Zurich University
of Applied Sciences

School of
Engineering
IDP Institute of Data Analysis
and Process Design

Bachelor thesis Engineering and Management

Explainability of artificial intelligence using price predictions of bitcoin with neural networks.

Author	Ken Geeler Pascal Simon Bühler Philipp Rieser
Supervisor	Marc Wildi
Date	11.06.2021

Zurich University
of Applied Sciences

School of
Engineering

DECLARATION OF ORIGINALITY
Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Name Student:**
Winterthur, 11.06.2021	Geeler Ken
Winterthur, 11.06.2021	Pascal Simon Bühler
Winterthur, 11.06.2021	Philipp Rieser

Contents

Abstract

1. Introduction . . . 1
    1.1. Intro . . . 1
2. Theory . . . 2
    2.1. Neural network . . . 2
        2.1.1. Perceptron . . . 2
        2.1.2. Backpropagation algorithm . . . 4
        2.1.3. Multilayer perceptron . . . 6
        2.1.4. Recurrent neural networks (RNN) . . . 6
        2.1.5. Long-short term memory (LSTM) . . . 8
        2.1.6. Challenges . . . 8
    2.2. Model comparison . . . 8
        2.2.1. Sharpe Ratio . . . 9
        2.2.2. Diebold Mariano . . . 9
        2.2.3. Mean Squared Error (MSE) . . . 10
    2.3. Bitcoin . . . 11
        2.3.1. Historical analysis . . . 11
        2.3.2. Bitcoin technology and cryptocurrencies . . . 13
        2.3.3. Valuation and digital gold . . . 14
    2.4. Explainable artificial intelligence . . . 15
        2.4.1. Classic Approach . . . 15
        2.4.2. Explainability for Financial Time Series . . . 16
3. Methodology . . . 21
    3.1. Data exploration . . . 22
    3.2. Network architecture . . . 24
        3.2.1. Defining train and test samples . . . 25
        3.2.2. Evaluating network architecture . . . 26
        3.2.3. Model selection . . . 32
        3.2.3.1 Benchmark . . . 32
    3.3. Trading strategy . . . 33
        3.3.1. Trading with neural networks and LPD . . . 34
        3.3.2. Neural network . . . 34
        3.3.3. LPD signal . . . 35
        3.3.5. GARCH volatility predictions . . . 38
        3.3.4. nn and lpd estimation . . . 40

3.3.6. Kapitel: Resultat nicht wie erhofft 41
4. Results 42
4.1. Results chapterino 42
5. Conclusion 43
5.1. Get rich or die tryin 43
5.2. Be GME stock, or not to be GME stock 43
References 44
Attachment 46

Abstract

1. Introduction

Ken is testing working with Github.does it work now?

1.1. Intro

2. Theory

The following chapter is intended to provide the theoretical foundations necessary for our work. It is divided into a part that provides an overview of artificial neural networks. Followed by section 2.2. which shows the background and the ecosystem of Bitcoin. This knowledge should be kept in mind, which should help in understanding the price formation of bitcoin.

2.1. Neural network

In the context of this work, artificial neural networks are used to answer supervised learning questions that focus on the classification of data. This means that a neural network finds a correlation between the data and their labels and optimizes its parameters to minimize the error for the next try. This process is called supervised training and is performed with a test data sample. An application example of classification is that a neural network is used for face recognition after it has learned the classification of different faces in the process of supervised training. Predictive analysis works similarly to the classification of labeled data. It estimates future values based on past events and can be trained with historical data. On the other hand, unsupervised learning (clustering) is applied to detect patterns from unlabeled data. Based on these patterns, for example, anomalies can be detected that are relevant in the fight against fraud (fraud detection). Unsupervised learning is not discussed further in this paper. Section 2.1.1. will demonstrate the functioning of a neural network using a simple perceptron.

2.1.1. Perceptron

The construction of an artificial neural network is demonstrated using a perceptron. It is a simple algorithm for supervised learning of binary classification problems. This algorithm classifies patterns by performing a linear separation. Although this discovery was anticipated with great expectations in 1958, it became increasingly apparent that these binary classifiers are only applicable to linearly separable data inputs. This was only later addressed by the discovery of multiple layer perceptrons (MLP) [1]. Basically, a perceptron is a single-layer neural network and consists of the following five components and can also be observed in figure 1.

1. Inputs
2. Weights
3. Bias
4. Weighted sum
5. Activation function

Inputs are the information that is fed into the model. In the case of econometric time series, it is mostly the current and historical log returns (lags). These are multiplied by the weights and added together with the bias term to form the weighted sum. This weighted sum is finally passed on to the non-linear activation function, which determines the output of the perceptron.

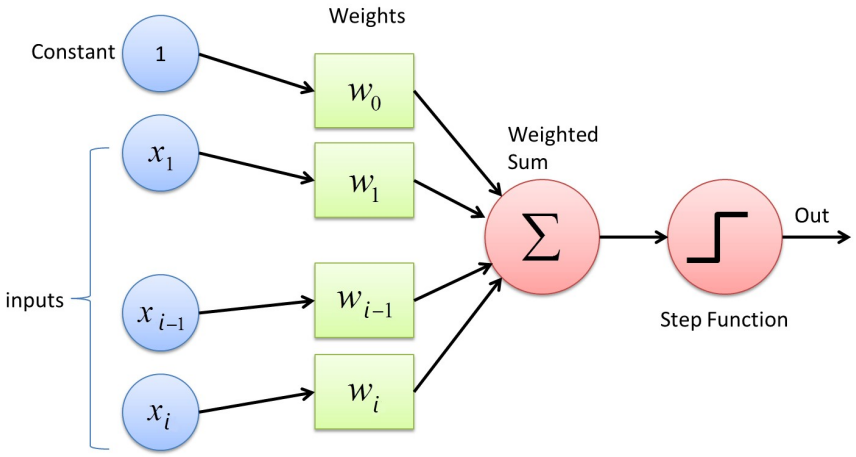

Figure 1: Schematic diagram of a perceptron.

The perceptron can also be represented as a function, which can be seen in equation 1. Analogous to the representation above, the inputs x_i are multiplied by the weights w_i in a linear combination. Then an error term is added so that the whole can be packed into the non-linear activation function $g(S)$. $\hat{y}$ is the binary output of this perceptron. With the aid of an activation function, binary output is obtained. The Heaviside step function shown in figure 1 is usually only used in single layer perceptrons, which recognize linear separable patterns. For the multi-layer neural networks presented later, step functions are not an option, because in the course of the backpropagation algorithm the gradient descent has to be minimized. This requires derivatives of the activation function, which in the case of this Heaviside step function equals 0. Because the foundation for the optimization process is missing, functions like the sigmoid function or the hyperbolic tangent function are used [2]. More about this topic is discussed in chapter 2.1.2.

$$\hat{y} = g(w_0 + \sum_{i=1}^{n} x_i w_i) \tag{1}$$

As just mentioned, the aim is to feed the perceptron with the training set and change the weights w_i with each cycle so that the prediction becomes more accurate. The output value is compared to the desired value. Finally, the sign of the difference $y - \hat{y}$ determines whether the inputs of that iteration are added to or subtracted from the weights. Ideally, the weights will gradually converge and provide us with a usable model [2].

2.1.2. Backpropagation algorithm

Finding the optimal weights of the neural network is achieved by finding the minimum of an error function. One of the most common methods for this is the backpropagation algorithm. This algorithm searches for the minimum of the error function by making use of a method called gradient descent. The gradient method is used in numerics to solve general optimization problems. In doing so, we progress (using the example of a minimization problem) from a starting point along a descent direction until no further numerical improvement is achieved. Since this method requires the computation of the gradient of the error function after each step, continuity and differentiability of this function must necessarily be given. The step function mentioned above in section 2.1.1. is therefore out of the question, but a non-linear function such as the logistic and the hyperbolic tangent functions (sigmoid) [3]. Both activation functions are visible in figure 2. While the target range of the 'ordinary' sigmoid function (equation 2) is between 0 and 1, the $\hat{y}$ of the hyperbolic tangent function (equation 3) ranges between -1 and 1. v_i equals the weighted sum including bias term.

$$\hat{y}(v_i) = (1 + e^{-v_i})^{-1} \tag{2}$$

$$\hat{y}(v_i) = \tanh(v_i) \tag{3}$$

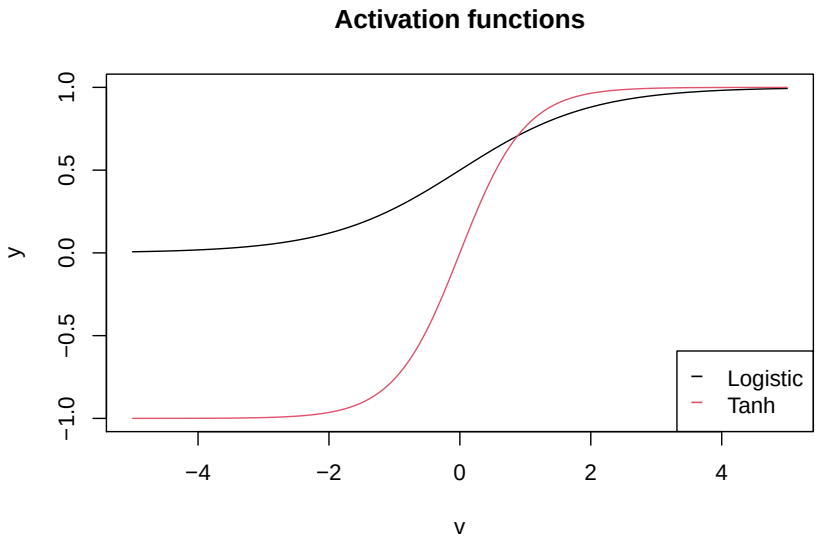

Figure 2: Two common sigmoid activation functions: logistic functions and hyperbolic tangent.

In the course of the error analysis, the output of the neural network respectively the result from the activation function in the output layer is compared with the desired value. The most commonly used error function E is the Mean Squared Error (MSE), which is seen in equation 4. y_i represents the actual value for the data point i, while $\hat{y}_i$ is the predicted value for data point i. The average of this error function is the average MSE, which is determined for a corresponding model. The learning problem is to adjust the weights w_i within the training sample so that $MSE(w)$ is minimized [4].

$$\begin{aligned} E &= MSE(w) \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - g(w_0 + x_i w_i))^2 \end{aligned} \tag{4}$$

As mentioned, this is searched for by the gradient descent method. The gradient of a function is a vector whose entries are the first partial derivatives of the function. The first entry is the partial derivative after the first variable, the second entry is the partial derivative after the second variable and so on. Each entry indicates the slope of the function in the direction of the variable to which it was derived. In this work, the notation ∇E is used when talking about the gradient for the error function E, which is displayed in equation 5 [3].

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_i}) \tag{5}$$

The weights get adjusted according to the following algorithm 6 where Δw_i is the change of the weight w_i and γ represents a freely definable parameter. In literature, this parameter is often called a learning constant [5]. The negative value is used because the gradient naturally points in the direction with the largest increase of the error function. To minimize the MSE, the elements in the gradient ∇E must be multiplied by -1.

$$\begin{aligned} \Delta w_i &= -\gamma \frac{\partial E}{\partial w_i}, \\ \text{for } i &= 1, 2, \ldots, n \end{aligned} \tag{6}$$

2.1.3. Multilayer perceptron

Multilayer perceptrons (MLP) are widely used feedforward neural network models and make usage of the backpropagation algorithm. They are an evolution of the original perceptron proposed by Rosenblatt in 1958 [1]. The distinction is that they have at least one hidden layer between input and output layer, which means that an MLP has more neurons whose weights must be optimized. Consequently, this requires more computing power, but more complex classification problems can be handled [6]. Figure 3 shows the structure of an MLP with n hidden layers. Compared to the perceptron, it can be seen that this neural network consists of an input layer, one or more hidden layers, and an output layer. In each layer, there is a different number of neurons, respectively nodes. These properties (number of layers and nodes) can be summarized with the term 'network architecture' and will be dealt with in this thesis.

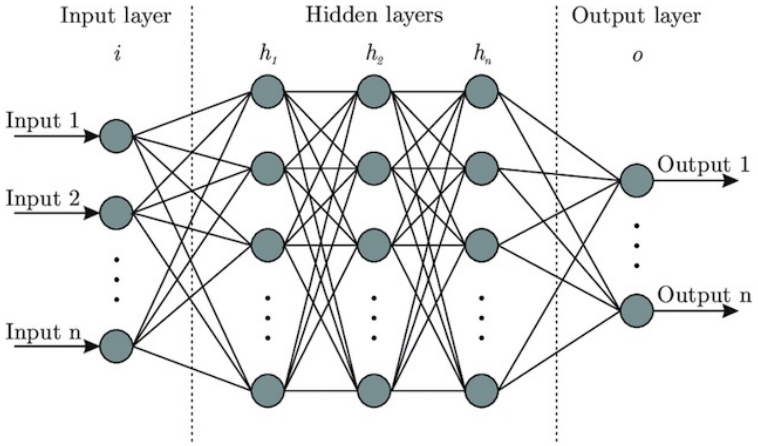

Figure 3: Schematic diagram of a multilayer perceptron

Every neural network has an input layer, which consists of one or more nodes. This number is determined from the training data and tells us how many features should be delivered to the neural network. In the case of bitcoin prices, we could use today's price and the prices of the last 10 days (lags 1-10), so the input layer would consist of 11 nodes. Some configurations also require a bias term to adjust the output along with the weighted sum, which is also added to the input layer. In contrast to the scheme of the MLP, this setup can be seen in figure 1 where the bias term is defined as 'constant.' Similarly to the input layer, each neural network has exactly one output layer. This can consist of one or more nodes. In this thesis, MLP is used as a regressor and therefore only one neuron is needed in this layer.

In between are the hidden layers, whose number and size can be configured as desired. The challenge is to find an optimal and efficient configuration without causing overfitting of the training data. The number of hidden layers depends primarily on the application area of the neural network. For example, working with image recognition would require more layers since the image file is broken down into individual pixels. Subsequently, the layers are used to optimize from rough outlines to the smallest detail. In our research, we came across several methods or 'rules of thumb' to optimize the model. A frequently suggested method is explained by Andrej Karpathy (director of the AI department of Tesla, Inc.). His GitHub entry recommends the approach of starting with a model that is too large that causes overfitting. Subsequently, the model is reduced by focusing on increasing training loss and improving validation loss [7].

2.1.4. Recurrent neural networks (RNN)

Recurrent neural networks (RNN) are a further development of conventional neural networks. While MLP use new inputs x_i in each epoch, RNN also use sequential data h_i in addition to x_i. This sequential data are called hidden states and result from the previous runs. This has the advantage that historical information stemming from past predictions is included for the prediction for $t+1$. This effect can be intuitively explained by an example in which the flight path of a scheduled flight is predicted using RNN. When predicting the

exact location (coordinates) of a plane, it is of great advantage to know the location at $t-1$ and to derive the flight direction from it. With the inclusion of this information, the target area can be narrowed down, which optimally leads to more accurate results. The same principle is used in applications like machine translation and speech recognition, where the result (here possibly letter or word) of the last epoch plays a big role for the next prediction [8].

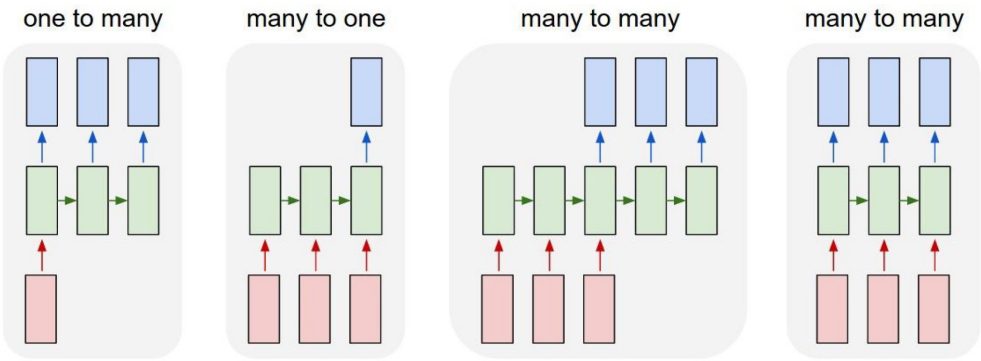

Figure 4: Process sequences of different applicances of RNN.

