Zurich University
of Applied Sciences

School of
Engineering
IDP Institute of Data Analysis
and Process Design

Bachelor thesis Engineering and Management

Bitcoin Trading using Deep Learning Neural Networks and Explainable Artificial Intelligence

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

We are in an age in which more and more application areas for machine learning are found. The possibilities are also being explored in the field of finance. However, unlike a simple method like linear regression, deep learning methods are rather complex and not tangible for potential investors. This thesis deals with the application of neural networks, their explainability, and the trading of Bitcoin. The theory of the neural networks used and the novel explainability aspect is explained, furthermore, there is a brief insight into the Bitcoin theory. We use the autocorrelation of log returns to find the input layer of the optimal network architecture. The number of hidden layers and neurons is then determined by quantitatively comparing the error function MSE and the Sharpe ratio for each possible combination. An XAI application, Linear Parameter Data (LPD), is used to determine in which phases the neural network is reliable. Combined with a volatility prediction from a GARCH model, a trading strategy is implemented and improved with the addition of the cryptocurrency Ether. The results are promising, as the inclusion of LPD lead to an added value in our scope. Furthermore, the addition of trades based on Ether enables to even outperform a Bitcoin buy-and-hold strategy.

Zusammenfassung

Wir befinden uns in einem Zeitalter, in dem immer mehr Anwendungsbereiche für maschinelles Lernen gefunden werden. Auch im Bereich der Finanzen werden die Möglichkeiten erforscht. Im Gegensatz zu einer einfachen Methode wie der linearen Regression sind Deep-Learning-Methoden jedoch recht komplex und für potenzielle Investoren nicht greifbar. Diese Arbeit beschäftigt sich mit der Anwendung von neuronalen Netzen, deren Erklärbarkeit und dem Trading von Bitcoin. Es wird die Theorie der verwendeten neuronalen Netze und der neuartige Aspekt der Erklärbarkeit erläutert, außerdem gibt es einen kurzen Einblick in die Bitcoin-Theorie. Wir verwenden die Autokorrelation der Log-Renditen, um den Input-Layer der optimalen Netzwerkarchitektur zu finden. Die Anzahl der Hidden-Layer und Neuronen wird dann durch den quantitativen Vergleich der Fehlerfunktion MSE und der Sharpe Ratio für jede mögliche Kombination bestimmt. Eine XAI-Anwendung, Linear Parameter Data (LPD), wird verwendet, um zu bestimmen, in welchen Phasen das neuronale Netzwerk zuverlässig ist. Kombiniert mit einer Volatilitätsvorhersage aus einem GARCH-Modell wird eine Tradingstrategie implementiert und mit dem Zusatz der Kryptowährung Ether verbessert. Die Ergebnisse sind vielversprechend, da die Einbeziehung von LPD zu einem Mehrwert in diesem Rahmen führt. Darüber hinaus ermöglicht das Hinzufügen von Trades, welche auf Ether basieren, sogar eine bessere Performance als eine reine Buy-and-Hold Strategie.

1. Introduction

The well-known asset classes such as equities, bonds, and money market instruments have existed for several decades and have established themselves as a fixed component in many portfolios. The approach often taken is that, in principle, the equity component is responsible for the main return and the remaining asset classes are used to optimize the risk/return ratio. Recently, there has been much discussion about a new asset class that has a low correlation to the equity markets, very high volatility, but also enormous opportunities for profit [1]. The best-known cryptocurrency with the largest market capitalization at the moment is Bitcoin. Compared to traditional asset classes, the characteristics of the Bitcoin market are relatively unexplored. Especially for the reason that only about ten years of historical data is available for Bitcoin. In addition to this aspect, the costs for trading Bitcoin are surprisingly low and are in the per mille range when using exchanges like Binance or Kraken. These characteristics make Bitcoin an attractive investment vehicle to test the application of artificial neural networks. In this sense, this thesis investigates how the use of feedforward neural networks with their non-linear properties can be used to build a profitable trading strategy. Three central questions are pursued:

1) What influence does the selection of the network architecture have on the quality of predictions? What influence can be found concerning the Sharpe ratio of a simple trading strategy?

2) Can aspects of explainable artificial intelligence (XAI) help to make the functioning of neural networks intuitively comprehensible?

3) How can the acquired information about neural networks, XAI, and general time series analysis be used to define an efficient trading strategy?

To investigate these aspects, the first step is trying to find an optimal network architecture. For this purpose, all possible combinations between the simplest model (one layer and one neuron) and the most complex model (three layers with ten neurons each) are tested and the results quantified using the common error function mean squared error (MSE) and the Sharpe ratio. Unlike linear regression, the neural network cannot be intuitively explained by the partial effects because it behaves non-linearly. Therefore, the linear parameter data (LPD) method is used to investigate how the network behaves with input stimuli. Furthermore, the volatility clusters typical for financial time series are investigated and evaluated with a simple GARCH model. The basic assumption is that the Bitcoin market, like stock markets, behaves asymmetrically. This phenomenon describes the tendency of market volatility to be higher in declining markets than in rising markets. As a further idea, it is suggested to exit the long position in certain phases and to switch to Ether (ETH) as an alternative investment instrument. These aspects are combined and defined as a trading strategy.

We believe that the examination of neural networks in the field of (crypto-)finance is of great importance. The benefits of deep learning have already been proven in many applications outside this field. Ever since Siri was launched by Apple Inc. in 2011, people outside the world of science have begun to have a rough idea of what artificial intelligence is. Although the application of neural networks in the use case of Siri and natural language processing (NLP) seems reasonable, the opinion is split in the application in the field of financial time series. Since most movements of financial instruments are based on white noise and thus have almost no dependency structure, the implementation of a meaningful method is challenging. Likewise, integration of neural networks only makes sense if a comprehensible and systematically replicable overperformance can be achieved. This bachelor thesis deals exactly with this topic. The application of neural networks for the prediction and trading of Bitcoin is investigated. Aspects of XAI and volatility modeling are included to possibly shed some light on this black box.

