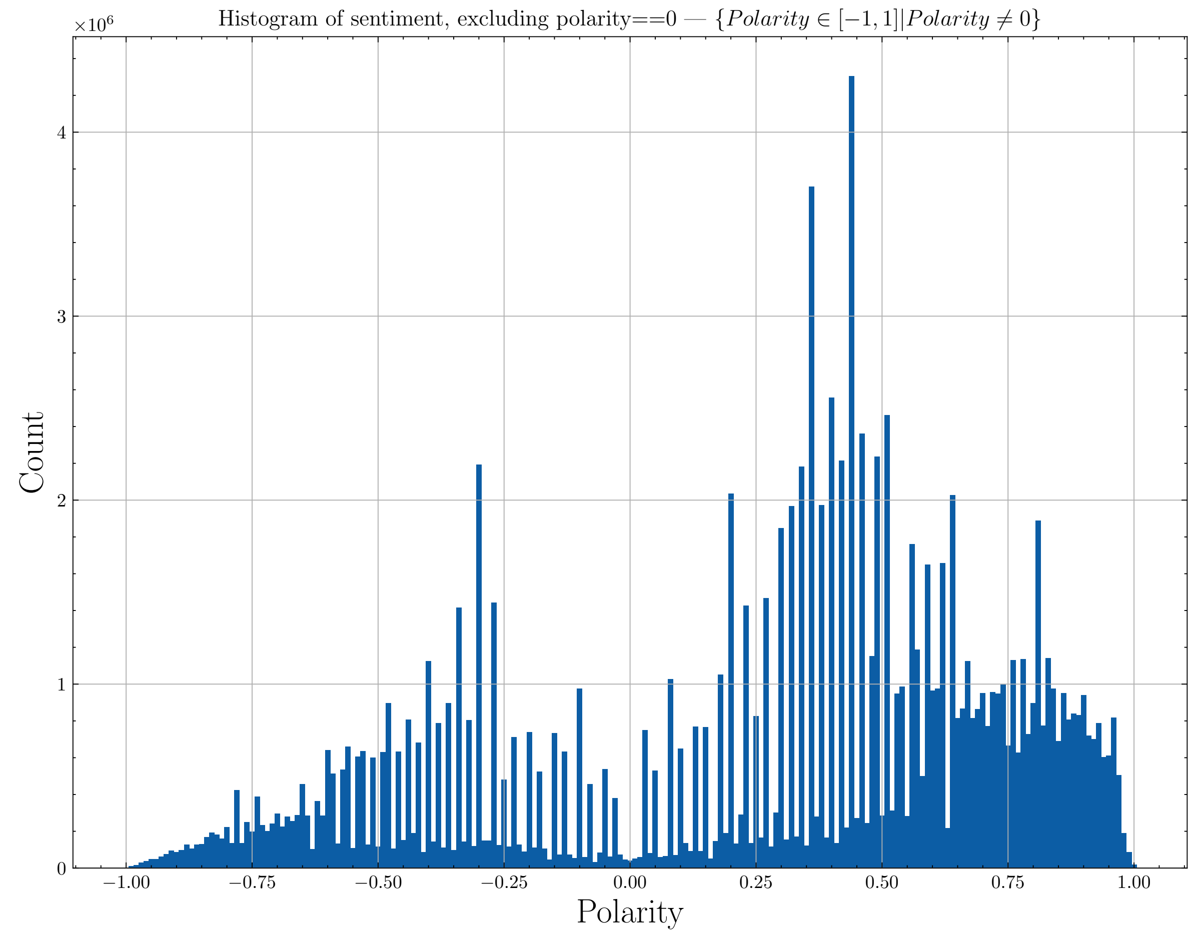


Figure A.2: Daily mean sentiment std. dev., comparing including and excluding $polarity = 0$