Figure 4 shows different process sequences of the RNN, which vary depending on the field of application. The red rectangles at the bottom represent the number of inputs. Similarly, the blue rectangles represent the outputs that come out of the RNN. The term 'many' refers to > 1 and is illustrated with three rectangles in the figure. The green ones represent the hidden states h_i of all time steps and thus can be seen as the memory of the neural network. The green arrows show that the previous hidden state is used as input for the current step. Starting from the left: one-to-many can be used for image captioning (extracting sequence of words from images), many-to-one for sentiment classification from sequence of words, many-to-many for machine translation (sequence of words in one language to sequence of words in another language) and many-to-many for video classification on frame level [9]. For the prediction of the BTC/USD exchange rate in this paper, we deal with the process many-to-one. This method combines information from inputs and hidden states into one single prediction value.

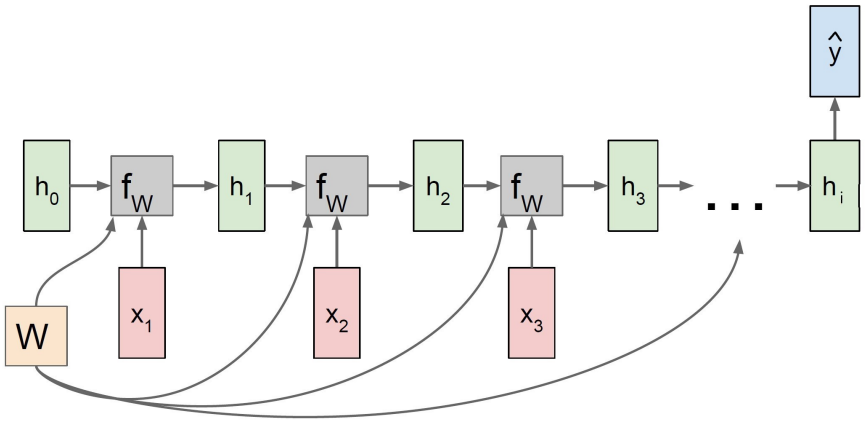

Figure 5: Computational graph of a many-to-one RNN.

$$\begin{aligned} h_i &= f_W(h_{i-1}, x_i) \\ &= \tanh(W_h h_{i-1} + W_x x_i + b) \end{aligned} \tag{7}$$

Equation 7 shows how the hidden states h_i are calculated at each time step, i where f_W is an activation function (here: hyperbolic tangent function), h_{i-1} is the previous state and x_i is the input vector at time step i. In some cases, a bias term b is added to the parameters. W_h represents the weight matrix for h_i with dimension (length(h)×length(h)). Thus, W_x is the weight matrix for x_i with dimension (length(h)×length(x)).

$$\hat{y}_i = W_y h_i \tag{8}$$

Looking at equation 8, y_i equals the output and desired prediction of the RNN. The prediction results from the matrix-vector product of the weight matrix W_y with dimension (length(h)×length(y)) and the hidden states vector h.

2.1.5. Long-short term memory (LSTM)

2.1.6. Challenges

2.1.6.1 Overfitting

We have encountered several challenges that can occur when using neural networks. One of these possible problems is called overfitting. The goal of a neural network is to build a statistical model of the training set that is capable of generating the data. In overfitting on the other hand, the exact conditions of the training data including noise are reproduced. The focus is no longer on the underlying function. Last but not least, an unnecessarily large number of parameters or epochs can be 'consumed' for this, which makes the whole process relatively inefficient [8].

=> still needs clarification how we solve these challenges in this thesis!

2.1.6.2. Vanishing gradient problem

Another characteristic that requires our attention is the vanishing gradient problem. As explained in chapter 2.1.2., the weights of the neural network are adjusted using the gradient of the loss function. Thereby, the problem can occur that the gradient almost vanishes. The error function's gradients become so small that the backpropagation algorithm takes smaller steps towards the loss function's minima and eventually stops learning. For example, if the derivative of an activation function such as the logistic sigmoid function approaches zero for extremely large or small values for x. To avoid these extreme values for x, the inputs are scaled and normalized in this paper. This ensures that the definition range is within the range where the gradient is still large enough for the backpropagation algorithm.

=> still needs clarification how we solve these challenges in this thesis!

2.2. Model comparison

This thesis sets the goal to compare the different neural networks presented. Besides the types of neural networks, the network architecture (number of layers and nodes) is explored. In addition, a comparison is made with the winner of the Forecasting Competition M4, which combines a standard exponential smoothing model with an LSTM network. To make the comparison meaningful enough, the following two figures are compared.

2.2.1. Sharpe Ratio

The first number refers to the performance of the trading strategy based on the sign of the prediction $t+1$ and is called Sharpe Ratio. Sharpe ratio is a very powerful and widely used ratio to measure performance and it describes return per risk.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma} \tag{9}$$

R_p represents the return of the portfolio, while R_f equals the risk free rate. σ is the standard deviation of the portfolios excess return (risk). For the comparison of different series, the Sharpe Ratio needs to be annualized with $\sqrt{365}$ as the crypto market is open 24/7.

2.2.2. Diebold Mariano

The second method used is the Diebold Mariano test, which compares the predictive accuracy between two forecasts. First, the loss differential d_i between two forecasts is defined in equation 11 where a loss function L of one model is subtracted from another model. The proposed loss functions include absolute errors (AE) and squared errors (SE) [10]. Given an expected value of $d = 0$, both forecasts are assumed to have the same accuracy. If the expected value differs from zero, the null hypothesis can be rejected. This would mean that the two methods have different levels of accuracy.

$$\begin{aligned} H_0 &: E(d_i) = 0 \\ H_1 &: E(d_i) \neq 0 \end{aligned} \tag{10}$$

with

$$d_i = L(e_{1i}) - L(e_{2i}) \tag{11}$$

and

$$\begin{aligned} e_{ti} &= \hat{y_{ti}} - y_i \\ &\text{for } t = 1, 2 \end{aligned} \tag{12}$$

Under the null hypothesis H_0, the Diebold Mariano test uses the statistics shown in equation 13 and is asymptotically N(0,1) distributed. On the other hand, the null hypothesis is rejected if the calculated absolute Diebold Mariano value is outside $-z_{\alpha/2}$ and $z_{\alpha/2}$. Thus, $|DM| > z_{\alpha/2}$ is valid when there is a significant difference between the predictions where $z_{\alpha/2}$ is the positive bound of the z-value to the level α.

$$DM = \frac{\bar{d}}{\sqrt{\frac{2*\pi*\hat{f_d(0)}}{T}}} \rightarrow N(0,1) \tag{13}$$

where $\bar{d}$ is the sample mean of the loss differential and $f_d(0)$ is the spectral density of the loss differential at lag k [11].

$$\bar{d} = \sum_{i=1}^{T} d_i \tag{14}$$

$$f_d(0) = \frac{1}{2\pi}\left(\sum_{k=-\infty}^{\infty} \gamma_d(k)\right) \tag{15}$$

In conclusion, the Diebold Mariano test helps us to understand whether the predictions of one model turned out better by chance or due to statistical significance.

2.2.3. Mean Squared Error (MSE)

The third performance measurement method is also widely used and called mean squared error. Its calculation is very simple, for every timestep the estimated value is subtracted from the real empirical value, squared and then summarized and divided by the absolute number of observations as seen in equation 16.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(realvalue_i - prediciton_i)^2 \tag{16}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{17}$$

The application of MSE is widely used in statistical modeling and supervised learning. However, it should be noted that the metric is very sensitive to outliers. In the presence of outliers, a robust variant should be considered as an alternative.

2.3. Bitcoin

In this section bitcoin as a crypto-curreny is introduced. The historical data is analyzed and commented. Further the technology in and around crypto-currencies in chapter 2.3.2. is briefly explained. A detailed explanation would require a paper itself, therefore the explanation is done as simple as possible.

In the following work bitcoin as a cryptocurreny is mentionend in its short term BTC, by the meaning of US Dollars per bitcoin.

2.3.1. Historical analysis

The story of bitcoin began with a paper published by the name of Satoshi Nakamoto [12]. The publisher of the document cannot be assigned to a real person, therefore the technology inventor remains mysteriosly unknown until today. In 2009 the first bitcoin transaction was executed. On account of the opensource technology of bitcoin, lots of alternative currencies were created.

Until 2013 the cryptocurrencies operated under the radar of most regulatory institutions. Because of the anonymity of the transactions, criminals were attracted by the newborn payment method. Headlines, such as the seizure of 26'000 bitcoins by closing the "Dark-Web" Website Silkroad through the Drug Enforcement Agency, followed moreoften in the newspapers.

Nevertheless in 2014 more companys, such as: Zynga, D LasVegas Casinos, Golden Gate Hotel & Casino, TigerDirect, Overstock.com, Newegg, Dell, and even Microsoft [13], began to accept bitcoin as a payment method. In 2014 the first derivative with bitcoin as an underlying was approved by the U.S.Commodity Futures Trading Commission. 2015 an estimated 160'000 merchants used bitcoin to trade.

Let us first look a the price in Figure 7 and the log(price) in Figure 6 and get a sense of the chart. Note: The data in the charts start in 2014 where it was listed in coinmarket, events between 2009 and 2014 are described without visualization.

Around 2010 bitcoin had the first increase in price as it jumped a 100% from 0.0008 USD to 0.08 Dollar [14]. In 2011 the price rose from 1 USD to 32 USD within 3 months and recessd shortly after to 2 USD this can be referred as a first price bubble in bitcoin, for the next year the price climbed to 13 Dollars and reached a never seen level of 220 USD, only to plunge to 70 USD within a half month in April 2013. By the end of the year a rally brought btc up to a peak of 1156 USD. The following year brought bad news and the price slowly decreased to 315 USD in 2015 after an observed drop of 20% after news from the trial of Ross Ulbricht, founder of Silk road marked in Letter **A**.

From this point in time, things began to change, more volume was flushed in the market and the price of BTC began to ascend and the real rally began ,the BTC rose up to 20k USD / BTC on 17th September 2017 **B**. After the rise comes the fall and BTC lost value for more than a year until **C** 2018-12-15 the trend reverted and found its peak after 6 months in **D** 2019-06-26, but oncemore it was not lasting for long as bitcoin lost **D** 2020-03-12 nearly half its value in 4 days.

But the story was not over by now, after the drop the price of the cryptocurrency regained value, passed previous levels and shortly after exploded, after companies like Tesla and Signal bought a big chunk of bitcoins, into a maximum of 58000 USD per bitcoin. It is also observed that the value of bitcoin is very volatile, we will discuss this in section 3.2..

Figure 6: Logarithmic BTC/USD

Figure 7: BTC/USD

2.3.2. Bitcoin technology and cryptocurrencies

This chapter focuses on the technical aspects of the cryptocurrency bitcoin. It describes the role that blockchain technology plays in cryptocurrencies and how this manifests itself in the case of bitcoin. Cryptocurrency as a more general term is used because bitcoin was only the first of its kind. One may look at a cryptocurrency similarly to a normal currency because you can buy and sell things and get bitcoin in exchange. But cryptocurrencies fundamentally differ to conservative currencies in merely all ways. The cryptocurrencies (not just bitcoin) are based on the blockchain technology introduced in Nakamotos paper [12]. The system is decentralized, where no institution or government regulates the market in terms of the blockchain itself. The transactions are signed by the participants via cryptographic hash functions, which generate a private and public key. This means that every signature can only be accessed by the owner of the private key i.e. it can not be copied. Once a transaction is signed, it is broadcasted into the network to all participants, so that everyone sees a transaction has been made. Around 2400 transactions are packed in a block (the blocksize is limited by memory) which are broadcasted to all participants of the system. Every block consists the transaction information, previous hash, the special number and their resulting hash as visualized in figure 8. Miners are now trying to approve the block by generating a hash with a certain pattern with the hashfunction f(prevhash, data of the block, special number), the so called proof of work. The first miner who finds the according special number to the hash with the certain pattern, gets an amount of bitcoin in reward. The block with its new hash and the special number are now added to the chain and are broadcasted to the network. If someone manipulates transactions in a block and finds the special number to the hash, he could potentially get away with it but not for long because for the next block he must be also the first to find the right hash and so on. In figure 8, the red block is a false one which gets attached and later declined because the other branch is longer. Only the longest chain is to be trusted, and because there are so many miners one must have more than 50 % off the calculating power to get the best chance to find the right hash. Therefore its almost impossible to manipulate the chain. The cryptocurrency itself is now entirely defined by a chain of approved blocks by all participants.

Another interesting fact about bitcoin is that the amount of coins is determined by the rewards of the miners. The first block (genesisblock) had a reward of 50 Bitcoins, every 210'000 blocks this reward gets halved. Since a new block is added every 10 minutes (this is the average time to solve a hash) the halving of the block rewards occurs approximately every 4 years. Under these conditions, one expects a block reward of zero in 2140 with a maximum number of Bitcoins of 21 million [15].

In recent days, the cryptocurrency has come under a lot of criticism. The immense computing expenditure has a very high power consumption which leads cryptomining companies to build huge farms with massive cooling aggregates. According to the article in Forbes magazine [16] the bitcoin mining process uses 0.51 percent (127TWh) of global energy consumption. The University of Cambridge created an index where the live energy consumption can be observed [17]. Right now China [18] contributes 70 % to the hash rate whereas the remaining 30 percent are distributed over the rest of the world.

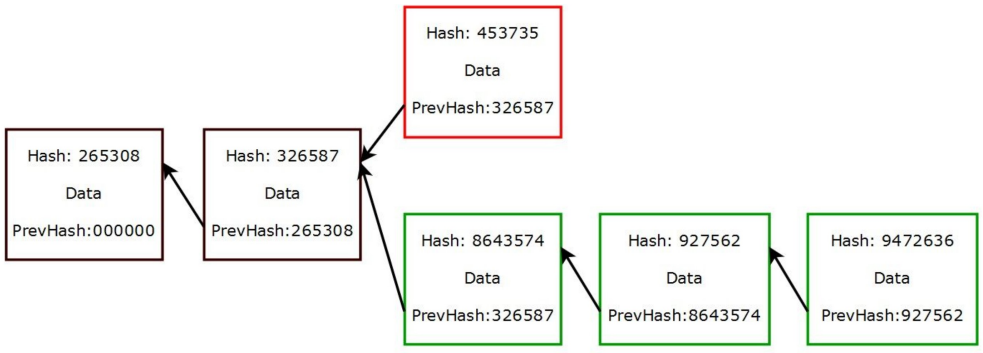

Figure 8: Blockchain schema

2.3.3. Valuation and digital gold

As mentioned in the previous chapter, the maximum possible amount of 21 million equals a fixed supply of this cryptocurrency. This leads to the question if this has an influence on the fair market value of bitcoins. A. Meynkhard is conducting research in this area and has concluded that it relies on the following three factors [19]:

- Fixed maximum supply
- Mining process
- Reward halving

First, it is emphasized that in a decentralized monetary system, the newly issued amount is defined by the cryptographic algorithm. Consistently, new Bitcoins enter circulation when a miner sells their received reward to fund operating costs or new equipment. Unlike a central bank, which typically aims for an annual inflation target of 2%, with bitcoin the number of new coins issued is decreased after each halving. Meynkhard describes that this decrease in newly issued Bitcoins, assuming constant demand, causes the market value to increase in the long-term. Although in contrast to stock markets, there is only a fraction of historical data available, this halving phenomenon could certainly be observed. The halving in 2016 is made responsible for the price increase from USD 500 to USD 20'000 by December 2017. The latest halving in May 2020 appears to be responsible for the present bull market, which let the price drive from USD 9000 to over USD 60'000. This deflationary characteristic of bitcoin led to it being more and more referred to as digital gold in the broad media [20].

2.4. Explainable artificial intelligence

Depending on the model architecture, a neural network can be a very complex construct. A number of weights and biases linked to the neurons lead to an output of the network through training. Understanding how exactly the alterations of the weights and biases lead to this output is a rather complex task. Due to this difficulty in interpretation, neural networks are often referred to as black boxes [21].

Although the networks may lead to desired results, it can be important to build an understanding of these models. Suppose we are developing a classification method in supervised learning for a particular problem. A classical approach such as linear regression is easy to understand and we can convince people with little knowledge of mathematics of the usefulness of this method. Consider that a good and simple explanation may depend on investment. Would an investor invest in something that is not understood and difficult to get a grasp of? Neural networks with their non-linearity and the large number of parameters does not make it easy for the user to convince an investor of the benefits of a neural network.