2. Theory

The following section 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 2.4 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) [2]. 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 lagged log returns. 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.

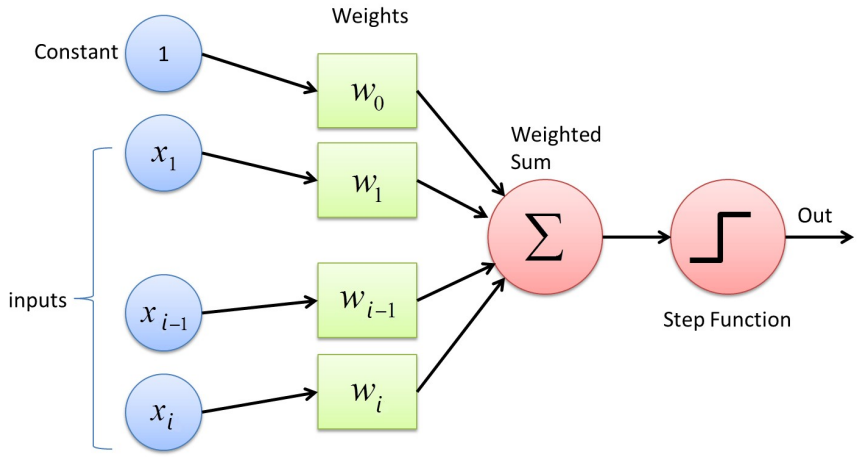

Figure 1: Schematic diagram of a perceptron. The input data is combined with the weights and added up to a weighted sum. After adding an error term (bias), the activation function is applied to the total sum to produce the output.

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, a 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 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 [3]. More about this topic is discussed in section 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 the weights or subtracted from them. Ideally, the weights will gradually converge and provide us with a usable model [3].

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. However, non-linear functions such as the logistic and the hyperbolic tangent functions fulfill these conditions [4].

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.

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

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

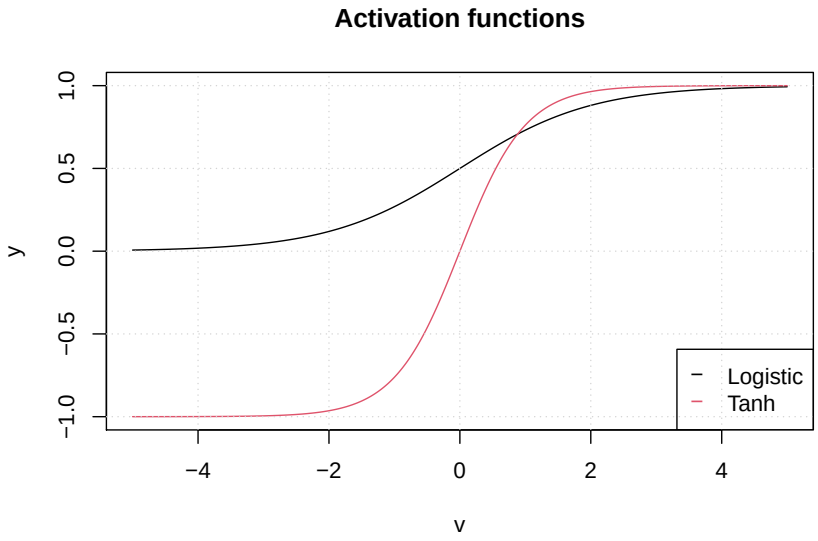

Figure 2: Two common non-linear sigmoid activation functions: In black, the logistic function, which returns values between 0 and 1. In red the hyperbolic tangent function, which returns values between -1 and 1.

In the error analysis, the output of the neural network 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 [5].

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

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 [4].

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

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 learning constant [6]. 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 [2]. 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 [7]. 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, which are also called nodes. These properties (number of layers and nodes) can be summarized with the term 'network architecture' and will be dealt with in this thesis.

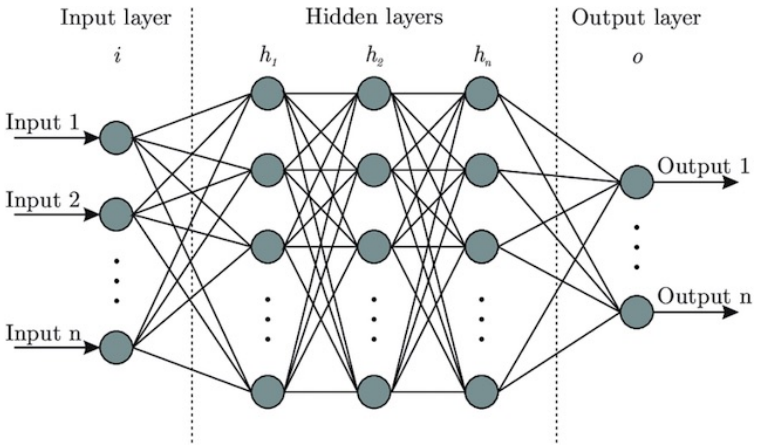

Figure 3: Schematic diagram of a multilayer perceptron. On the left, you see the input layer, on the right the output layer. All layers in between are described as hidden layers. If there is more than one hidden layer, it is referred to as a deep learning neural network.

Every neural network has an input layer (also called features), which consists of one or more nodes. This number is determined by 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 from the last 10 days (lags 1-10), so the input layer would consist of 10 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 and causes overfitting. Subsequently, the model is reduced by focusing on increasing training loss and improving validation loss [8].

2.1.4. Challenges

In this section, we address the two fundamental challenges when dealing with neural networks.

2.1.4.1. Overfitting

We encounter 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 [9].

2.1.4.2. Vanishing Gradient Problem

Another characteristic that requires our attention is the vanishing gradient problem. As explained in section 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 input values 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.