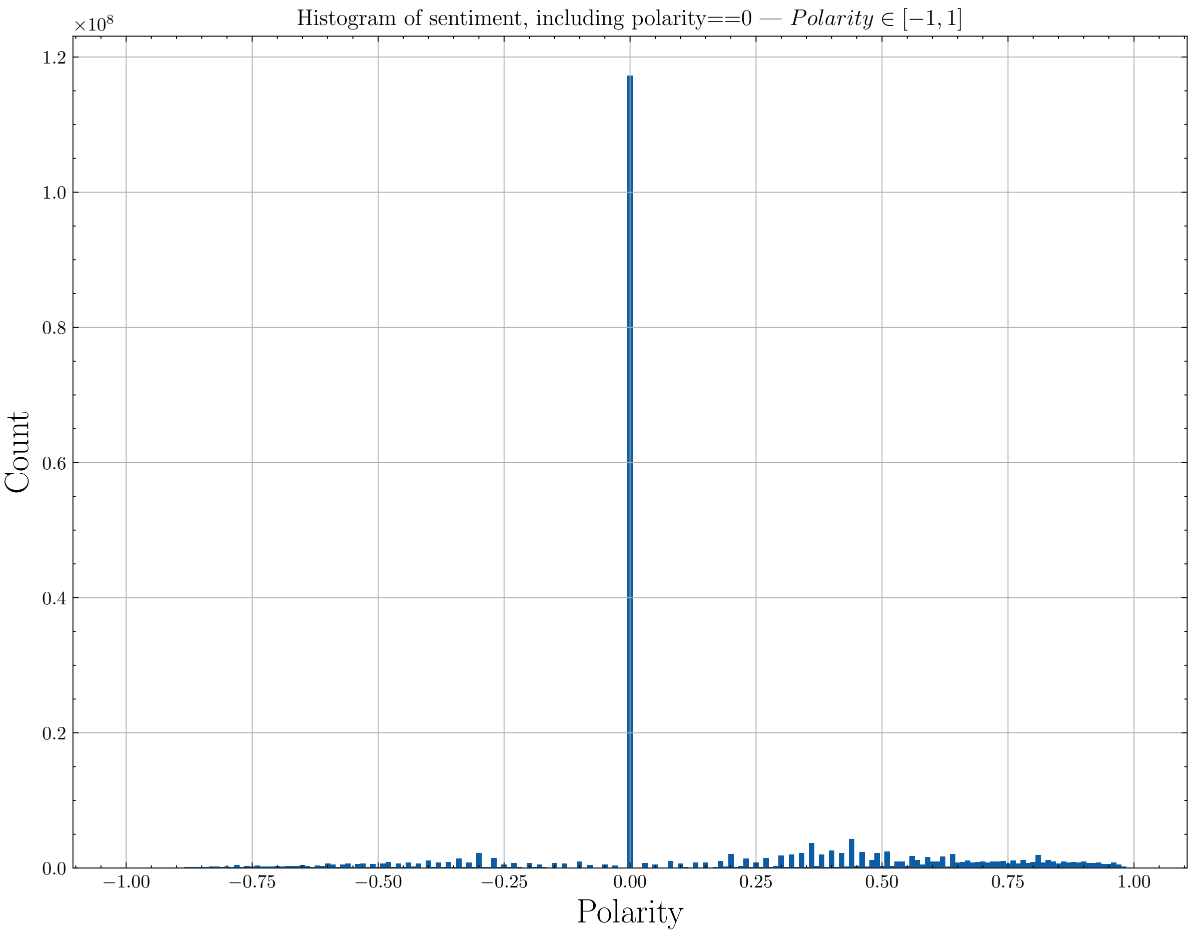


Figure A.3: Histogram of all sentiment values, with and without $polarity = 0$

A.3 Loss Function for 10 optimal hyperparameters