2.4.1. Classic Approach

As mentioned earlier, it is almost impossible to explain the networks based on the weights and biases. The following methods try to find effects on the features. One tries to find out which influence a certain feature has on the prediction of the network. There are classical approaches for explaining neural networks in the applications such as image recognition or text mining.

A widely used approach is the Shapley value which has its origin in game theory. With this method, one tries to find out how big the influence of a feature is. So how much a feature contributes to the prediction. The problem with this method is that it mixes the data at different points in time. In this paper, we study the prediction of financial time series, i.e. autocorrelated data. Thus, this approach is not suitable for our application to interpret the importance of individual features (lagged log returns) [22].

An approach like ALE (Accumulated Local Effects) examines how the network reacts to change [22]. The features (lags) are plotted in order against ALE in a graph. The problem with this method is that the plotted features do not contain the dimension of time. It could be that two very high values are very close to each other in the plot, but in reality, they occurred years after each other.

Another approach is the LIME (Local Interpretable Model-agnostic Explanations). This method examines the change of the forecast when changing the input data. A permuted data set is generated from the given data (for example, adding standardized noise). Using this artificially altered data set, an interpretable model (regression) is created to analyze the change in features [23]. Changing the data affects the time dependence of the data. Thus, this method is also unusable for our application.

2.4.2. Explainability for Financial Time Series

For the interpretation of financial time series, we would like to keep the dependency structure. The changes in trend or variability should be maintained. The sequence should not be shuffled. When mixing up the order, a value from far back could suddenly play a bigger role for the model than a current one. Intuitively, this would not add any value to financial time series.

For our application, the lagged log returns of bitcoin prices, are the features to be studied. We want to find out how the features affect the output of the neural network. A network is trained with delayed values as the input layer to match the output as closely as possible to the original values (the non-delayed ones). This concept strongly resembles a linear regression. What would this look like.

$$Y_i = \beta_0 + \beta_1 * x_i^{(1)} + \beta_2 * x_i^{(2)} + ... + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{18}$$

In our concrete example, the equation would look like this.

$$\text{Original Data}_t = \text{Intercept} + \beta_1 * \text{Data}_t^{(lag=1)} + \beta_2 * \text{Data}_t^{(lag=2)} + ... + \epsilon_i \tag{19}$$

Now, what could the fitted regression, respectively the regression coefficients tell us about the respective lagged values. For that, we can look at the autocorrelation function of the bitcoin log returns in figure 9. Lags at 6 and 10 have a positive impact on the original data structure. In the context of the time series, this means that there is a strong 6- or 10-day dependency.

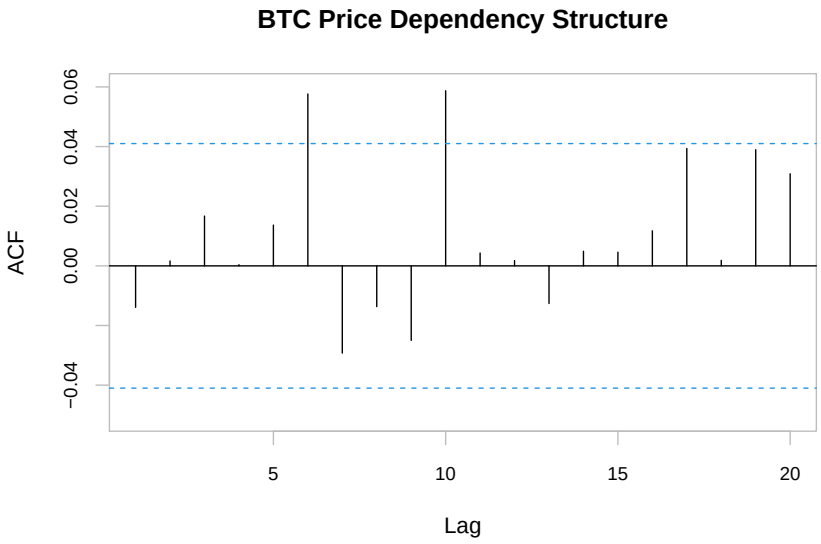

Figure 9: Autocorrelation function of BTC/USD.

Looking at the regression coefficients in table 1, we can discover the relationship between the ACF's and the coefficients of the regression. Again, lags 6 and 10 make the largest positive contribution to the fit of the model. Lags 7, 8, 9 make a negative contribution, as can also be seen in the ACFs. Sign and value are in line with the ACF's.

Table 1: Coefficient of the linear regression of the BTC log returns.

Intercept	Lag 1	Lag 2	Lag 3	Lag 4	Lag 5	Lag 6	Lag 7	Lag 8	Lag 9	Lag 10
0.0019	-0.0131	0.0038	0.0186	6e-04	0.0152	0.0569	-0.03	-0.0166	-0.0268	0.059

Now we would like to create such an analogy with neural networks. In the case of linear regression, the coefficients, or the weights of the lagged data are the solutions of the partial derivatives of the optimization function. One obtains a coefficient for each lag, thus the respective weight, is time independent. We would now like to extend this concept. We want to keep the structure of the time series, the time dependence.

We can now calculate the partial derivatives of the output of a neural network with the respective input data of time t for each time t. We obtain the coefficients or weights β_{it}.

$$\beta_{it} = \frac{\partial \text{Output}_t}{\partial \text{Data}_t^{(lag=i)}} \tag{20}$$

Basically, we can thus state the following relationship between output and weighted input.

$$\text{Output}_t = \text{Intercept} + \beta_{1t} * \text{Data}_t^{(lag=1)} + \beta_{2t} * \text{Data}_t^{(lag=2)} + ... + \beta_{qt} * \text{Data}_t^{(lag=q)} \tag{21}$$

In simpler terms, we train a neural network with input data of length up to delay q. Now we change a data point at time t, we add a disturbance term by δ. So we change the value of an explanatory variable, one of our features. Suppose our input data at time t looks like this.

$$X = x_{t-1} + x_{t-2} + ... + x_{t-q}$$

For the feature at lag 1, we now want to calculate the partial derivative. We create a new data set Y and change the data point at lag 1.

$$Y = x_{t-1} + (\delta * x_{t-1}) + x_{t-2} + ... + x_{t-q}$$

With the trained network we now generate two predictions for time t. Once with the original data $NN(X)$ and once with the data with the slightly altered value $NN(Y)$. Now we can calculate the discrete approximated partial derivative.

$$\beta_{1t} = \frac{\text{NN(X) - NN(Y)}}{\delta * x_{t-1}}$$

The output of the neural network changes by the value β_{1t} if the input is changed by 1. This procedure can now be done for each feature at each point in time. Then one has for each feature at each time the derivations, which possibly makes an explanation of neuronal networks possible.

asdasdasd

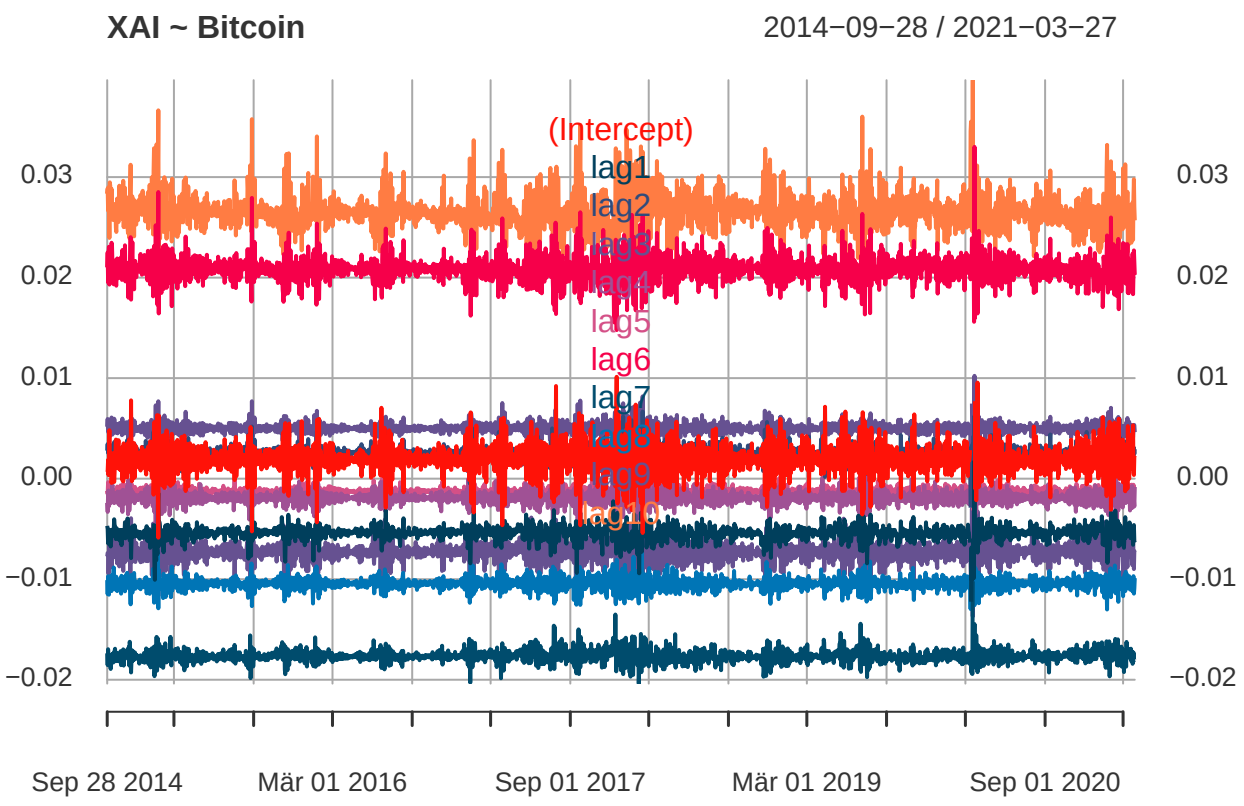

Figure 10: XAI.

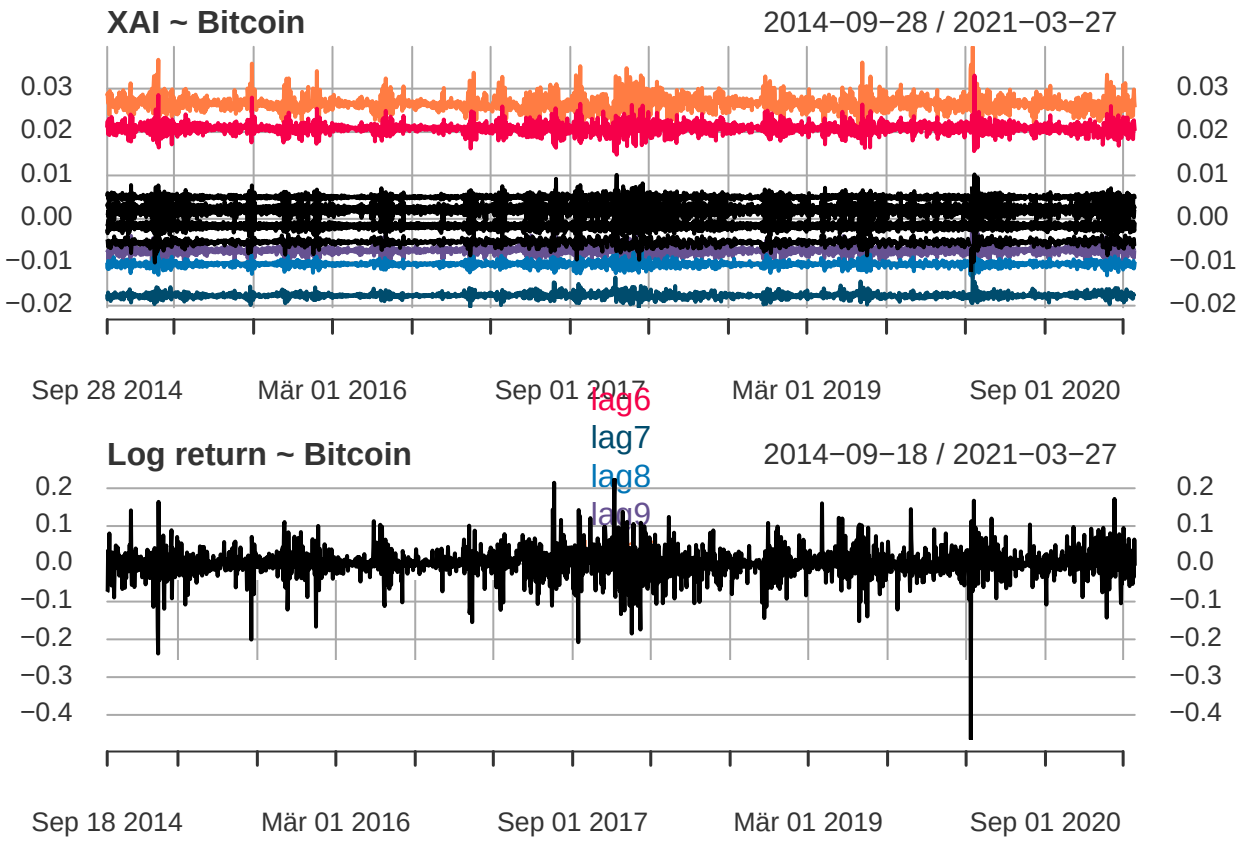

Figure 11: XAI compared to log returns of BTC.

```
##         lag6         lag7         lag8         lag9        lag10
##  0.020840265 -0.017641435 -0.010419764 -0.007275959  0.026554337
```

```
##        lag6        lag7        lag8        lag9       lag10
##  0.05687588 -0.03004978 -0.01664858 -0.02683498  0.05899144
```

3. Methodology

The focus of this thesis can be divided into two areas. First, the aim is to find an optimal neural network including a network architecture. This should perform well in the application area, in which the future log return of the bitcoin is predicted on the basis of historical log returns. In a second step, we will focus on defining a trading strategy based on our findings. All considerations and findings will be presented in a quantitative way and compared with each other. Figure 12 helps to get an overview of the individual steps followed in this chapter.

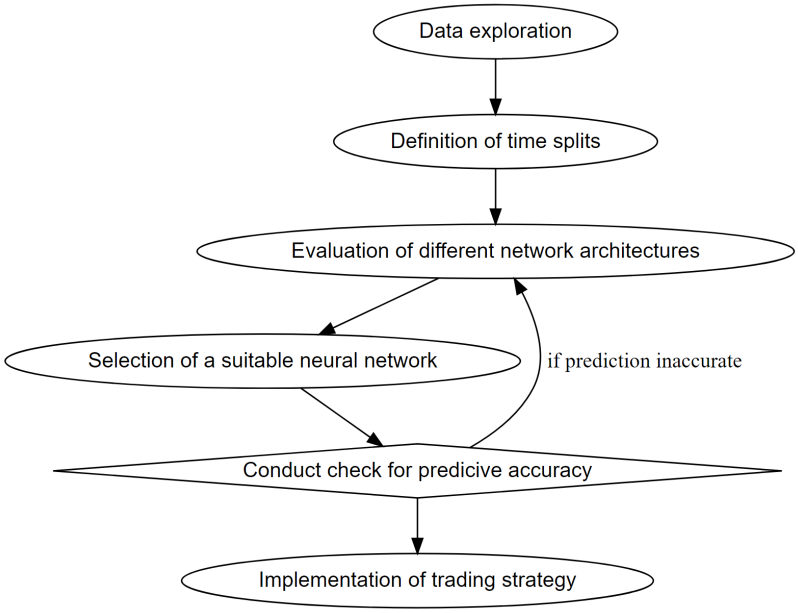

Figure 12: This flowchart illustrates an overview of the individual intermediate steps that are covered in the Methodology chapter.

3.1. Data exploration

The data in this paper is accessed through the API of Yahoo Finance and is originally provided by the price-tracking website CoinMarketCap. We use the daily closing price of bitcoin in USD with the ticker BTC-USD. As cryptocurrencies are traded 24/7, the closing price refers to the last price of the day evaluated at the last time stamp according to the Coordinated Universal Time (UTC).