2.2. 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 [10].

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. Considering that a good and simple explanation may depend on investment, would an investor fund something that is not understood and difficult to get a grasp of? 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.

2.2.1. 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. 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) [11].

An approach like ALE (Accumulated Local Effects) examines how the network reacts to change [11]. The features (lags) are compared in order against ALE. The problem with this method is that the features do not contain the dimension of time. It is possible that two very high values are close to each other, but in reality, they occure 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 [12]. Changing the data affects the time dependence of the data. Thus, this method is also unusable for our application.

2.2.2. 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, an older value could suddenly play a bigger role for the model than a current one. This would not add any value to financial time series modelling.

For our application, the lagged log returns of Bitcoin prices are the features to be studied. In this paper, we focus on Bitcoin, here we are already using it as a means of explaining neural networks. A detailed description of BTC follows in sections 2.4 and 3.1. We want to explore 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. This concept strongly resembles a linear regression. The following equation establishes a relationship of the lagged data with the regression coefficients.

$$Y_i = \beta_0 + \beta_1 * x_i^{(1)} + \beta_2 * x_i^{(2)} + ... + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{7}$$

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

$$\text{Data}_t^{(lag=0)} = \text{Intercept} + \beta_1 * \text{Data}_t^{(lag=1)} + \beta_2 * \text{Data}_t^{(lag=2)} + ... + \epsilon_i \tag{8}$$

Now, what could the fitted 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 4. Lags at 6 (red) and 10 (orange) 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.

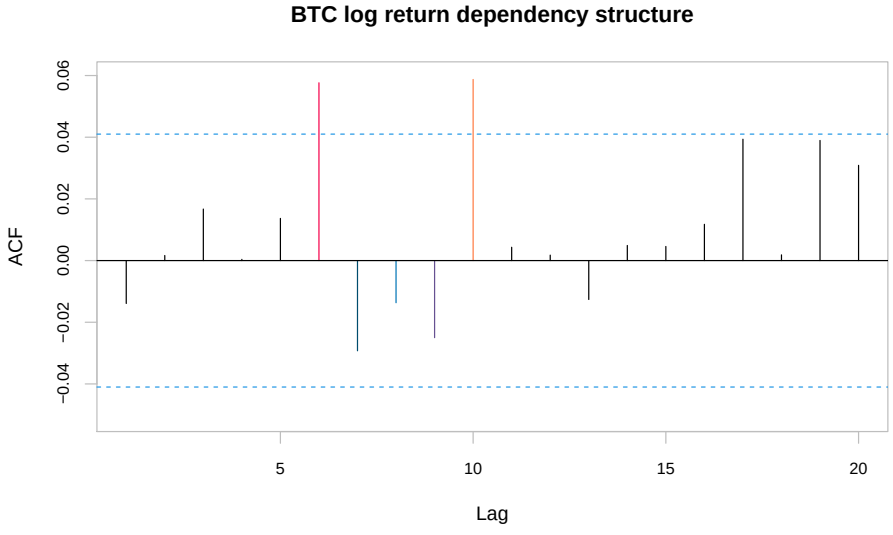

Figure 4: Autocorrelation function of BTC/USD. In red and orange you can see significant positive ACF's at lags 6 and 10. These signal some importance for model considerations. Between 6 and 10, the ACF's are negative and thus would have a negative effect on a model. For further considerations, all lags from 6 to 10 are included.

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 ACF's. Sign and value are in line with the ACF's.

Table 1: Coefficients 6 (red) and 10 (orange) have the greatest positive impact on the model. In between, the coefficients are negative.

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 Δ_{jt}.

$$\Delta_{jt} = \frac{\partial \text{Output}_t}{\partial \text{Data}_t^{(lag=j)}} \tag{9}$$

Basically, we have a similar concept as above with the regression. The difference is that the coefficients are the partial derivatives that establish the relationship between input and output at each point in time. Whereas in regression you would only have 1 coefficient to estimate per input neuron β_i, here you have one coefficient Δ_{it} for each input neuron at each time point t.

$$\text{Output}_t = \text{Intercept} + \Delta_{1t} * \text{Data}_t^{(lag=1)} + \Delta_{2t} * \text{Data}_t^{(lag=2)} + ... + \Delta_{qt} * \text{Data}_t^{(lag=q)} \tag{10}$$

In simpler terms, we train a neural network with input data of length up to lag 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.

$$X_t = x_{1t} + x_{2t} + ... + x_{qt}$$

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.

$$Y_t = x_{1t} + (\delta * x_{1t}) + x_{2t} + ... + x_{qt}$$

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.

$$\Delta_{1t} = \frac{\text{NN}(X_t) - \text{NN}(Y_t)}{\delta * x_{1t}}$$

The output of the neural network changes by the value Δ_{1t} if the input is changed by 1. This procedure can now be performed for each feature at each time point. This gives us the derivatives for each feature and time point, which makes an explanation of neural networks more feasible. We call these newly generated data sets linear parameter data (LPD) [11].

In figure 5, you can see the Δ_{it}, that is, the partial derivatives from equation 10 up to $q = 10$, and you can see the intercept. For the comparison to the linear regression and the autocorrelation, only the lags 6 to 10 are explicitly highlighted. A detailed identification of all LPD's can be seen in figure 48.

It is noticeable that the structure of LPD's strongly resembles the logarithmic returns of BTC. The heteroskedasticity of the log returns, i.e. the volatility clusters are also evident in the LPD's. If the BTC is in a turbulent phase, then the partial derivatives also fluctuate.