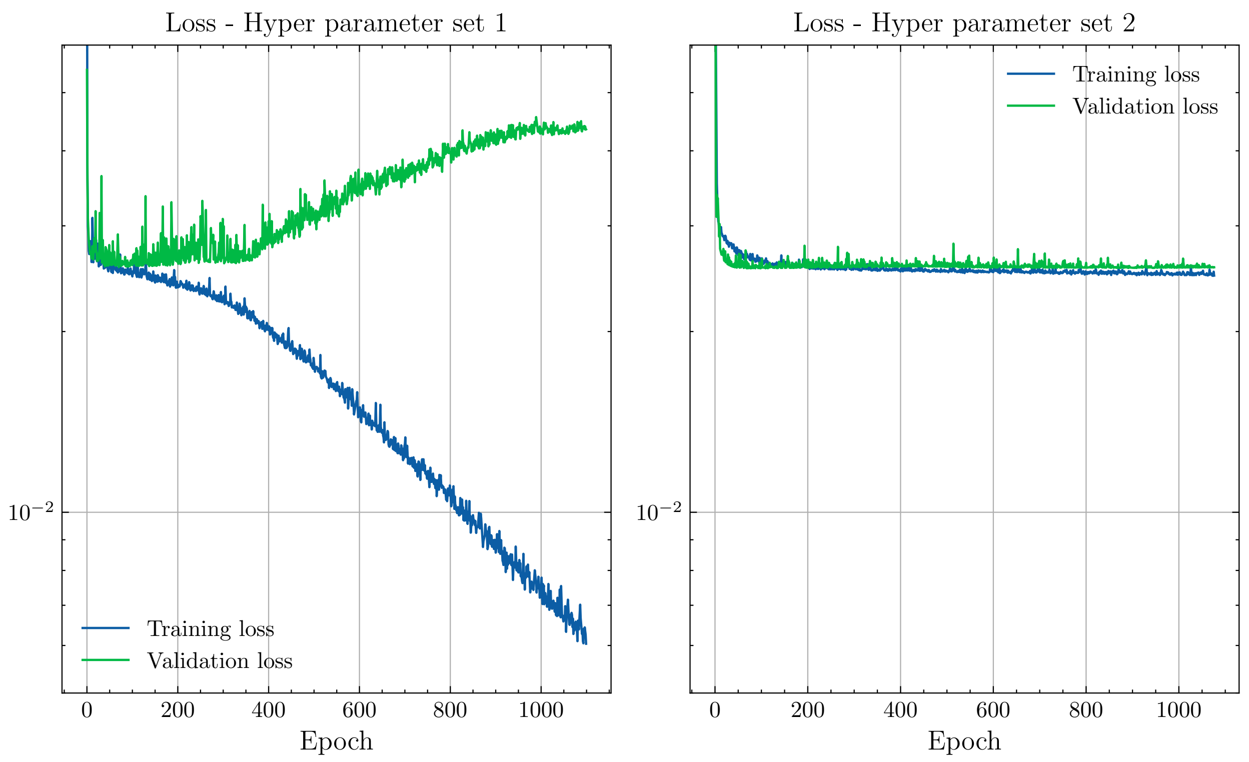


Figure A.4: Loss functions for hyperparameter set 1 and 2.

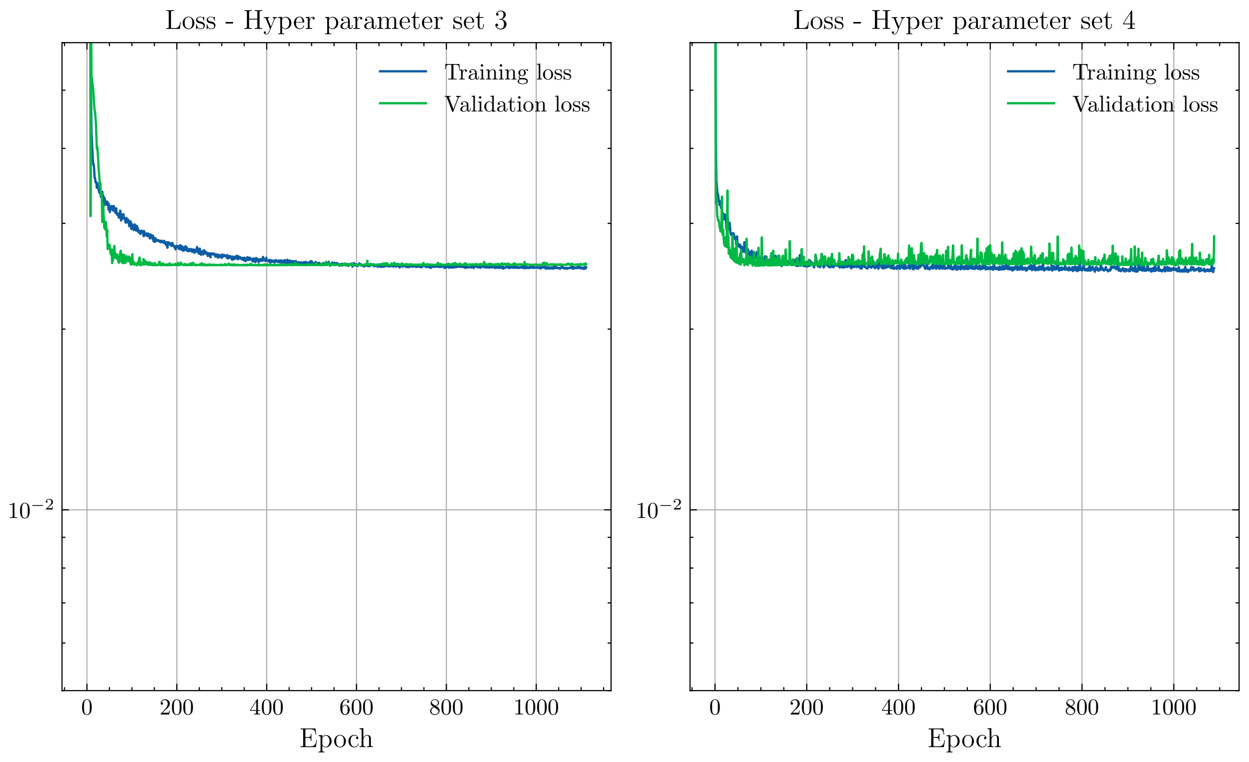


Figure A.5: Loss functions for hyperparameter set 3 and 4.