In chapter 2.3., the bitcoin price and the logarithmic price is visualized. For processing and analyzing the data in order to fulfill the weak stationarity assumptions, we transform the data into log returns according to equation 22.

$$\text{LogReturn} = \log(x_t) - \log(x_{t-1}) \tag{22}$$

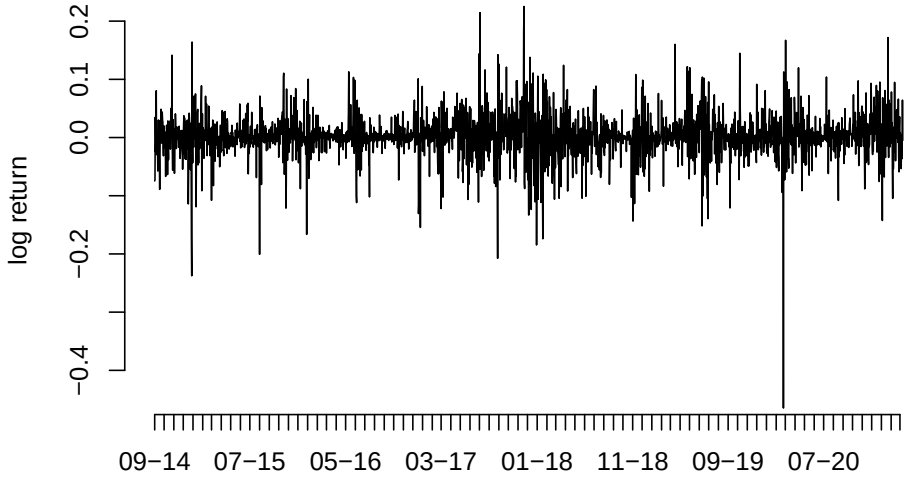

Figure 13: BTC log returns

Figure 13 displays the historical log returns. In addition to the volatility clusters typical for financial time series, large outliers are visible. The negative outlier at the beginning of 2020 is particularly noticeable. By computing the autocorrelation (ACF) of the series in figure 14, we can describe the dependency in these clusters. According to the ACF the lags 6 and 10 are significant on a 5% level.

Curios from which distribution the log returns might originate, we are fitting a normal distribution and a Students-t distribution to the data in figure 15. Interestingly the mean is shifted slightly (0.002) to the positive side. By inspecting the tails, one can observe that the negative tail is not fitted as good as the positive part by the t distribution. The two normal distributions either over- or underestimate the values in the tails, therefore we conclude that the proposed t-distribution fits the data better but also not perfect. Concerning the extreme outlier discussed earlier, visible in figure 13 towards the end, the density plot makes clear how unimaginable small the probability of this extreme observation is. Altough the histogram is might useful for value-at-risk considerations, for trading purposes its use is mitigated due to its complete loss of the dependency structure by plotting the returns in a density-distribution.

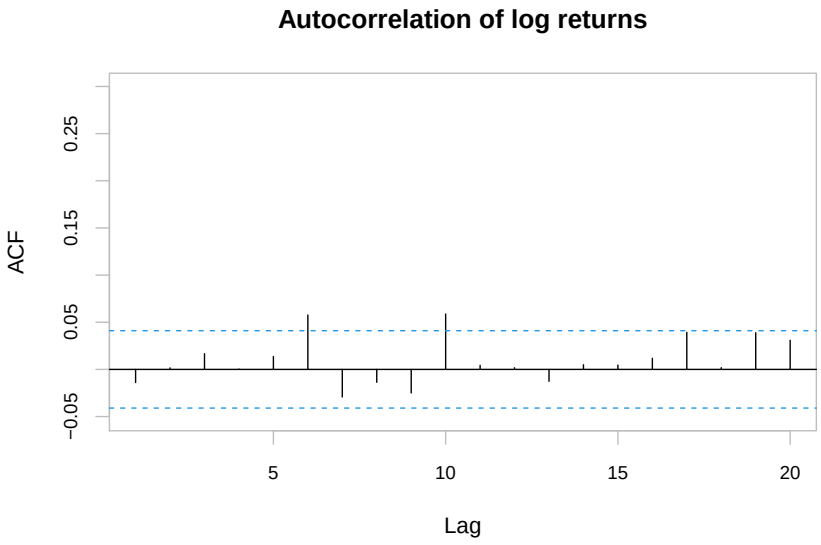

Figure 14: Autocorrelation of BTC log returns for the entire time window

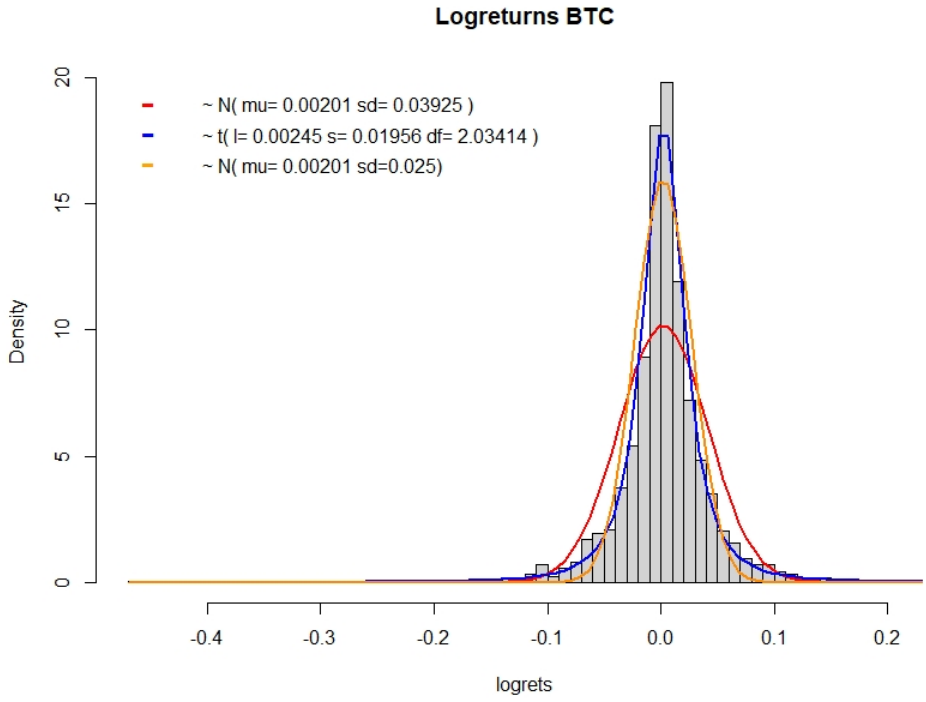

Figure 15: Distribution of BTC log returns

3.2. Network architecture

As mentioned in chapter 2.1.3., choosing an appropriate network architecture for bitcoin price prediction is a crucial step in order to achieve useful forecasts while avoiding overfitting. Due to the complexity as well as the non-linearity of neural networks, the interpretation cannot be performed intuitively. For this reason, an approach is pursued in which neural networks with different numbers of layers and neurons are compared with each other by using the MSE loss and Sharpe ratio. This allows us to compare accuracy, respectively trading performance and possibly see a connection with network architecture.

To find the optimal network architecture, we test a maximum of 3 layers with a maximum of 10 neurons each. More complex models are not included in this thesis, as this would exceed the time frame. Furthermore, the application of complex network architectures for financial time series can be expected to lead to overfitting and thus to no real added value. The simplest network has one layer with one neuron (1), while the most complex has 3 layers with 10 neurons each (10,10,10). The total number of different combinations can be expressed as follows:

$$\text{comb} = \sum_{i=1}^{L} N^i \tag{23}$$

with:

L = maximum Layer $\in \mathbb{N}^*$

N = maximum Neuron $\in \mathbb{N}^*$

comb = Number of all combinations

Thus, with our initial setup, we obtain a maximum neuron-layer combination of 1110. To respond to the challenges mentioned in section 2.1.6., not only a single network per neuron-layer combination is trained, but a whole batch of 50 networks. So we end up with a total of 55'500 trained networks. For each individual network, the in-sample and out-of-sample MSE as well as the Sharpe ratios are determined. We use these values to find an optimal network architecture based on the statistical error as well as on the trading performance (daily trading).

Finally, to ensure that the network architecture does not perform only for the selected time frame, the in-sample and out-of-sample split as well as the time period is discussed.

3.2.1. Defining train and test samples

We are looking for an optimal network, the optimal network should also provide reasonable and reliable predictions for different periods. For further analysis, we use a subset of the introduced closing prices of the bitcoin. Starting from the first of January 2020 to the 27th of March in 2021, we only consider 15 months for our data.

The reason for doing so is, we do not believe that the historical data longer than a year consists any information about the price tomorrow. By optimizing our models we found that more data would bring no additional performance, therefore the selected subset should be sufficient. Regarding consistency, the terms train and test set are used in the same sense as in-sample and out-of-sample. As proposed in [24] we choose a test train split from 6 months in-sample and 1 month out-of-sample. This split is applied to the whole subset in form of a rolling window. By stepping forward with this 6/1 split by steplength of one month we end up with 9 data splits in total. In figure 16 this procedure is visualized, for every new timestep a new month is considered for the out of sample and the first month of the in sample, falls out of the frame.

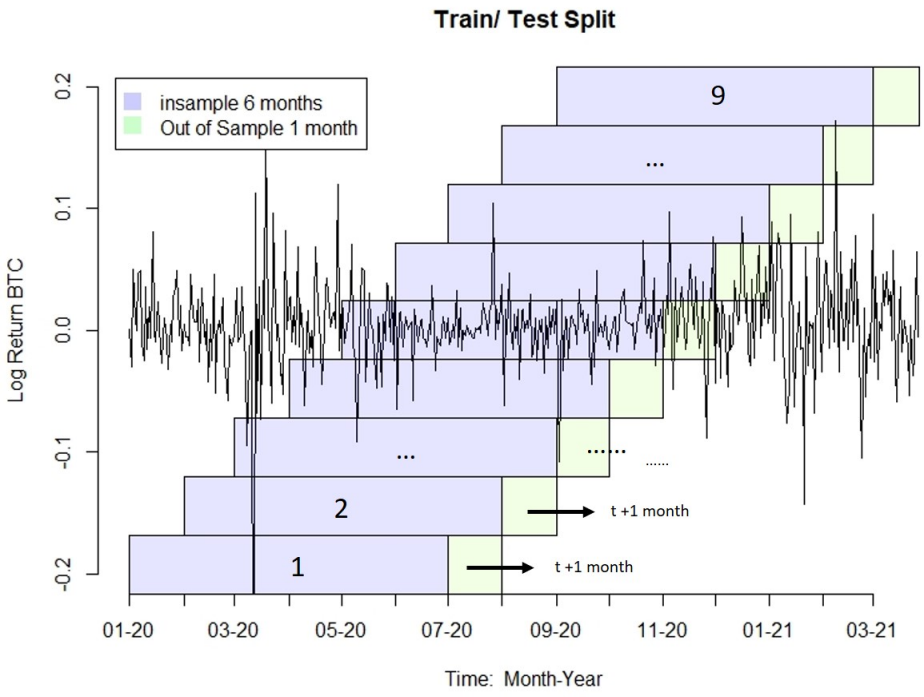

Figure 16: Distribution of BTC log returns

In the time series in figure 16, one can see different periods. Strongly volatile as well as rather calm phases occur. With the rolling window, we can train and test the networks based on different phases. Thus, we can also evaluate the performance of the networks based on different phases and not only on a predefined single test and train split.

The complexity of the search for the optimal network architecture increases significantly here. With the conditions defined for us, we train and test a total number of 499'500 networks to define the optimal network.

3.2.2. Evaluating network architecture

Here we would like to focus on some findings that we discovered during the processing of the trained networks. To illustrate the results, an extract is discussed here, namely only the 5th train/test split (in figure 16 the middle one).

The plot in figure 17 compares one layer networks with different numbers of neurons with each other. Networks with a maximum of ten neurons are compared. These different configurations can be seen on the x-axis. The first data point corresponds to a simple network with one neuron. The y-axis shows the MSE values obtained with the respective trained model. As already explained, we use 50 different optimizations of each configuration to get a better idea of a potentially systematic relationship with the MSE. In the plot, each of the 50 configurations is drawn using a different color.

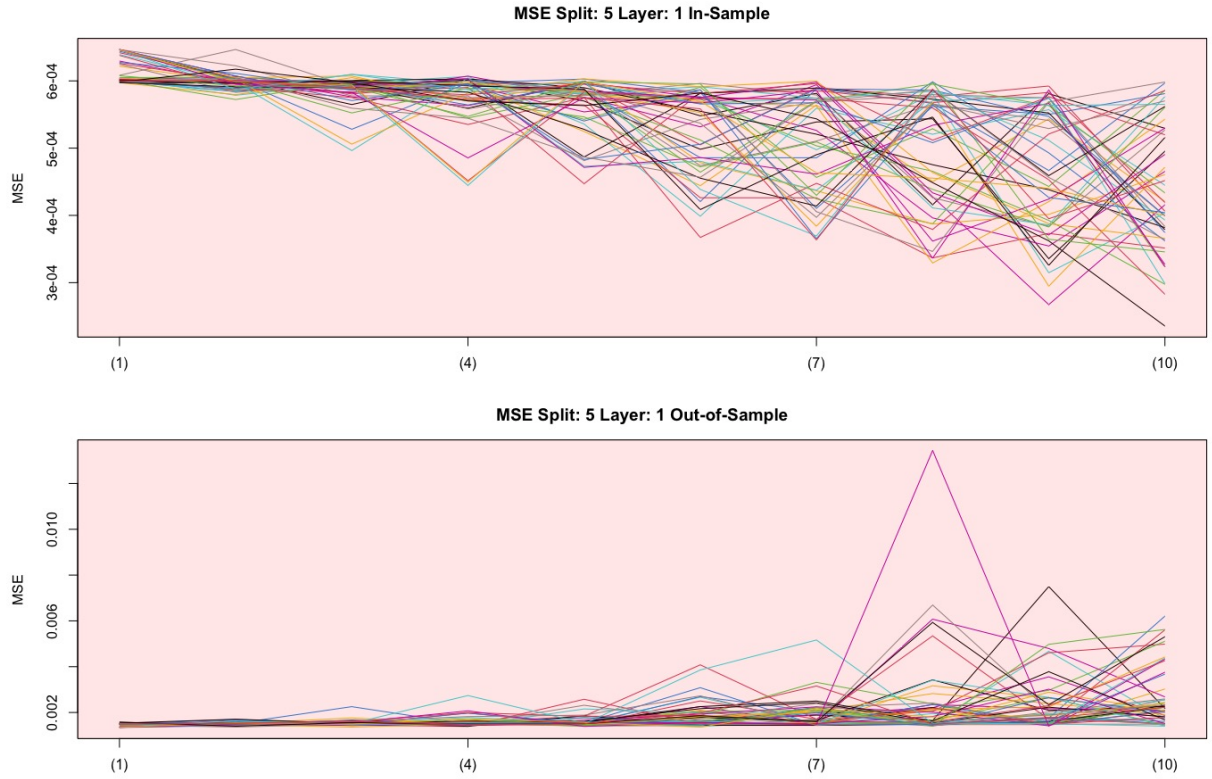

Figure 17: Fifth train/test split, 1 layer with 10 different networks.

What is already noticeable here is that with increasing complexity, i.e. with increasing number of neurons, the in-sample MSE decreases. The in-sample forecasts are thus becoming more accurate. At the same time, you can see how the out-of-sample MSE increases with increasing complexity, which means that the forecast accuracy tends to get worse.

If you add another layer to the network architecture, the number of different networks with the same number of layers also increases. In the following figure 18, the simplest network is a (1,1) network. So 2 layers with one neuron each. The most complex is a network with a (10,10) architecture.

As noted earlier in figure 17, the values for the MSE also fluctuate more and more with increasing complexity. Small in-sample MSE for more complex networks lead to rather high out-of-sample MSE. This leads us to the previously mentioned challenges in section 2.1.6.1., and that is that too many estimated parameters can lead to overfitting of the network.