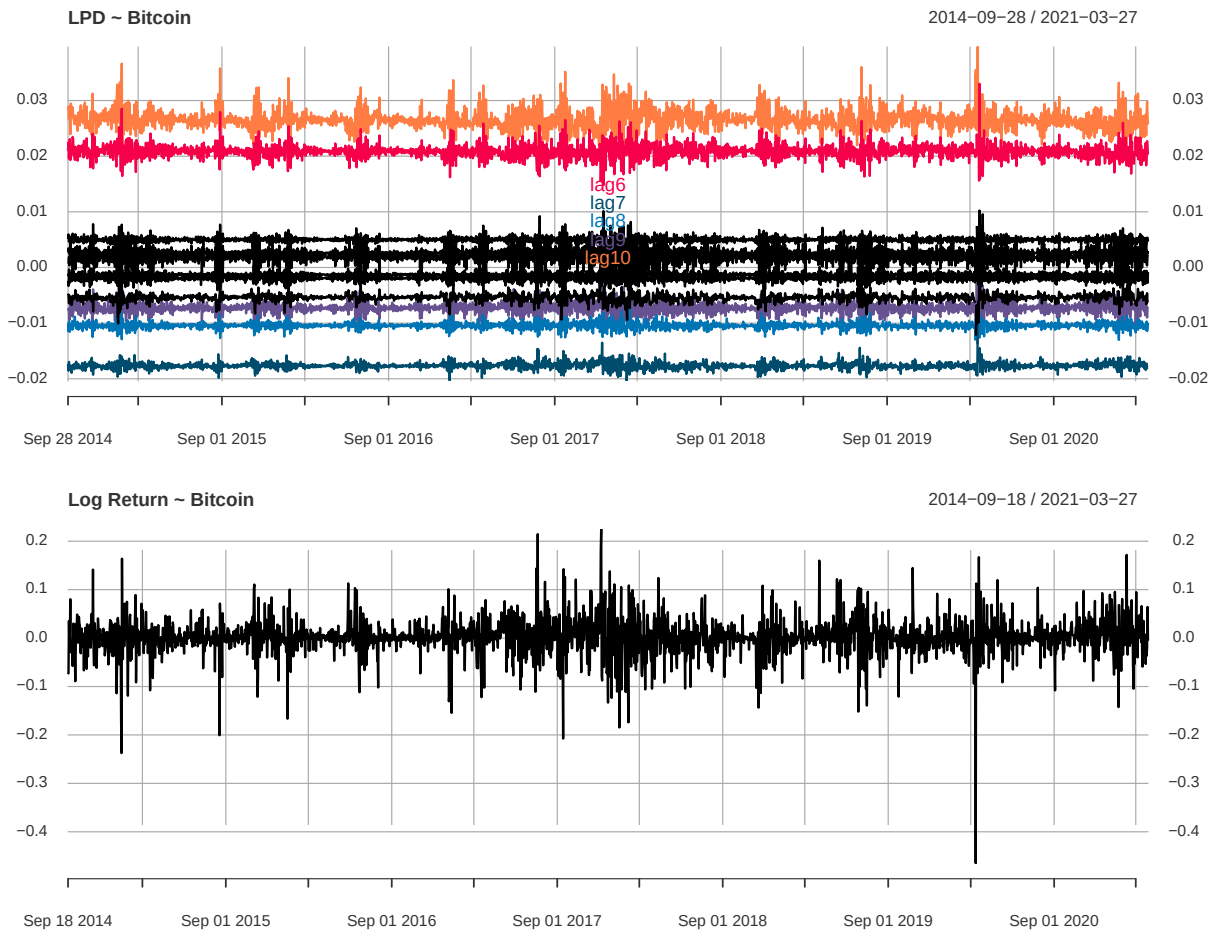

Figure 5: Upper panel: LPD's of BTC's log returns. Only lags between 6 and 10 are highlighted in color. Lag 6 (red) and 10 (orange) have the largest positive impact. 7, 8 and 9 have a negative impact. Lower panel: Logarithmic returns of BTC. Phases with high volatility are visible in both panels, as the input data (log returns) affect the LPD's.

In the autocorrelations (figure 4) and in the regression coefficients (table 1), lags 6 and 10 have a significant positive effect, while lags 7, 8, and 9 have a negative effect. This is also the case for the LPD's. Lag 6 (red) and lag 10 (orange) have a positive sign and have the largest positive effects on an eventual model in terms of absolute value. Lags 7, 8 and 9 (dark blue, light blue and purple) have the largest negative effects here, as well as in the autocorrelations and regression coefficients.

To provide a more accurate and better comparison between regression and XAI, table 2 compares the regression coefficients with the mean values of the LPDs. Although the values are not equal, the signs are the same. With this knowledge, we can say that both procedures are similar. Through this similarity, a connection can be made. As already explained in section 2.2.1, a linear regression is easier to explain and therefore more likely to be used. With this transfer between these two procedures, a bridge can be built.

Table 2: Comparison between linear regression (LR) and linear parameter data (LPD) coefficients. Again, lags 6 and 10 have the largest positive effect in the respective model. Also 7, 8 and 9 all have a negative sign.

	Lag 6	Lag 7	Lag 8	Lag 9	Lag 10
LR	0.05688	-0.03005	-0.01665	-0.02683	0.05899
LPD	0.02084	-0.01764	-0.01042	-0.00728	0.02655

Together with the findings from the visualization in figure 4 and the autocorrelations in figure 5, a connection can be established. Strong autocorrelations, i.e. large dependencies on a given lag, can be recognized in the partial derivatives. With the help of the LPD's and the autocorrelations, the behavior of neuronal networks could be explained with the handling of time series. The lagged input of a neural network seems to have probably a connection with the ACF's.

Thus, we can say that at best, with this method of calculating the partial derivatives for each feature at each time t, we may have found an explanation for the black box of neural networks. With this method we cannot explain how the network led to exactly this prediction using the backpropagation algorithm with the many weights and error terms, but the basic structure, i.e. information which is in the data (for example the autocorrelation), can be found. The ACF is a basic method to find structures in time series and here, with the LPD's you can fall back on exactly this ACF. Despite the complexity of the networks, this special procedure for the XAI leads to the basic ACF's, thus improving the explainability.