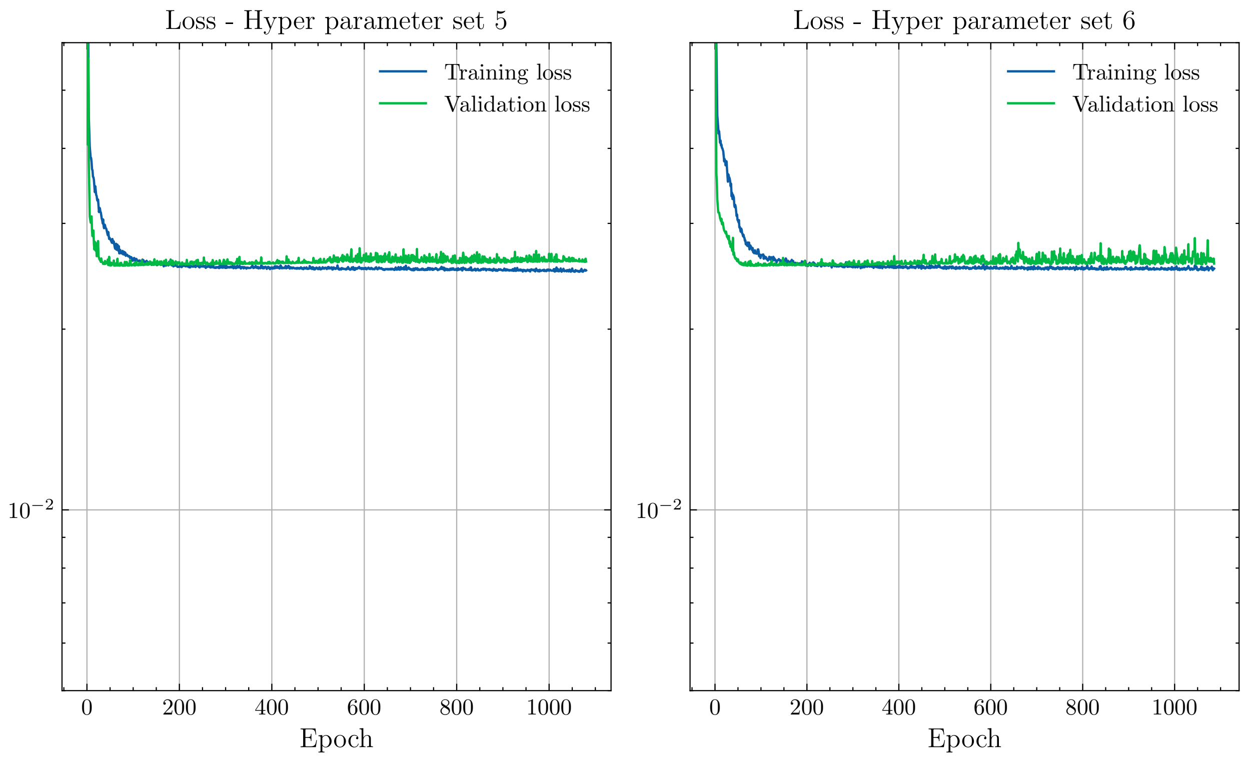


Figure A.6: Loss functions for hyperparameter set 5 and 6.