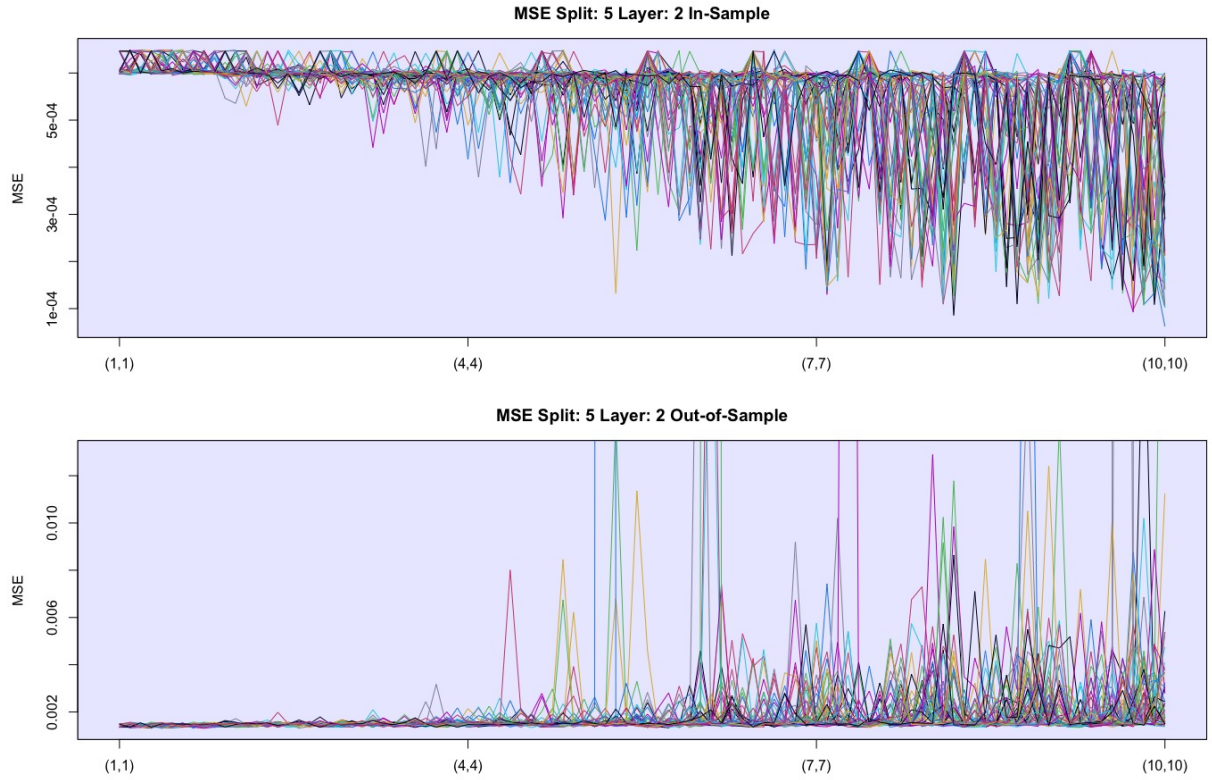

Figure 18: Fifth train/test split, 2 layers with 100 different networks.

Looking at the out-of-sample MSE's in the graph below in figure 18, we can see lines that are outside of the blue rectangle. These values are extreme outliers that indicate the randomness of neural networks. This again confirms that choosing an optimal network over several equal networks (50 in our case) makes more sense than making the choice depend on only one randomly trained network. Depending on which solution the training algorithm finds, the results can be very different. The y-axis was scaled for better comparability of in-sample and out-of-sample, but one loses the overview of how much the outliers differ from the rest.

Lastly, we look at the results of the different network architectures with a third layer. In figure 19, we can see very well the inverse correlation between the in-sample and out-of-sample MSE. Again, the in-sample MSE gets better with increasing complexity while the out-of-sample MSE gets worse. There is also a certain recurring pattern that is striking. After a certain complexity, the in-sample MSE decreases steadily and then increases abruptly. The opposite pattern can also be observed out-of-sample. These patterns emerge during transitions from more complex to more simple architectures. For example, the transition from a model with (8,10,10), with a total of 28 neurons, to a model with (9,1,1) with only 11 neurons.

It is interesting that at the beginning, with the rather simple model architectures, the MSE of all realizations is very constant and only varies very slightly.

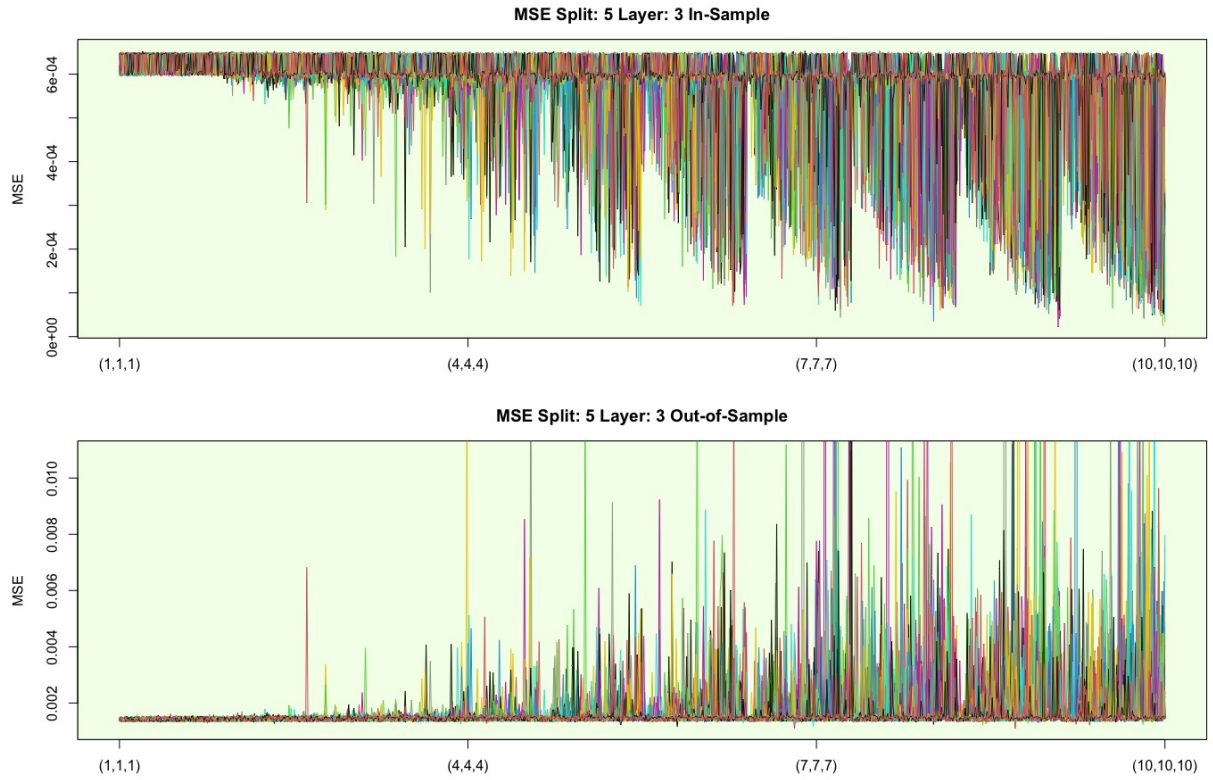

Figure 19: Fifth train/test split, 3 layers with 1000 different networks.

Figure 20 shows the MSE of the models with 1-3 layers, i.e. the last three plots side by side. As mentioned, it can be seen here that the in-sample MSE scatter towards the bottom respectively decreases with more complex architectures. This does not have a positive effect on the out-of-sample, since in the same area the MSE deteriorates massively (note the different scaling of the y-axes). As a result, we have no real added value from more complex models. Also to be noted is that the in-sample MSE does not get worse than a certain threshold at the upper boundary. This asymmetric scattering around this value is likely due to numerical characteristics of the optimization algorithm.

At this point it should be emphasized that only the analyses from time split 5 are visualized in this chapter. Our primary goal is to compare the performances of different network architectures using MSE in order to find the optimal network that works in every time split. However, finding an optimal architecture using such visual analysis of the MSE seems nearly impossible. Nevertheless, the main finding of this section is that the MSE deteriorates massively with more complex models and thus a simpler one should be considered. Equally remarkable is the fact that the same model architectures produce such different results. Whilst many models range in a more or less solid midfield, traits of overfitting can be recognized. These are reflected by the spikes in the out-of-sample MSE.

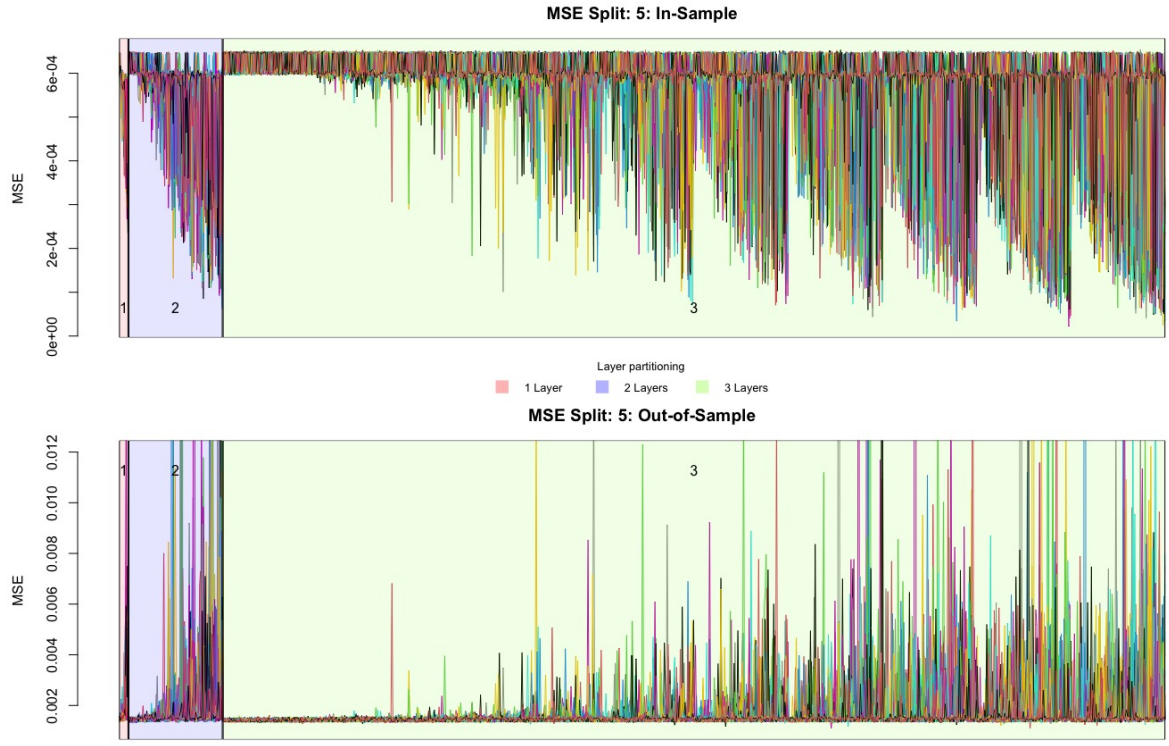

Figure 20: Fifth train/test split, all layers with 1110 different networks.

What has not yet been studied is the dependence of the neural network's behavior on the different window splits we defined in figure 16. Considering that the same network architectures provide MSE's of different quality (including huge outliers), the results for each configuration are summarized using a robust method. We make use of the median of the MSE's of all 50 equal networks over all time splits in order to evaluate the accuracy of the corresponding model. We consider this a better method than the arithmetic mean as figure 20 shows large outliers. Thus, it can be better investigated whether the corresponding network architecture provides sound results apart from this one outlier.

The medians of the MSE's of all 50 equal networks over all time splits are plotted in figure 21. We restrict ourselves to neural networks with 1-2 layers (recognizable in the red (1) and blue (2) rectangles), since it can be assumed that too complex models are not suitable for the target. The lines represent the medians of the MSE of all 50 optimized neural networks at a given network architecture (x-axis). The nine different colors specify the specific time split in which the neural networks have been trained and tested. We anticipate that this comparison will facilitate finding a network architecture that performs over all time splits.

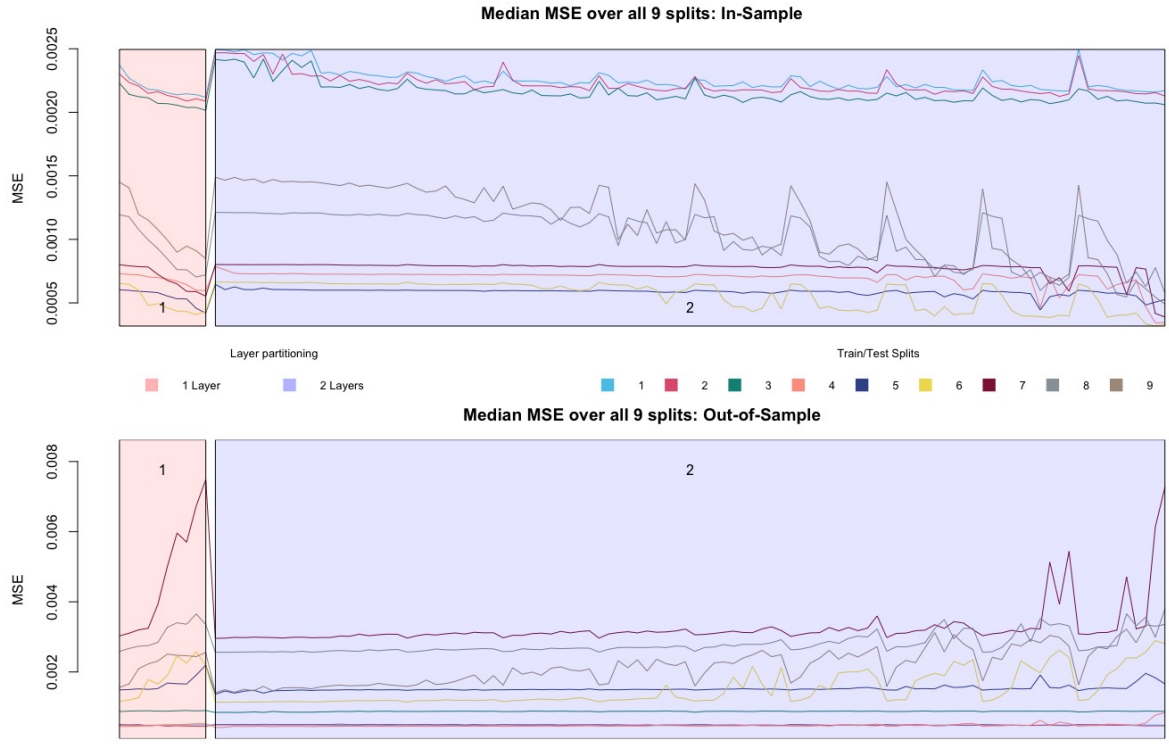

Figure 21: MSE median over all 9 splits. For each network architecture, 50 neural networks were trained (55'500 models per time split). The nine colors illustrate how these models behave in the different time splits.

First, there is evidence of overfitting in the medians as well. In particular, splits 1-6 show a somewhat expected picture: the in-sample error becomes smaller with more complex architecture, while the opposite can be seen in the out-of-sample. However, the periodically appearing spikes visible in the time splits 8 and 9 of the in-sample plot seem unnatural. The plausible difference between this and the other splits is the underlying data. Therefore, we take a look at what the initial prices of bitcoin are doing in this period. The plots with the time splits of the logarithmized prices can be found in the appendix starting with figure 30. As a rule of thumb, the neural network behaves better when the train and test pairs behave similarly. In the plots of splits 8 and 9 it is clearly visible that in each case the in-sample shows a bullish behavior. This trend is not continued in the out-of-sample part, which probably leads to biased predictions due to amplified dependence structures.

Our goal in the second part of this thesis is to work out a trading strategy with a suitable neural network. Therefore, as a last comparison, we visualize in figure 22 how well the different network architectures behave in sign trading. This is a simple trading which depends on the sign of the prediction $\hat{y}_{t+1}$ i.e. next expected log return. If a positive prediction is forecasted, the trader is in a long position, otherwise in a short position. As with the previous plot, the Sharpe ratios of each neural network realization perform differently despite having the same network architecture. Therefore, we again decided to plot the median of all Sharpe ratios with the same network architecture. The nine different colors indicate the time interval during which the neural networks were trained and tested.