Another finding is the correlated behavior of the partial derivatives and the original time series. Volatile log return of the BTC leads to unstable derivatives. In other words, during a turbulent phase in the market, the neural networks are unstable, leading to unstable solutions. This particular behavior can potentially be used for possible trading decisions, which will be explored in section 3.3 .

2.3. Model Comparison

This thesis sets the goal to compare the different neural network architectures presented in order to find the best model. To quantify the quality of the prediction as well as the trading performance, we make use of the following measures.

2.3.1. Sharpe Ratio

The first measure 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.

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma} \tag{11}$$

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.

2.3.2. Mean Squared Error (MSE)

The second 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 12.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\text{realvalue}_i - \text{prediction}_i)^2 \tag{12}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{13}$$

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.

2.4. Bitcoin

In this section, Bitcoin as a cryptocurreny is introduced by analyzing and commenting historical events. Further the technology in and around cryptocurrencies in section 2.4.2 is briefly explained. This section serves to get an overview of the technology and background of Bitcoin. A more detailed description is beyond the scope of this thesis and will therefore be refrained from. We work with the short form 'BTC,' which is the abbreviation for the currency.

2.4.1. Historical Analysis

The story of Bitcoin began with a paper published in the name of an alleged person called Satoshi Nakamoto [13]. The publisher of the document cannot be assigned to a real person, therefore the technology inventor remains mysteriously unknown until today. In 2009, the first Bitcoin transaction was executed. Over the years, several other cryptocurrencies have emerged as a result of the open source-based code. Until 2013, cryptocurrencies operated under the radar of most regulatory institutions. The character traits, including total anonymity, led Bitcoin to become a common alternative to cash in criminal transactions. Silk Road was one of the most famous dark web trading platforms, where the trade of illegal narcotics and fake ID cards flourished. Due to this, Bitcoin was portrayed in a rather bad light by the media and newspapers. The crowning moment was when the drug enforcement agency shut down Silk Road and confiscated 26,000 Bitcoins. Nevertheless, in 2014, more companies, such as Zynga, The D Las Vegas Casinos, Golden Gate Hotel & Casino, TigerDirect, Overstock.com, Newegg, Dell Technologies, and even Microsoft Corporation [14], began to accept Bitcoin as a payment method. In the same year, the first derivative with Bitcoin as an underlying was approved by the U.S. Commodity Futures Trading Commission. In 2015, an estimated 160,000 merchants used Bitcoin to trade. Since then, cryptocurrencies have become increasingly popular among businesses and individuals, which has also been reflected in the exchange market.

Therefore, we examine the price of BTC/USD and the logarithmic price in figures 6 and 7. The latter plot helps to compare the relative price changes as the price level changed dramatically over the years. These historical prices originate from the price-tracking website CoinMarketCap and show data originating from 2014.

In 2010, the events just described caused the price to increase a hundredfold from 0.0008 USD to 0.08 USD [15]. The first bubble phenomenon was observed after that. In 2011, the price moved from USD 1 to USD 32 within three months, after which it settled back at USD 2. The following year, the price climbed to 13 USD and reached a never-before-seen level of 220 USD, only to plunge to 70 USD within two weeks in April 2013. By the end of the year, a rally brought BTC to a price of USD 1156. Negative media reports triggered mixed moods. The trial of Silk Road founder Ross Ulbricht was one of the more notable events that unsettled the market. This time point is marked with the 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. The price soared to 20,000 on 17th September 2017, which can be seen in letter **B**. After this tremendous bull run, BTC lost value for more than a year until mid-December 2018 (letter **C**). The trend reverted and found its peak after 6 months in June 2019 (letter **D**). After this on 3 March 2020, BTC lost nearly half of its value in 4 days, which can be seen in letter **E**. This is likely due to irrational behavior during the outbreak of the Covid-19 pandemic. Subsequently, the price recovered rapidly. Institutional investors and prominent companies such as Tesla Inc. announced their purchase, which in combination with the halving effect in May 2020 led to this effect. At the time of writing this thesis, the registered BTC all-time high is USD 64,805. In summary, the BTC market is strongly driven by emotional trends and is very volatile. The time series is analyzed in more detail in section 3.1.

Figure 6: Price development of BTC in USD from fall 2014 to spring 2021. Events worth mentioning are marked with letters A to E and described in the text above.

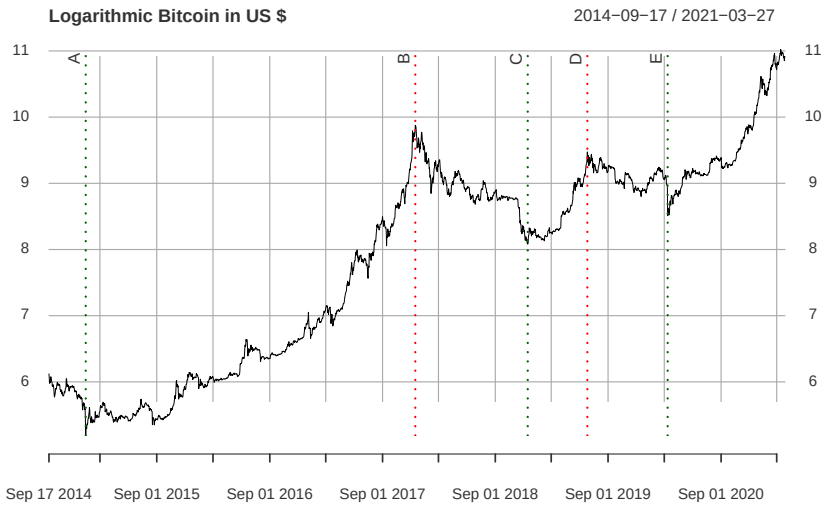

Figure 7: Logarithmic transformation of the price development of the upper chart. In this representation, the relative price changes are better visible. Thus, the events marked with the letters can be better compared with each other. In contrast to the upper chart, here you can see not only the extreme price spikes, but also the rather small price movements.