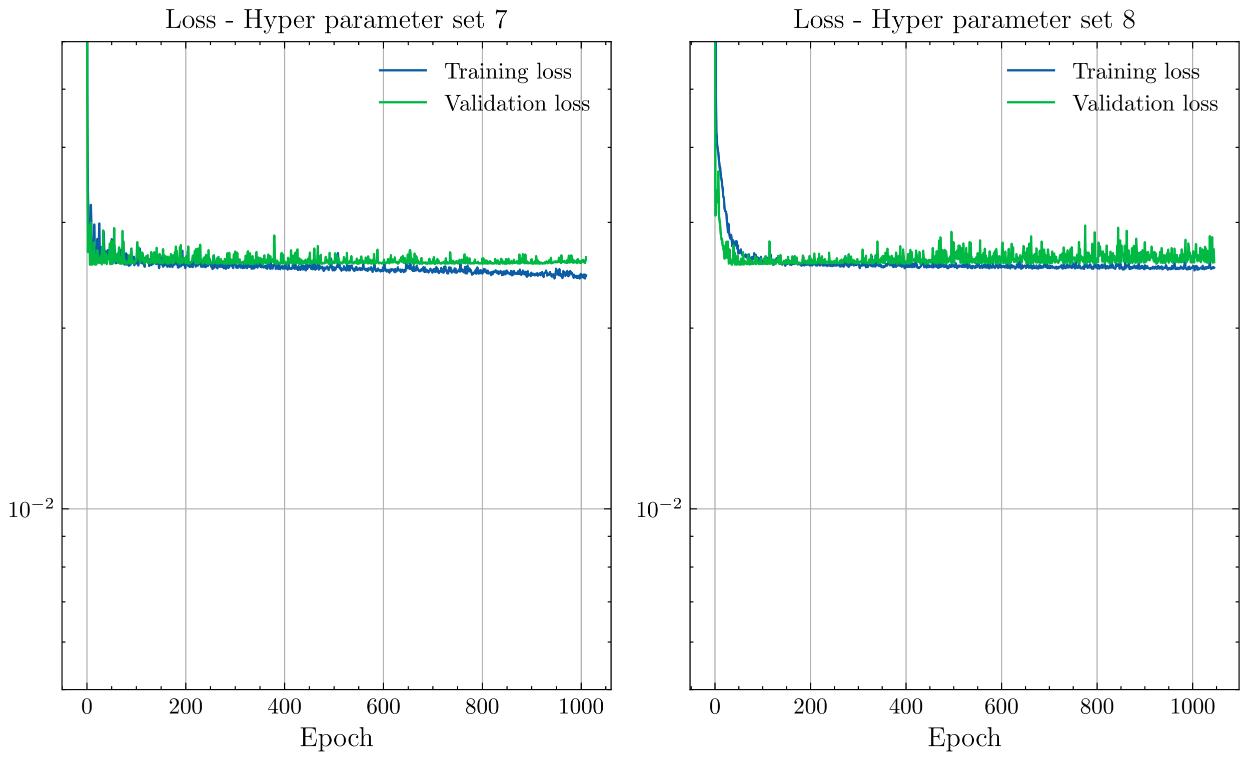
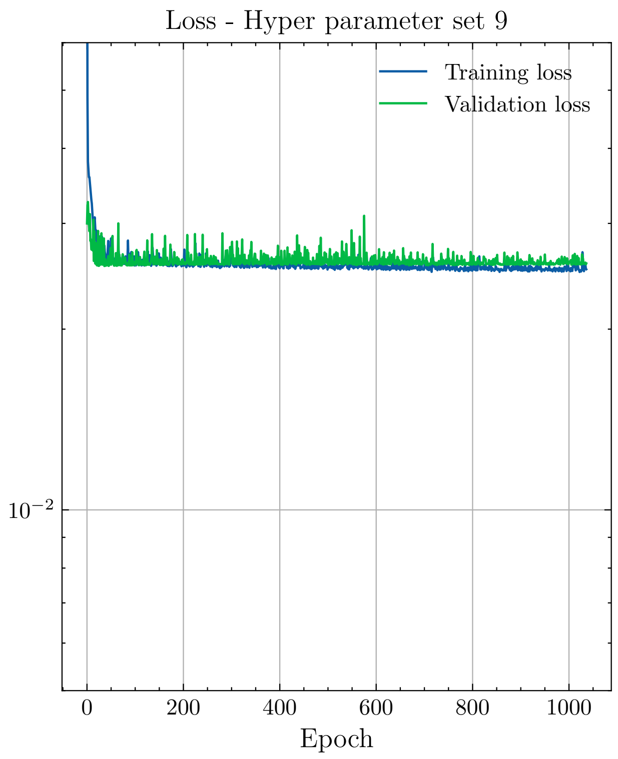


Figure A.7: Loss functions for hyperparameter set 7 and 8.