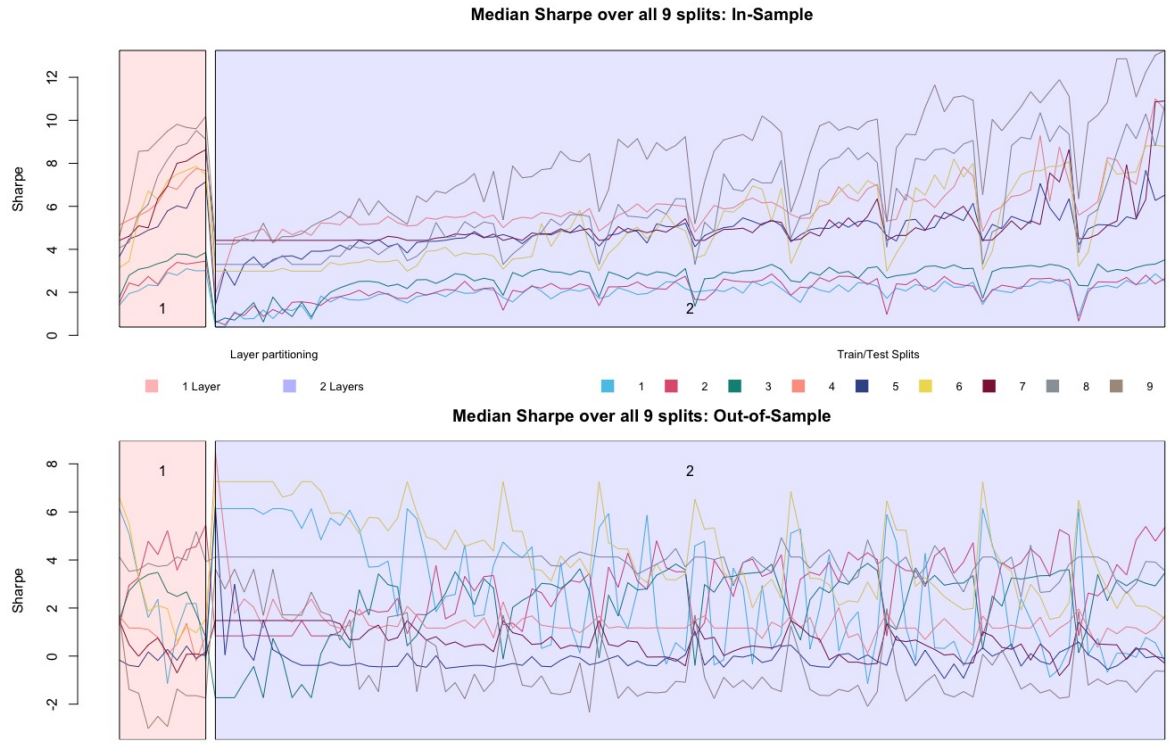

Figure 22: Sharpe median over all 9 splits.

When looking at the Sharpe ratios, an inverse relationship between the in-sample and out-of-sample can be seen. The in-sample Sharpe ratio improves in most time splits with increasing complexity. In the out-of-sample, however, the medians decrease. It can also be seen that some of the Sharpe ratios show periodic spikes again. Apart from these aspects, no clear patterns or correlations could be identified. To put these numbers into perspective, an one-time investment in the S&P 500 ten years ago would have resulted in a Sharpe ratio of 0.83 [25].

3.2.3. Model selection

- Vorschlag: zwei Modelle (z.B. c(10,10) und c(7,7) wählen, die irgendwie Sinn machen und dann mit Diebold Mariano Predictive Accuracy vergleichen. Das bessere wird verwendet.

3.2.3.1 Benchmark -buy and hodl kurz erklären im out of sample (alle grünen beim 9 split)

wiso ist der so stark

schwierig zu schlagen

3.3. Trading strategy

This chapter describes the trading strategy in more detail. In basic terms, the findings of the previous chapters are extended with considerations from explainable artificial intelligence (XAI) as well as from more traditional tools used in time series analysis. The following flowchart in figure 23 shows a broad overview of how the different factors are combined to generate the final trading signal.

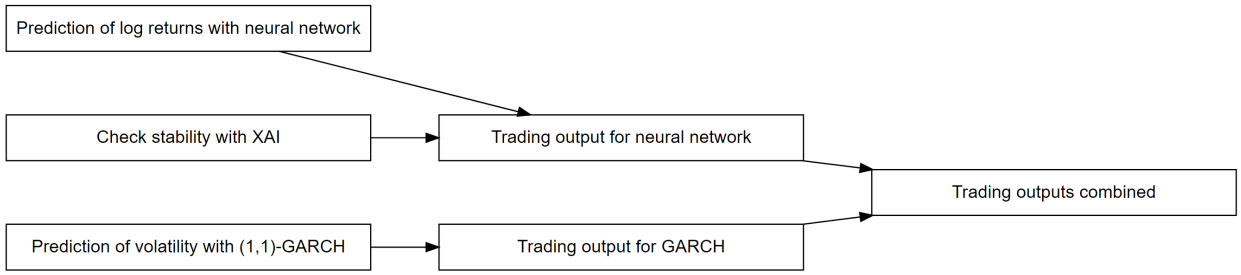

Figure 23: This flowchart illustrates an overview of the trading strategy applied in this chapter.

The main component is the prediction of the neural network. This reflects the expected log return of the next day and is thus an important indicator of whether the money should remain invested or not. However, the output of these neural networks are to be treated with caution, as we have seen in the chapter 3.2.2. Therefore, we rely on XAI to detect instability and incorporate this information into the final trading signal. Furthermore, volatility persistence is observed in financial time series, i.e. large changes can be expected after large changes. Leverage effects are also probable, which means that the tendency to achieve a negative return is higher when volatility is large. A GARCH model is used to model these phenomena, whose volatility predictions provide important information for the final trading signal.

While these methods sound promising, it should not be forgotten that a buy-and-hold strategy (given nerves of steel) also led to remarkable performances in the past that even outperformed traditional asset classes. The price development is described in chapters 2.3.1. and 2.3.3.. That is why buy-and-hold is frequently used as a comparison.

For trading the returns, we must first define the environment and make some assumptions. Tradingcosts are crucial in high frequency trading, nevertheless we assume that transaction costs are non exsistant. Further we assume the possibility of shortselling in BTC. These assumptions allow us to wether stay in the market(signal = 1), to exit the market (signal = 0) or to sell tomorrows return (signal = -1). If in the two latter tomorrows return is negative, a gain in performance is realized.

3.3.1. Trading with neural networks and LPD

In chapter 3.2.3 we have decided to fix the architecture 7 layers 7 nets. In the following we describe how in this case the neural network is used, in order to create a trading signal. Further we use the LPD, explained in chapter 2.4.2., based on the computed neural networks to derive an additional signal for trading.

3.3.2. Neural network

Due to its randomness in finding the optimum, it is proposed by the Central Limit Theorem [26] to use an approximate number of iterations $N_{total} \geq 100$ for an accurate mean value. In order to find a balance between precision and computation time, we found 1000 iterations should be sufficient.

Let p be the predicted value of the neural network in a certain time $_t$. With formula $f(p_k)$ in every $_t$ a trading signal is derived by majority decision. How clearly the decision of the forecasts must be, is decided by parameter β. For further evaluation of neural networks β can be used as an optimizing factor.

$$f(p_k) = \begin{cases} 0 & \text{if } \frac{1}{N_{total}} \sum_{k=1}^{N_{total}} signum(p_k) < \beta \\ signum(\sum_{j=1}^{N_{total}} p_k) & \text{else} \end{cases}$$

N_{total} = Total number of neural networks

β = Ratio of majority decision

p_k = Predicted value from neural network

3.3.3. LPD signal

In the same manner as before, for every k of the N_{total} network in every time step t the derivative as in chapter 2.4.2. is calculated. From the insample lpd the mean $\bar{Y}$ 25 and the standarddeviation sdY 24 for every lag_j is derived. In the out of sample $\bar{Y}_j$ and sdY_j are used to generate a signal by the following:

With Formula 26 in every lag_j it is decided wether the predicted lpd exceeds $\bar{Y}_j \pm \lambda sdY_j$. In Figure 24 this procedure is visualized, just for one lag due to a smoother visualisation.

Function $g(X)$ 26 is further applied in $lpdsignal(x_t)$ to every timestep $_t$ in order to generate a signal with $lpdsignal(x_t)$. The output of $g(X)$ 26 is between 0 and lag_j.

With η_{lower} and η_{upper} it is decided which signal output the lpd $_t$ will be assigned. We have decided; for values bigger than η_{upper} we conclude a big change is likely. In chapter 3.1. last section we observed, that larger negative returns are more likely then positive, therefore the proposed signal is 0. For values between η_{lower} and η_{upper} we propose a signal 0.5.

$$sdY = \sqrt{\frac{1}{n_{in}} \sum_{t=1}^{n_{in}} (Y_t - \bar{Y})^2} \tag{24}$$

$$\bar{Y} = \frac{1}{n_{in}} \sum_{t=1}^{n_{in}} Y_t \tag{25}$$

$$\mathrm{g(X)} = \sum_{j=1}^{lag_n} (X_j > \bar{Y}_j + \lambda sdY_j) \vee (X_j < \bar{Y}_j - \lambda sdY_j) \tag{26}$$

$$lpdsignal(x_t) = \begin{cases} 0.5 & \text{if }, \eta_{lower} < g(x) \leq \eta_{upper} \\ 0 & \text{if }, \eta_{upper} < g(x) \\ 1 & \text{else} \end{cases}$$

Y_t = insample lpd observed

X_t = outsample lpd predicted

lag_j = lags

lag_n = maximum number of lags

λ = scaling parameter for standard deviation

$\eta_{lower}, \eta_{upper}$ = lower and upper border for signal decision

$g(X)$ = Function of lags exceeding band $\in \{0..\mathrm{j}\}$

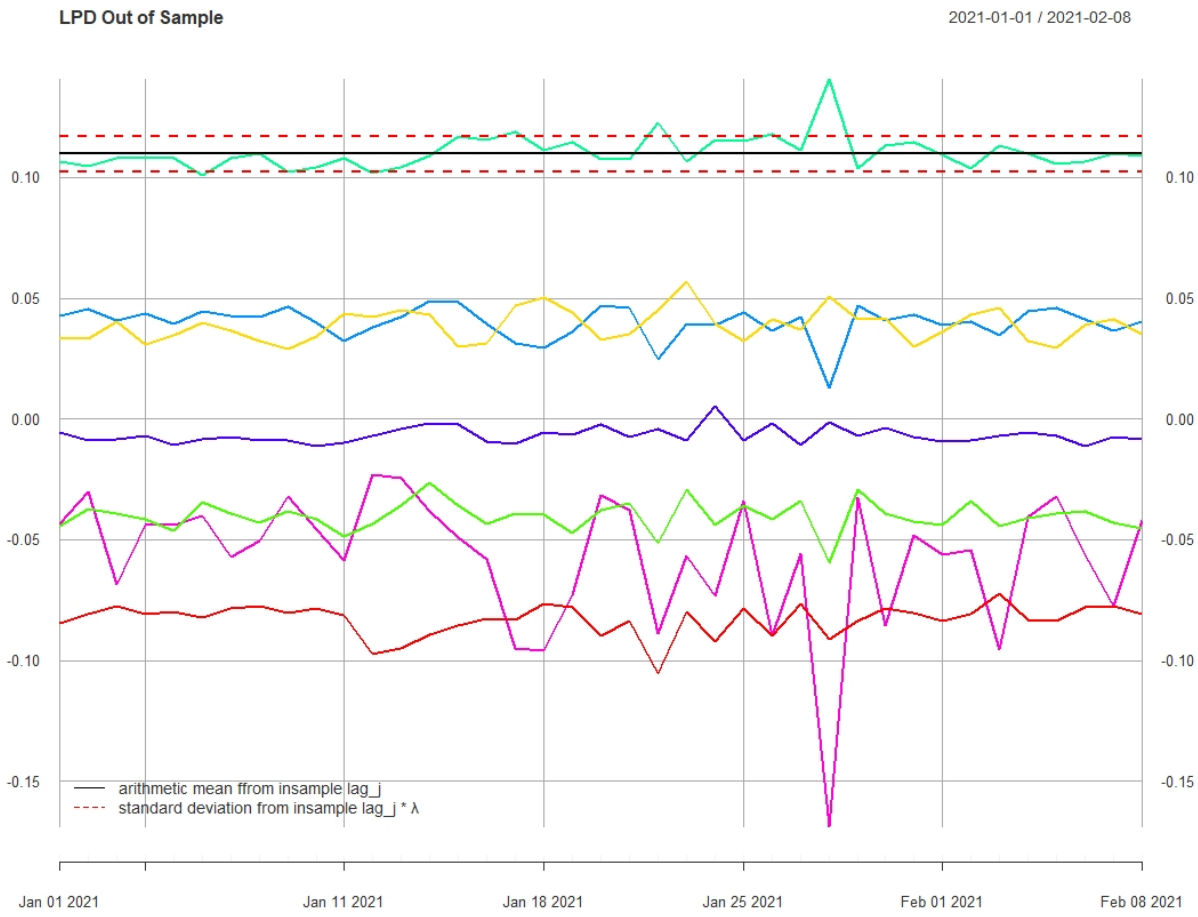

Figure 24: LPD Plot for illustration

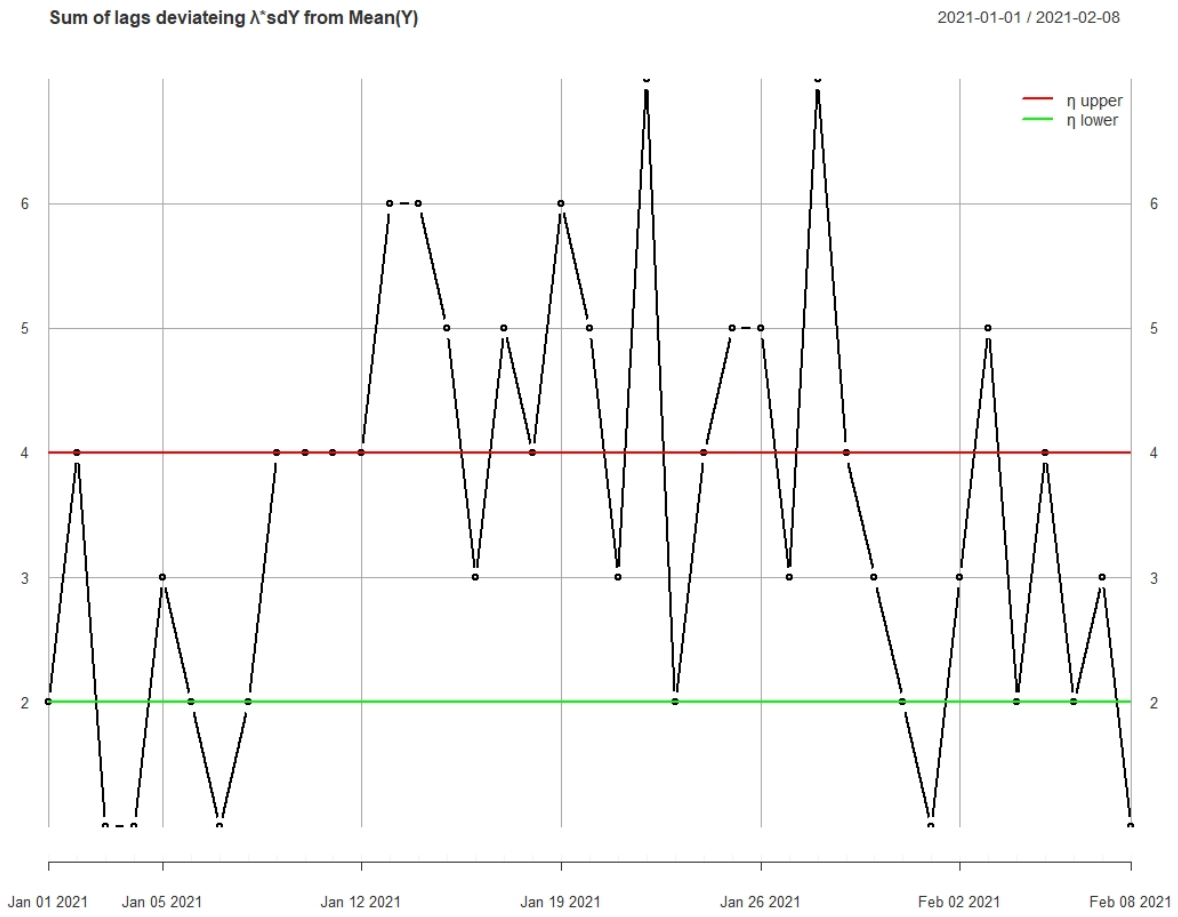

Figure 25: LPD Sum Plot for illustration

3.3.5. GARCH volatility predictions

In a further step, we examine the time series with a traditional GARCH model that allows heteroskedasticity. T. Bollerslev proposes to use this to model time-dependent variance as a function of lagged shocks and lagged conditional variances [27]. Based on an ARMA(1,1)-GARCH(1,1), we conduct one-step-ahead predictions using a rolling window of size 365 days and refit the model after 30 days. This results in two different trading strategies of which one is based on the signs of the forecasts and the other on predicted volatility. The predictions of future volatilities are presented in the appendix in figure 39.