2.4.2. Cryptocurrencies and Bitcoin Technology

This section 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. 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 from conservative currencies in almost all ways. The cryptocurrencies (not just bitcoin) are based on the blockchain technology introduced in Nakamoto's paper [13]. 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 broadcast into the network to all participants, so that everyone sees a transaction has been made. Around 2400 transactions are packed in a block (the block size is limited by memory) which are broadcast 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 hash function f(previous hash, data of the block, special number), the so-called proof of work. The first miner who finds the special number according to the hash with a certain pattern, gets an amount of Bitcoin in reward. The block with its new hash and the special number is now added to the chain and is broadcast 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 also be the first to find the right hash and so on. In figure 8, the red block is a false one that gets attached and later declined because the other branch is longer. Only the longest chain can 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, it is 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 total number of coins is determined by the rewards of the miners. The first block (genesis block) 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 [16].

In recent days, cryptocurrency has come under a lot of criticism. The immense computing expenditure has a very high power consumption which leads crypto mining companies to build huge farms with massive cooling aggregates. According to the article in Forbes magazine [17] 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 [18]. Right now China [19] contributes 70 % to the hash rate whereas the remaining 30 percent is distributed over the rest of the world.

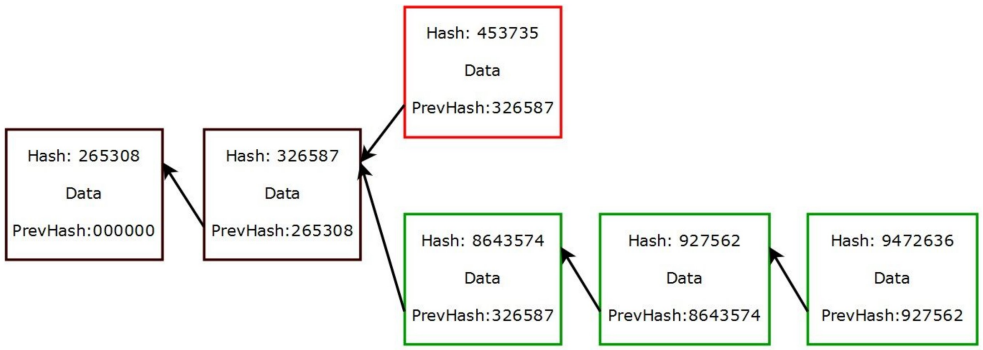

Figure 8: Schematic diagram of a blockchain. In black are the previous blocks. This is followed by a fork in red and green. The red block is wrong and is not accepted, because the green path is longer and only this one is trusted.

2.4.3. Valuation and Digital Gold

As mentioned in the previous section, the maximum possible amount of 21 million equals a fixed supply of the cryptocurrency Bitcoin. This leads to the question of whether this influences the fair market value of bitcoins. A. Meynkhard is researching this area and has concluded that it relies on the following three factors [20]:

- 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. In contrast, this amount is determined by the national banks in the case of conventional currencies. 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%, the number of newly issued coins decreases 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 subsequent bull market, which let the price drive from USD 9000 to a new all-time high greater than USD 60'000. The energy-intensive mining of Bitcoin is essential for the blockchain and thus the currency to function. Additionally, the code determines that the maximum supply is limited. This analogy to gold mining is why the broad media often refer to it as digital gold [21].

3. Methodology

The focus of this thesis can be divided into three areas. First, the aim is to find an optimal 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, it will be evaluated whether added value can be found with the help of XAI. Finally, 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 9 helps to get an overview of the individual steps followed in this section.

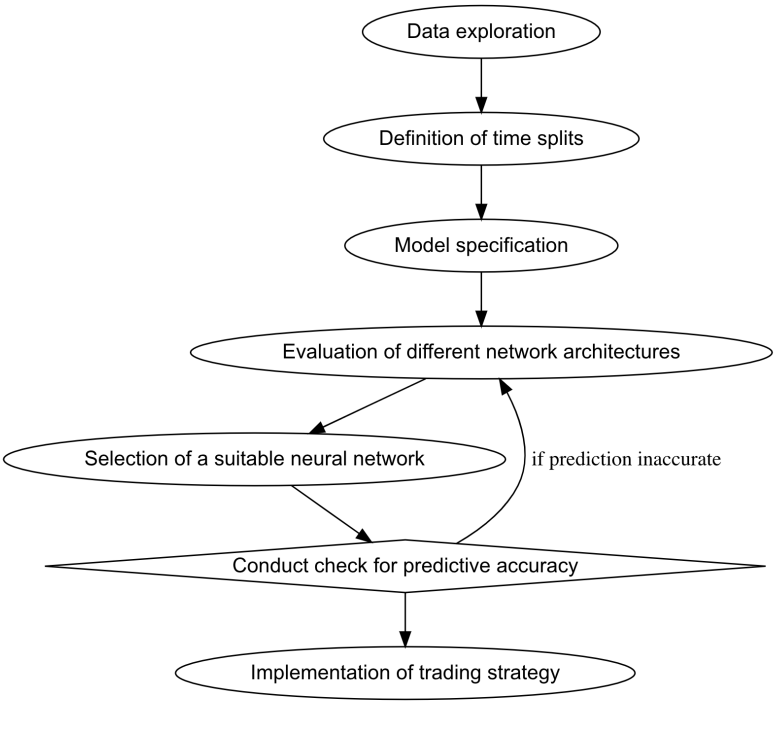

Figure 9: This flowchart illustrates an overview of the individual intermediate steps that are covered in the methodology section. After the data is explored, the details for the network archtecture are examined and compared quantitatively. Finally, aspects of explainable artificial intelligence (XAI) and traditional time series analysis are applied for the implementation of the trading strategy.