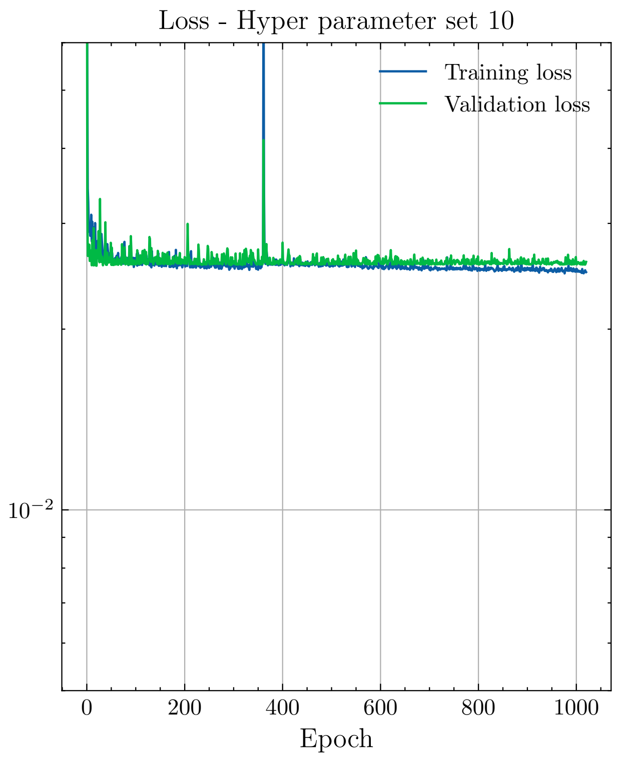


Figure A.8: Loss functions for hyperparameter set 9 and 10.

B Discussion Paper

B.1 Daniel Lindestad - Topic: International

This discussion paper is written as a mandatory part of the master‘s program in business and administration with specialization in analytical finance. To get this master thesis approved, there has been a couple of requirements. Such as a mandatory meeting with supervisors, an oral presentation of the thesis, and writing a discussion paper. This discussion paper sheds light on some of insights that have been accumulated during the master‘s program, and the development of the thesis. This discussion paper also serves as a means of a self reflection for the author about his experience in the master‘s program.

This master thesis explores machine learning predictions on large datasets. Machine learning has been able to develop comprehensive models that can make astonishing predictions in fields like text and speech recognition, and image classification. These models rely on large datasets to be able to create models with such great applications. When machine learning has been applied to finance, there has been a trend of not being able to fully utilize the depth and complexity of Neural Networks. This is because financial data is often very limited in frequency and span. When looking at returns one might only have one data point per day (or per month), if only a few features are available then little information can be extracted from this. This thesis attempts to remedy this limitation by broadening the feature selection available, by using a prediction target where data on fundamental factors are more easily available, including market perception, through sentiment analysis.

In the thesis, a rich dataset is gathered on historical Bitcoin prices, market sentiments through social media sentiment analysis, and various fundamental features of the underlying Bitcoin infrastructure, in the form of blockchain information. LSTM Neural Networks are trained on the data, and hyperparameters are optimized. A final model is trained, and compared against simpler methods of estimating returns.

Machine learning as a topic is currently of massive importance internationally. Developing intelligent algorithms to automate procedures has been a key factor to gain competitive advantage in numerous business fields (Attaran & Deb, 2018). An increasing number of everyday functions are being automated, a lot of this automation is powered by machine learning. The applications of Neural Networks will likely change the lives of most people drasticly, be this in the form of better healthcare through better diagnistoc abilities and better early warning of cancers and diseases such as Alzheimer‘s, or through self driving cars that lower the amount of automotive related deaths. It is of paramount importance for Norway to take part in this international endeavour, both through helping businesses make use of currently available technologies, and becoming a leader in research within the field, through strengthening national academia, and aiding in international academic collaboration.

The master‘s program included many courses where "international" was a major emphasis. For instance, in the first year of the program, we covered a topic called sustainable capitalism. The first half of this course was divided between lectures and oral presentations, while the second half involved individual work on a term paper that covered environmental concerns. Here, we discovered the value of international cooperation in lowering global pollution levels. The issue is seen as a global issue since it imperils all forms of life on Earth. Thus, international cooperation is also required to address these difficulties.

Over the past several centuries, technology has advanced rapidly, and as a result of digitization, the globe is becoming ever more connected. Private investors' ability to buy and sell stocks and funds has significantly increased as additional trading platforms have entered the market. Only 20 years ago, placing an order required calling your broker; today, it only takes a few clicks on your computer, tablet, or phone. Additionally, this has increased global connectivity.

Money markets are inherently connected at an international level, meaning that crises that emerge from international factors, will still impact the national economy. We can see this with the Oslo Stock Exchange, as it is impacted by what happens in other nations, all the funds are somehow linked to the global market. For example, the COVID-19 pandemic, the 2015 Greek financial crisis, and the 2008 financial crisis all had an impact on the financial markets of all nations. That's because everyone is now connected.

Governments, notably their monetary policies, have perhaps the biggest influence on financial markets. The extent to which governments control the free market has become abundantly obvious. The monetary and fiscal policies that governments and their central banks have implemented have a major impact on the financial sector. By raising or lowering interest rates, the US Federal Reserve, for instance, may really slow down or speed up economic growth in the nation.