The first trading strategy is based on one-step-ahead rolling window forecasts (here: log return) resulting from the ARMA(1,1)-GARCH(1,1). When we predict a positive value, the algorithm decides to enter, respectively remain in a long position. When predicting a negative value, we enter a short position to benefit from the anticipated market movement.

The second strategy is solely based on volatility predictions resulting from a GARCH(1,1) and tries to take an advantage from the asymmetric volatility phenomenon by F. Black. This phenomenon describes a negative correlation between the volatility of the return and the achieved return [28]. Therefore, we define the following trading rule:

FOR EACH i in $\sigma_{predicted}$
  IF $\sigma_{predicted,i} \geq 1.64\sigma_{historical}$:
    $signal_i = 0$;
  ELSE
    $signal_i = 1$;

In other words, the code checks whether the predicted volatility is significantly greater than the 95% confidence interval (based on historical data). If this is the case, the trading signal is set to 0, i.e. the position is sold respectively we stay out of the market. If the expected volatility is within the normal range, we enter respectively remain in a long position. For simplicity, a threshold of 1.64 is used, which corresponds approximately to the upper 95% confidence interval for a standard normal distribution.

Figure 26 illustrates how the GARCH trading strategies shown as well as a buy-and-hold strategy would have performed in a backtesting. It is easy to see that buy-and-hold outperforms the other two. The GARCH Signum strategy misjudges the situation at key points in time and thus hurts the return and Sharpe Ratio. Only during the Covid-19 crash in March 2020, the GARCH Signum strategy suffered a smaller loss. On the other hand, we missed the following bull run. Also, the horizontal lines in the GARCH Volatility strategy are clearly visible at which the predicted volatility is too large and thus the market exits. While some draw-downs are dampened, upside moves are also avoided.

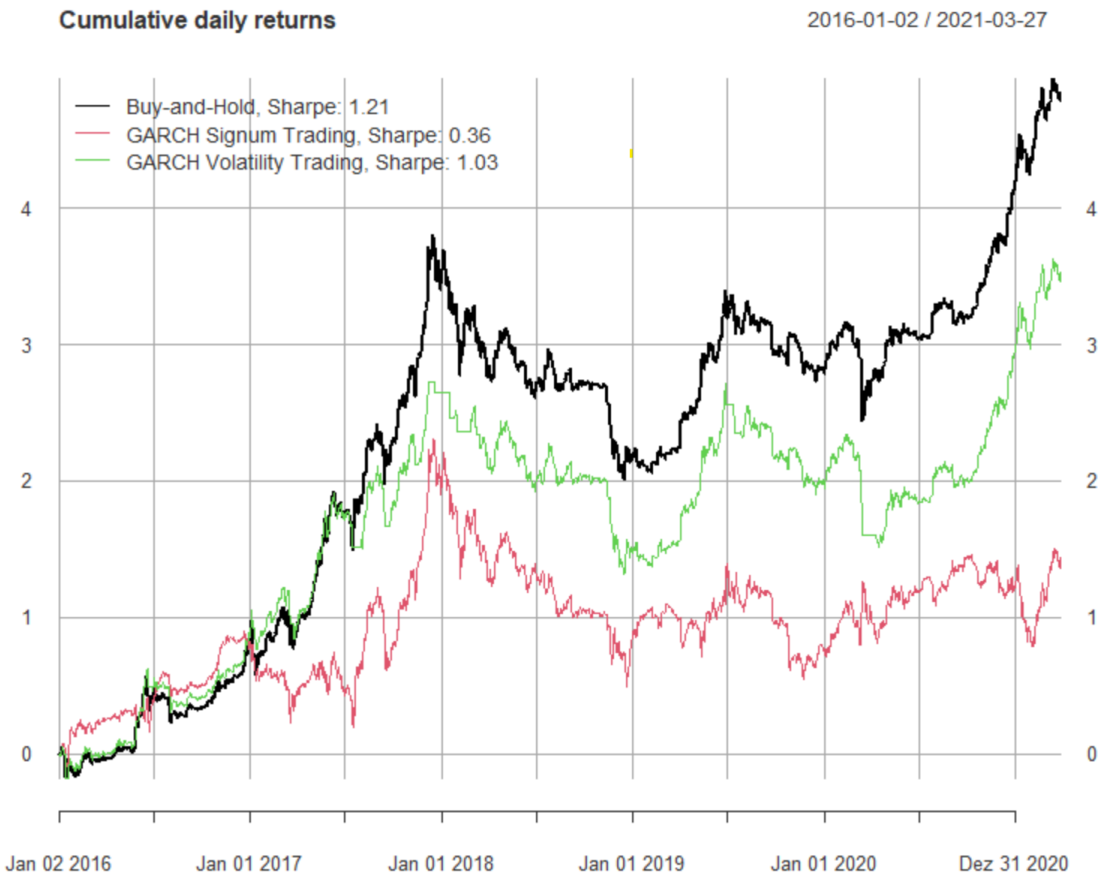

Figure 26: Cumulative daily returns of two different GARCH trading strategies and a simple buy-and-hold strategy. The GARCH Signum strategy is based on the ARMA(1,1)-GARCH(1,1) prediction, while the GARCH Volatility strategy simply checks if the threshold is exceeded. The time periods where we quit the market is clearly visible as horizontal lines.

3.3.4. nn and lpd estimation

the 9 splits which were in chapter 3.2.1. introduce are now used for trading. The out of sample performances of every split are added together and are compared with the benchmark from bench.

With the previously established architecture only the three parameters λ, $\eta_{lower}, \eta_{upper}$, β are optimized for performance.

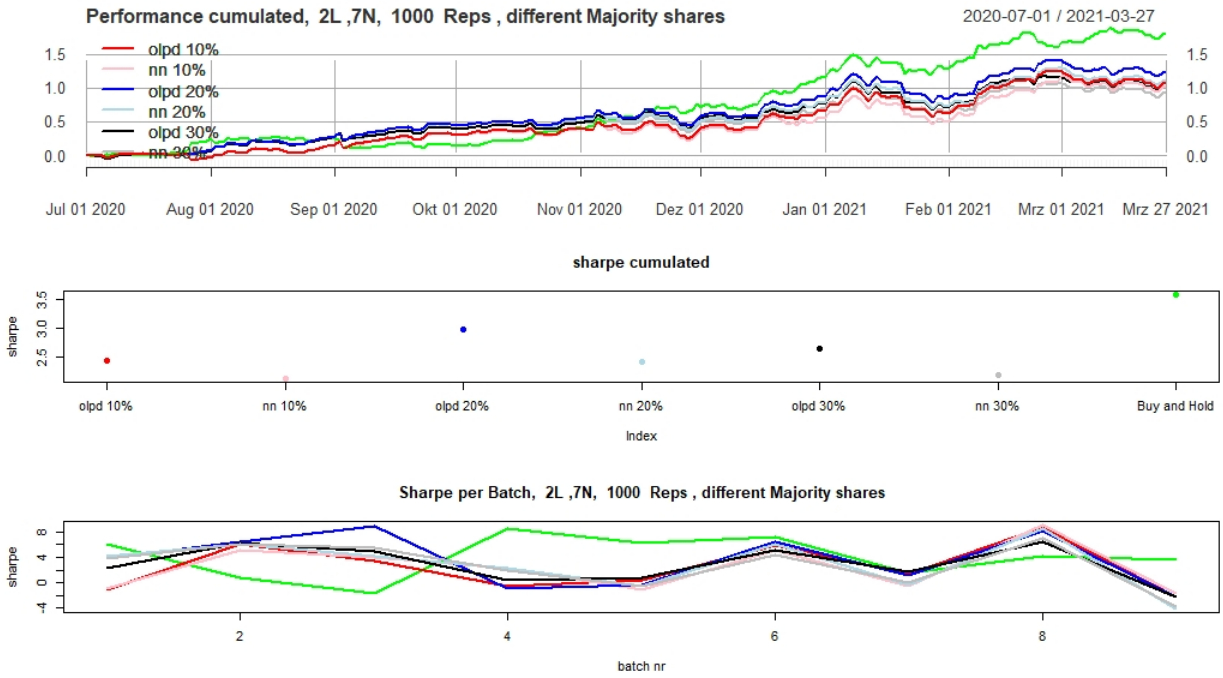

Figure 27: Neural network LPD results

3.3.6. Kapitel: Resultat nicht wie erhofft

- Vorschläge:
- Resultat nochmals prüfen mit “gutem” Seed
- Versuchen auf andere Zeitreihe
- Preise nehmen anstatt log returns

4. Results

4.1. Results chapterino

5. Conclusion

Best Trading Algorithm ever!

5.1. Get rich or die tryin

Neque volutpat ac tincidunt vitae semper quis. At elementum eu facilisis sed odio morbi quis commodo odio. Eget dolor

5.2. Be GME stock, or not to be GME stock

Tellus at urna condimentum mattis pellentesque id nibh. Morbi tempus iaculis urna id volutpat lacus laoreet. Sem fringilla

References

[1] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review, 1958, pp. 386–408.

[2] P. L. B. Martin Anthony, *Neural network learning: Theoretical foundations.* Cambridge University Press, 1999.

[3] R. Rojas, *The backpropagation algorithm.* Springer Berlin Heidelberg, 1996, pp. 149–182.

[4] G. B. O. Yann Lecun Leon Bottou, *Efficient BackProp.* Image Processing Research Department AT& T Labs, 1998, pp. 1–44.

[5] L. Hunsberger, "Back propagation algorithm with proofs." https://www.cs.vassar.edu/~hunsberg/cs365/handouts-etc/backprop.pdf (accessed Mar. 21, 2021).

[6] M. A. J. I. Hassan Ramchoun Youssef Ghanou, *Multilayer perceptron: Architecture optimization and training.* International Journal of Interactive Multimedia; Artificial Intelligence, 2016, p. 26.

[7] A. Karpathy, "A recipe for training neural networks." https://karpathy.github.io/2019/04/25/recipe/ (accessed Mar. 24, 2021).

[8] M. N. S. S. Ke-Lin Du, *Recurrent neural networks.* Springer London, 2014, pp. 337–353.

[9] S. Y. Fei-Fei Li Justin Johnson, *Lecture 10: Recurrent neural networks.* Stanford University, 2017.

[10] R. S. M. Francis X. Diebold, *Comparing predictive accuracy.* University of Pennsylvania, 1995.

[11] U. Triacca, *Comparing predictive accuracy of two forecasts: The diebold-mariano test.* Universita dell Aquila, NA.

[12] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system.* online: www.bitcoin.org, 2008, p. 9.

[13] U. W. Chohan, *A history of bitcoin.* University of New South Wales, Canberra, 2017.

[14] J. Edwards, "Bitcoins price history." https://www.investopedia.com/articles/forex/121815/bitcoins-price-history.asp (accessed Mar. 01, 2021).

[15] J. Frankenfield, "Block rewards." https://www.investopedia.com/terms/b/block-reward.asp (accessed Apr. 18, 2021).

[16] L. Wintermaier, "Bitcoin's energy consumption is a highly charged debate – who's right?" https://www.forbes.com/sites/lawrencewintermeyer/2021/03/10/bitcoins-energy-consumption-is-a-highly-charged-debate--whos-right/?sh=41f655eb7e78 (accessed Mar. 26, 2021).

[17] C. C. for Alternative Finance, "Live bitcoin electricity consumption index." https://www.forbes.com/sites/lawrencewintermeyer/2021/03/10/bitcoins-energy-consumption-is-a-highly-charged-debate--whos-right/?sh=41f655eb7e78 (accessed 2021).

[18] C. C. for Alternative Finance, "Live bitcoin electricity consumption map." https://cbeci.org/mining_map (accessed 2021).

[19] A. Meynkhard, *Fair market value of bitcoin: Halving effect.* Investment Management; Financial Innovations, 2019, pp. 72–85.

[20] F. L. Konstantinos Gkillas, *Is bitcoin the new digital gold? Evidence from extreme price movements in financial markets.* SSRN Electronic Journal, 2018.

[21] W. Knight, "MIT technology review: The u.s. Military wants its autonomous machines to explain themselves." https://www.technologyreview.com/2017/03/14/243295/the-us-military-wants-its-autonomous-machines-to-explain-themselves/ (accessed May 06, 2021).

[22] M. Wildi, *XAI time series.* ZHAW, 2021, p. 49.

[23] F. Pretto, "Uncovering the magic: Interpreting machine learning black-box models." https://towardsdatascience.com/uncovering-the-magic-interpreting-machine-learning-black-box-models-3154fb8ed01a (accessed May 01, 2021).

[24] R. R. Georgios Sermpinis Andreas Karathanasopoulos, *Neural networks in financial trading.* Springer Science+Business Media, LLC, part of Springer Nature 2019, 2019, pp. 204–308.

[25] Morningstar, "S&p 500 PR sharpe ratio." https://www.morningstar.com/indexes/spi/spx/risk (accessed Apr. 24, 2021).

[26] P. Pfeiffer, "13.2: Convergence and the central limit theorem." https://stats.libretexts.org/Bookshelves/Probability_Theory/Book%3A_Applied_Probability_(Pfeiffer)/13%3A_Transform_Methods/13.02%3A_Convergence_and_the_Central_Limit_Theorem#:~:text=Central%20Limit%20Theorem%20(Lindeberg%2DL%C3%A9vy%20form)&text=By%20the%20convergence%20theorem%20on,)%E2%86%92%CF%95(t).&text=The%20theorem%20says%20that%20the,does%20not%20tell%20how%20fast. (accessed Aug. 05, 2020).

[27] T. Bollerslev, *Generalized autoregressive conditional heteroskedasticity.* Journal of Econometrics, 1986, pp. 307–327.

[28] F. Black, *Studies of stock price volatility changes.* Proceedings of the 1976 Meetings of the American Statistical Association, 1976, pp. 177–181.

Attachment

This project work is created with R-4.0.2 , RStudio Version 1.4.904 and RMarkdown in collaborative working via Git / Github

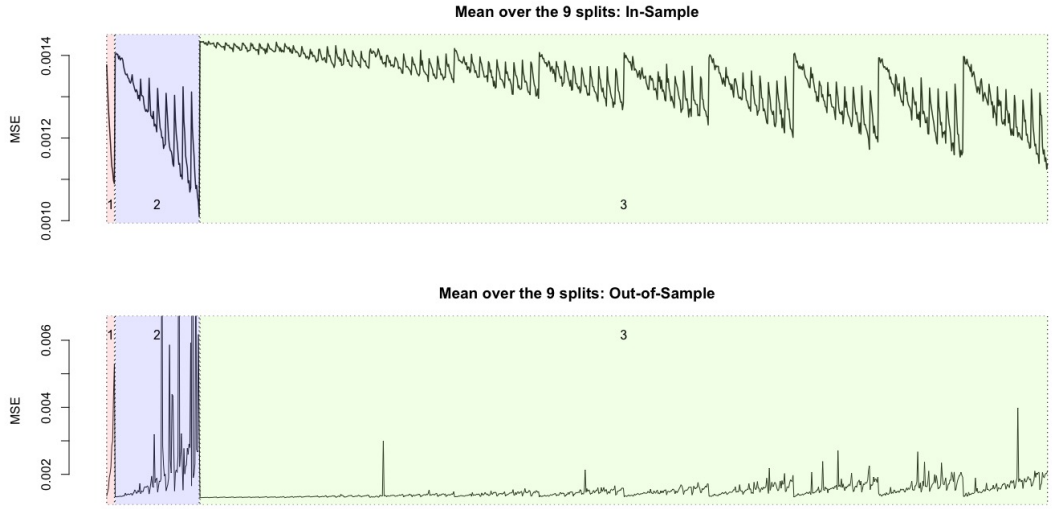

Figure 28: MSE mean over all 9 splits with all 3 layers.