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 timestamp according to the Coordinated Universal Time (UTC).

In section 2.4, the Bitcoin price and the logarithmic price are 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 14.

$$\text{LogReturn}_t = \log(x_t) - \log(x_{t-1}) \tag{14}$$

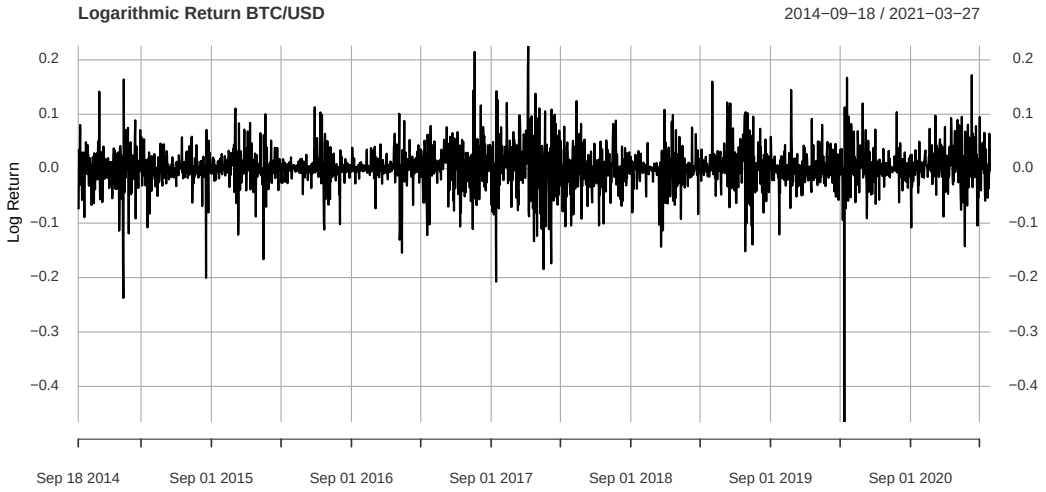

Figure 10: Logarithmic returns of BTC/USD. Data are available from fall 2014 to spring 2021. The volatility clusters typical for time series are apparent. In these phases, the log returns fluctuate strongly.

Figure 10 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 11, we can describe the dependency in these clusters. According to the ACF, the lags 6 and 10 are significant on a 5% level.

We are curious about how the log returns are distributed. Therefore, we fit a normal distribution and a Student-t distribution to the histogram of the log returns, which can be seen in figure 12. 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 well as the positive part by the Student-t distribution. The two normal distributions either over- or underestimate the values in the tails, therefore we conclude that the proposed Student-t distribution fits the data better but also not perfect. Concerning the extreme outlier discussed earlier, visible in figure 10 towards the end, the density plot makes clear how unimaginably small the probability of this extreme observation is. Although the histogram might be 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.

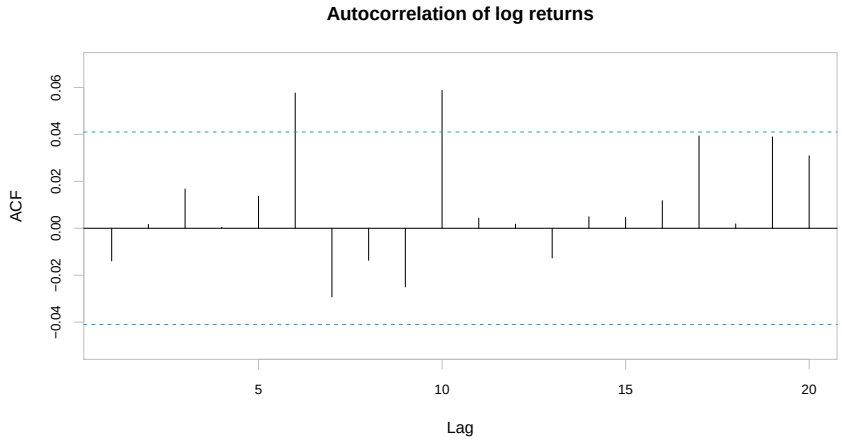

Figure 11: Autocorrelation function of BTC log returns during the entire time period from fall 2014 to spring 2021. Lag 6 and 10 are significant (values exceed the blue dotted line) and thus have an impact on possible models.

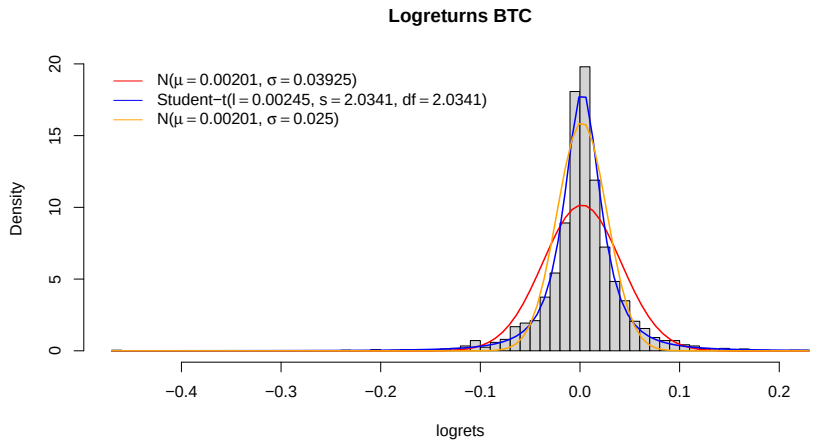

Figure 12: This histogram of BTC log returns illustrates how well it fits to different distributions. The blue t-Student distribution would fit the data better in the tails, while the yellow normal distribution would disperse a little better around the expected value. The distribution should be taken with a grain of salt, as the time dependence of the volatility is not taken into account.

3.2. Network Architecture

As mentioned in section 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:

$$\text{Comb} = \sum_{i=1}^{L} N^i \tag{15}$$

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 1,110. To respond to the challenges mentioned in section 2.1.4, 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 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 signum trading).