Financial markets act as the conduit between monetary policy and the actual economy. Central banks frequently set objectives to maintain an inflation rate constant at about 2% in order to maintain the stability of financial markets as much as feasible. The 10-year inflation-indexed bond will always represent a 2% inflation rate in ten years if the credibility of the central bank is such that market participants believe inflation will remain set at 2% over the long term. However, if central banks were to grow complacent with the idea that market assumptions represent steady inflation expectations, they may eventually stray from the best course of action. Markets will finally see the fault in the policy. We have recently seen how inflation has soared, and how both governments and markets have struggled to tackle this challenge. The UK "minibudget" incident showed the world that governments may not be able to easily perceive how risky the markets view some of their policy actions. This serves as a warning to other governments to tread very carefully.

The COVID-19 pandemic has sped up the move for many businesses to new forms of remote work, whether it is for meetings or just doing work from home when necessary. This will probably continue to have an impact on how funds operate and are provided to consumers, whether they need fewer employees or can more efficiently manage their production. Since index funds are so inexpensive, the funds will probably need to reduce expenses everywhere they can so that they can truly provide the consumers with the services they have paid for. Future fund decisions will certainly be influenced by technological developments.

In conclusion, the international financial system is enormous and complex. Techniques that lie in the field of machine learning, such as Neural Networks might be able to help us develop tools and methods to navigate it‘s complexity. This thesis has attempted to be a small part of that puzzle. Through this it also serves as an aid in helping the University of Agder in being an international force in academia.

Finally, I would like to express my gratitude to the School of Business and Law at the University of Agder. Here I have experienced five years of academic, social, and personal development. I feel well prepared to enter the work-force after taking these courses, and I have no doubt that the information I have gained over the last five years will be useful to me in my future undertakings. Although there were highs and lows throughout my stay, it has been a period I will cherish and be proud of.

Bibliography

Attaran, M., & Deb, P. (2018). Machine learning: The new 'big thing' for competitive advantage. *5*, 277–305. https://doi.org/10.1504/IJKEDM.2018.10015621

Brownlee, J. (2017). What is the difference between a parameter and a hyperparameter? https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

De Bondt, W. F. M., & Thaler, R. (1985). Does the stock market overreact? *The Journal of Finance*, *40*(3), 793–805. https://doi.org/https://doi.org/10.1111/j.1540-6261.1985.tb05004.x

Dickey, D., & Fuller, W. (1979). Distribution of the estimators for autoregressive time series with a unit root. *JASA. Journal of the American Statistical Association*, *74*. https://doi.org/10.2307/2286348

Djordjevic, K., Jordovic Pavlovic, M., Cojbasic, Z., Galovic, S., Popovic, M., Nesic, M., & Markushev, D. (2022). Influence of data scaling and normalization on overall neural network performances in photoacoustics. *Optical and Quantum Electronics*, *54*. https://doi.org/10.1007/s11082-022-03799-1

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/https://doi.org/10.1016/0364-0213(90)90002-E

Fama, E. F. (1965). The behavior of stock-market prices. *The Journal of Business*, *38*(1), 34–105. Retrieved December 14, 2022, from http://www.jstor.org/stable/2350752

Frostig, R., Johnson, M. J., Maclaurin, D., Paszke, A., & Radul, A. (2021). Decomposing reverse-mode automatic differentiation. https://doi.org/10.48550/ARXIV.2105.09469

Geron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and TensorFlow* (2nd ed.). O'Reilly Media.

Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying lstm to time series predictable through time-window approaches. In G. Dorffner, H. Bischof, & K. Hornik (Eds.), *Artificial neural networks — icann 2001* (pp. 669–676). Springer Berlin Heidelberg.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232. https://doi.org/10.1109/TNNLS.2016.2582924

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hutto, C., & Gilbert, E. (2015). Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014*.

Israel, R., Kelly, B., & Moskowitz, T. (2020). Can machines 'learn' finance? *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3624052

Jang, H., & Lee, J. (2018). An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information. *IEEE Access*, *6*, 5427–5437. https://doi.org/10.1109/ACCESS.2017.2779181

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. technical report, june 1985-march 1986. https://www.osti.gov/biblio/6910294

Jr., G. S. S., & Yoon, Y. (1992). Applying artificial neural networks to investment analysis. *Financial Analysts Journal*, *48*(5), 78–80. https://doi.org/10.2469/faj.v48.n5.78

Kienzler, R. (2018). Keras lstm tutorial – how to easily build a powerful deep learning language model. https://adventuresinmachinelearning.com/keras-lstm-tutorial/

Kwiatkowski, D., Phillips, P. C., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, *54*(1), 159–178. https://doi.org/https://doi.org/10.1016/0304-4076(92)90104-Y

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 9459–9474, Vol. 33). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series.