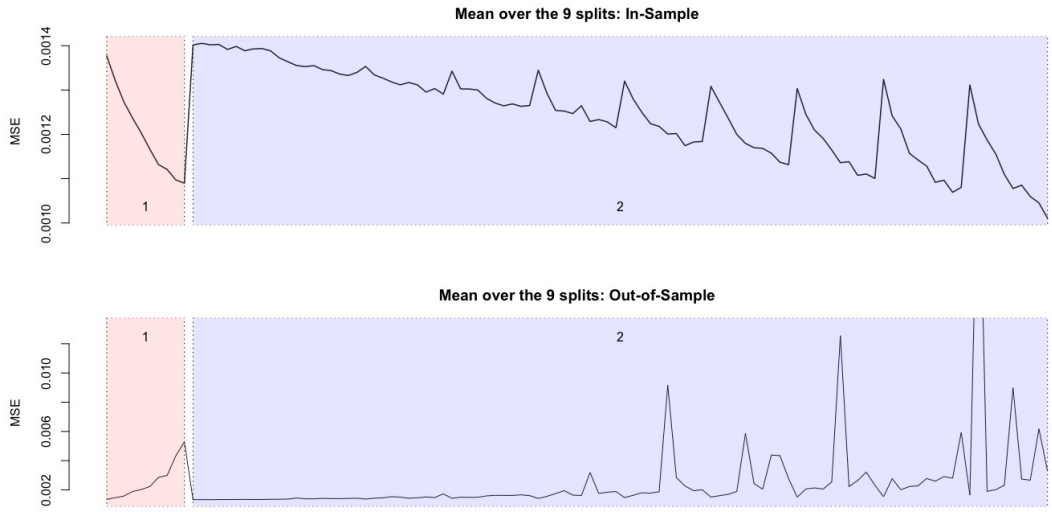

Figure 29: MSE mean over all 9 splits with only 2 layers.

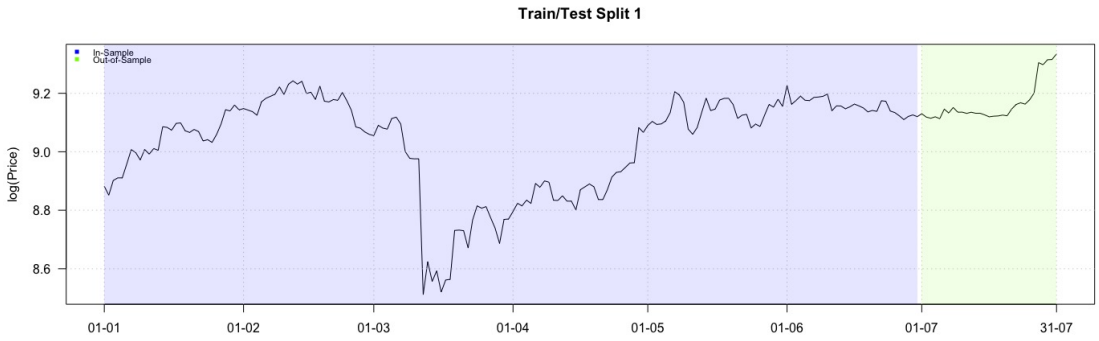

Figure 30: Split 1.

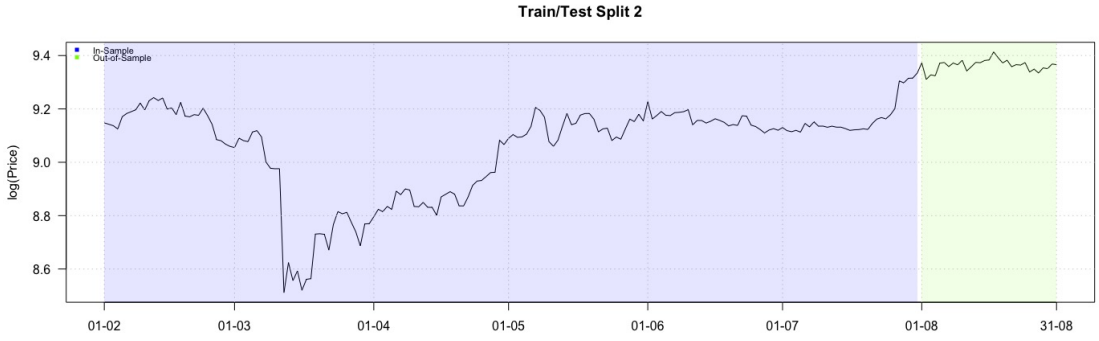

Figure 31: Split 2.

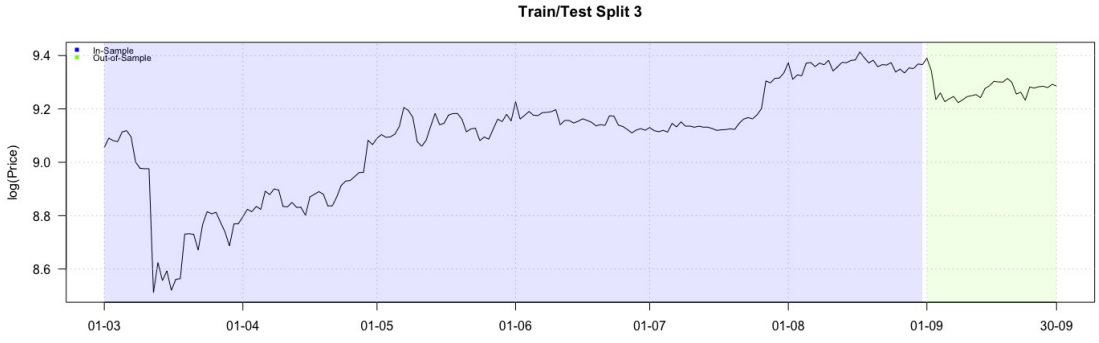

Figure 32: Split 3.

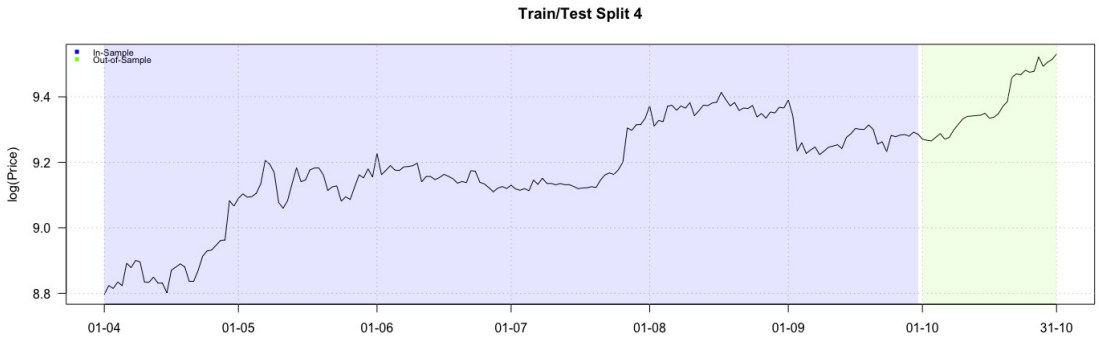

Figure 33: Split 4.

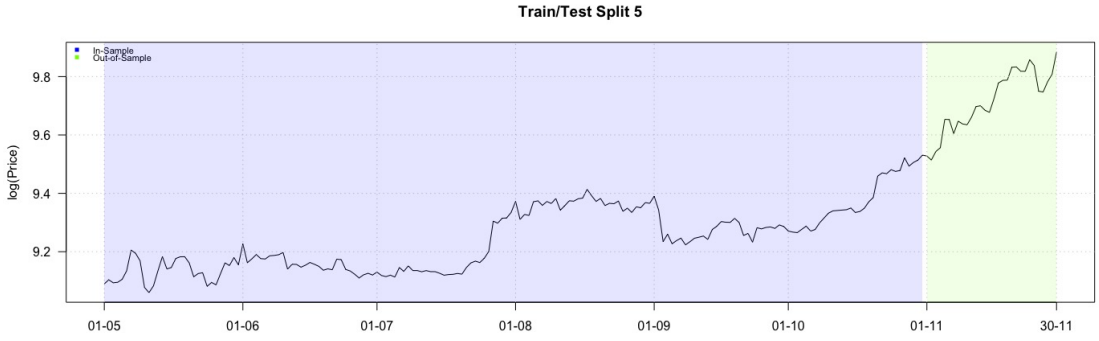

Figure 34: Split 5.

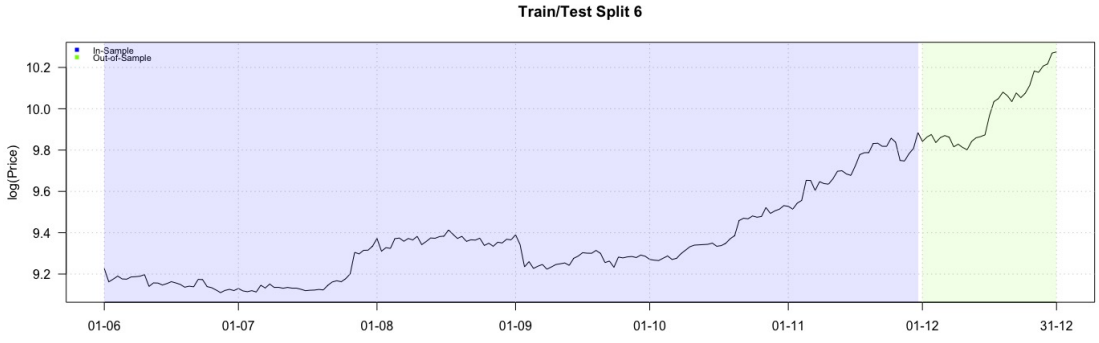

Figure 35: Split 6.

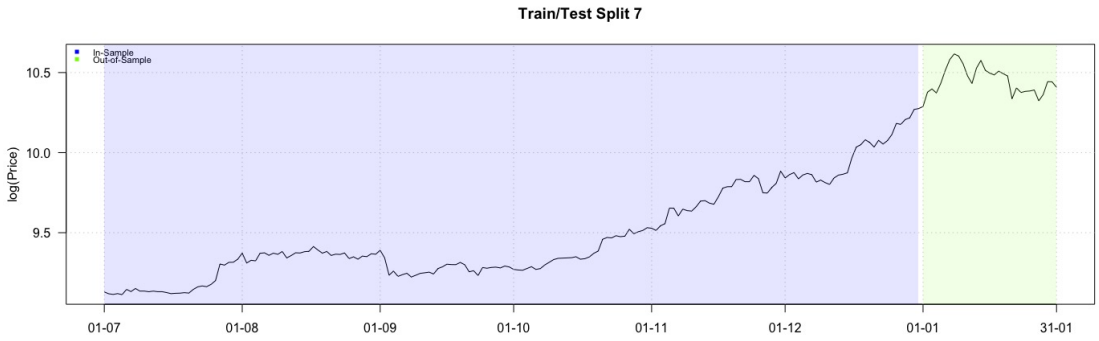

Figure 36: Split 7.

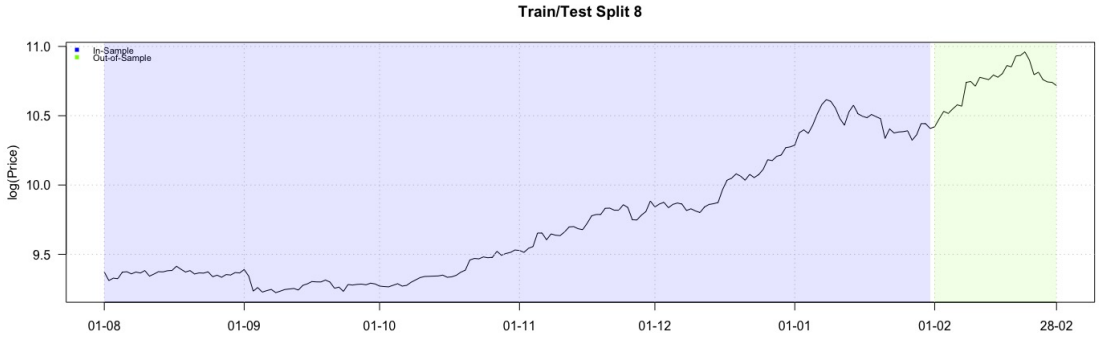

Figure 37: Split 8.

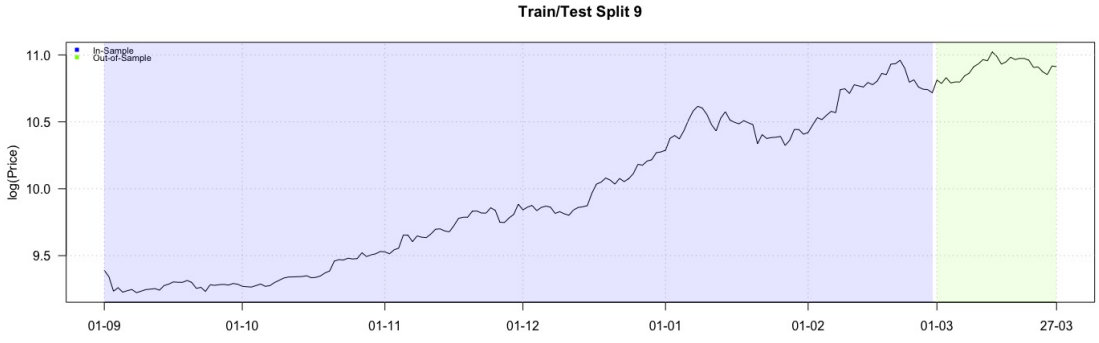

Figure 38: Split 9.

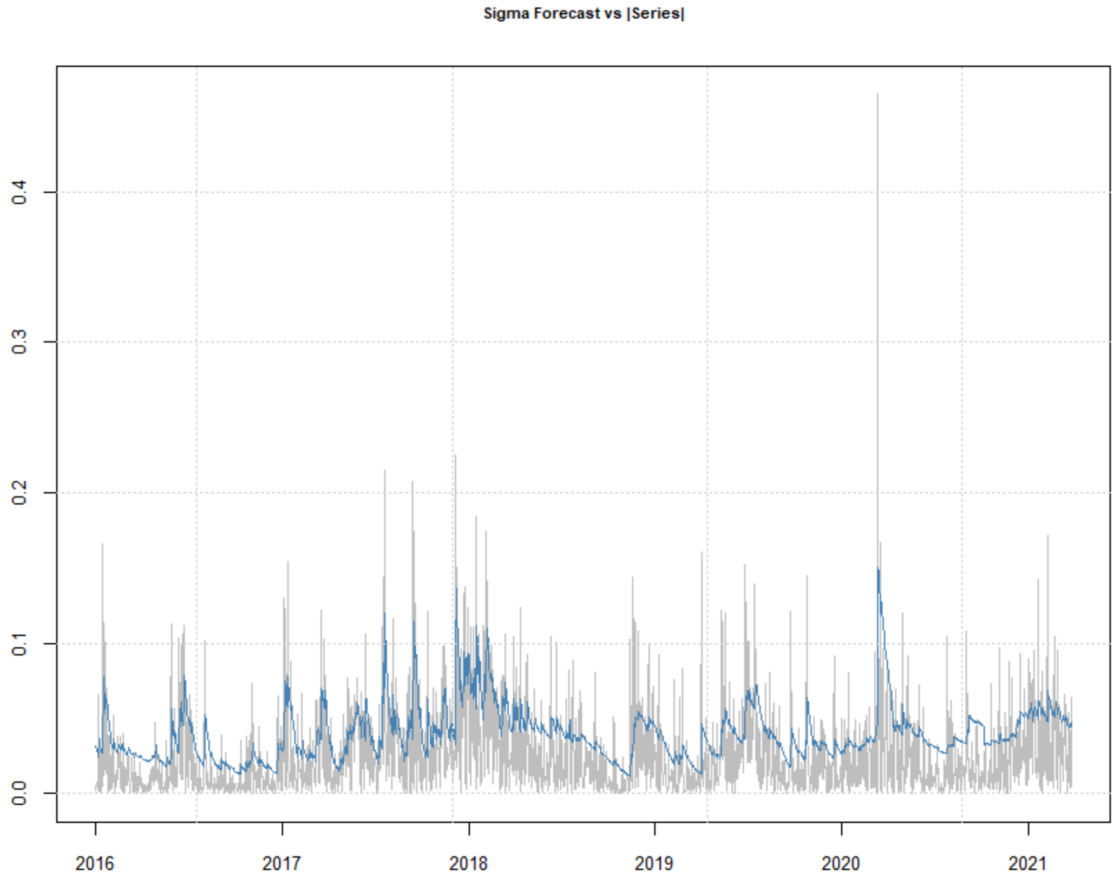

Figure 39: One-step-ahead forecasts of volatility using rolling window of size 365. Refitting model after every months.