Minsky, M., & Papert, S. (1969). *Perceptrons*. M.I.T. Press.

Mo, H., Lucca, F., Malacarne, J., & Iacca, G. (2020). Multi-head cnn-lstm with prediction error analysis for remaining useful life prediction. *2020 27th Conference of Open Innovations Association (FRUCT)*, 164–171. https://doi.org/10.23919/FRUCT49677.2020.9211058

Mohapatra, S., Ahmed, N., & Alencar, P. (2020). Kryptooracle: A real-time cryptocurrency price prediction platform using twitter sentiments. https://doi.org/10.48550/ARXIV.2003.04967

Mostafa, M. M. (2010). Forecasting stock exchange movements using neural networks: Empirical evidence from kuwait. *Expert Systems with Applications*, *37*(9), 6302–6309. https://doi.org/https://doi.org/10.1016/j.eswa.2010.02.091

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt. https://doi.org/10.48550/ARXIV.1912.02292

Narayanan, A., Bonneau, J., Felten, E. W., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies*. Princeton University Press.

Olson, D., & Mossman, C. (2003). Mossman, c.: Neural network forecasts of canadian stock returns using accounting ratios. international journal of forecasting. 19, 453-465. *International Journal of Forecasting*, *19*, 453–465. https://doi.org/10.1016/S0169-2070(02)00058-4

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *ArXiv e-prints*.

Pano, T., & Kashef, R. (2020). A complete vader-based sentiment analysis of bitcoin (btc) tweets during the era of covid-19. *Big Data and Cognitive Computing*, *4*(4). https://doi.org/10.3390/bdcc4040033

Pant, D. R., Neupane, P., Poudel, A., Pokhrel, A. K., & Lama, B. K. (2018). Recurrent neural network based bitcoin price prediction by twitter sentiment analysis. *2018 IEEE 3rd*

International Conference on Computing, Communication and Security (ICCCS), 128–132. https://doi.org/10.1109/CCCS.2018.8586824

Rosenblatt, F. (1960). Perceptron simulation experiments. *Proceedings of the IRE*, *48*(3), 301–309. https://doi.org/10.1109/JRPROC.1960.287598

Rundo, F., Trenta, F., di Stallo, A. L., & Battiato, S. (2019). Machine learning for quantitative finance applications: A survey. *Applied Sciences*, *9*(24), 5574.

Serafini, G., Yi, P., Zhang, Q., Brambilla, M., Wang, J., Hu, Y., & Li, B. (2020). Sentiment-driven price prediction of the bitcoin based on statistical and deep learning approaches. *2020 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206704

Sewell, M. (2011). History of the efficient market hypothesis. *Research Note No. RN/11/04). London: UCL Department of Computer Science.*

Shi, M. (2019). Research on parallelization of microblog emotional analysis algorithms using deep learning and attention model based on spark platform. *IEEE Access*, *7*, 177211–177218. https://doi.org/10.1109/ACCESS.2019.2955501

Sitte, R., & Sitte, J. (2000). Analysis of the predictive ability of time delay neural networks applied to the s&p 500 time series. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *30*(4), 568–572. https://doi.org/10.1109/5326.897083

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, 2951–2959.

Spilak, B. (2018). Deep neural networks for cryptocurrencies price prediction.

Stephenson, C., & Lee, T. (2021). When and how epochwise double descent happens. https://doi.org/10.48550/ARXIV.2108.12006

Sussillo, D., & Abbott, L. F. (2014). Random walk initialization for training very deep feedforward networks. https://doi.org/10.48550/ARXIV.1412.6558

Tripathi, B., & Sharma, R. K. (2022). Modeling bitcoin prices using signal processing methods, bayesian optimization, and deep neural networks. *Computational Economics*. https://doi.org/10.1007/s10614-022-10325-8

Velastegui, R., Zhinin-Vera, L., Chang, O., & Pilliza, G. (2020). Time series prediction by using convolutional neural networks. https://doi.org/10.1007/978-3-030-63128-4_38

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. https://doi.org/10.1109/5.58337

White. (1988). Economic prediction using neural networks: The case of ibm daily stock returns. *IEEE 1988 International Conference on Neural Networks*, 451–458 vol.2. https://doi.org/10.1109/ICNN.1988.23959