To ensure that the neural network with the chosen network architecture does not produce trustworthy predictions only for a specific time period, the in-sample and out-of-sample splits are examined for different time periods.

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 Bitcoin. Starting from the first of January 2020 to the 27th of March in 2021, we only consider 15 months for our data.

We do not believe that data older than one year provides useful information for predicting the following day. The reason is that the market environmnet is constantly changing and institutional investors are not yet in agreement about Bitcoin as an asset class. By optimizing our models we found that more data would offer 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 [22] 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 step length of one month we end up with 9 data splits in total. In figure 13 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.

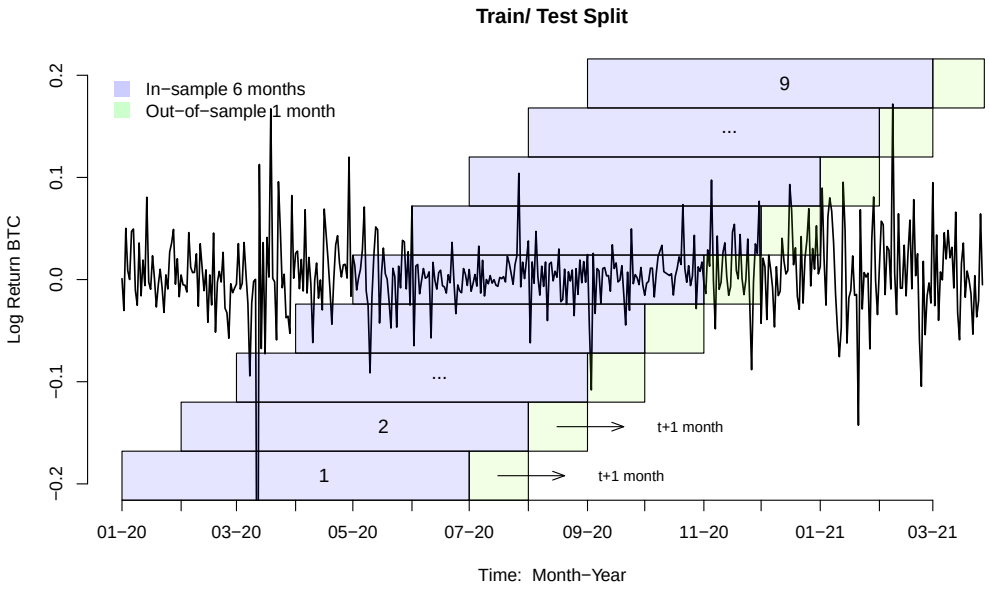

Figure 13: Nine different in- and out-of-sample splits. The 6-month training phase (in-sample) is indicated in blue and the 1-month test phase (out-of-sample) in green. The splits capture different phases of the time series. Both very turbulent phases (split 1) and rather calm phases (split 5) can be recorded.

In the time series in figure 13, 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. Defining Input-Layer

To train the feedforward neural networks, we need to specify the input layer in addition to the network architecture. For this, we take the autocorrelation function of the time period specified earlier.

In contrast to the ACF's of the entire data set shown in figure 11 where lags 6 and 10 have a high impact, here in figure 14 we see that lags 4 and 10 are significant. If one uses the ACF's as an indicator for the specification of a model, it must be considered that for other time periods also other rules apply. For modeling the entire data set, one could safely specify the input layer up to a lag of 10. Here for our subset, we specify the input layer up to a lag of 7. Lag 7 is just significant and could potentially explain an important dependency in the data. Since Bitcoin can be traded 7 days a week, including weekends, the dependency of 7 lags in the ACFs would also be explained.

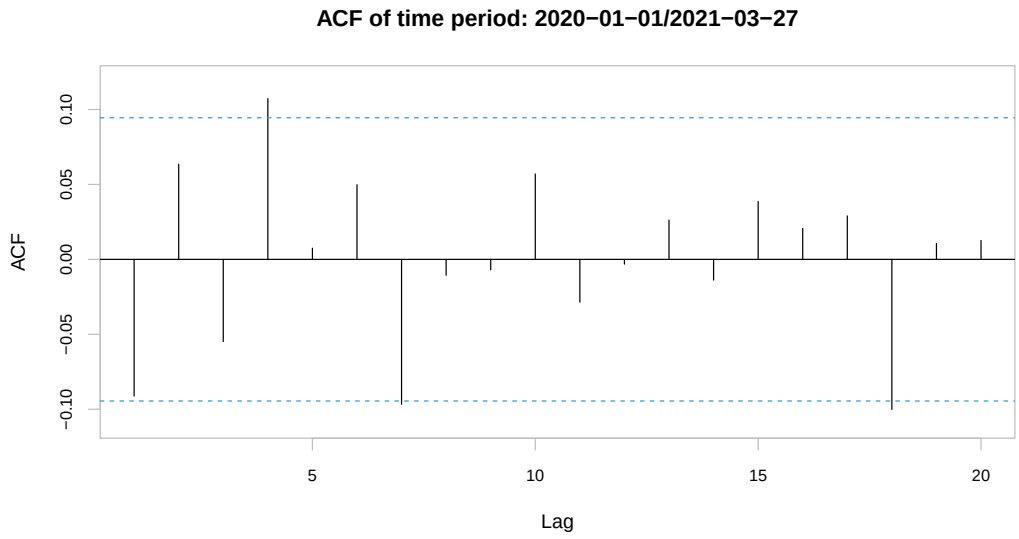

Figure 14: Autocorrelation function of our chosen time period. One can clearly see an increased influence of lags 4 and 7, not as in the ACF's of the entire data set with significant values at 6 and 10. Only late at lag 18 does the ACF show a significant value again.

For the following procedure for finding an optimal network architecture we use an input layer of 7 lags. Since we will also use the same time period later in the trading section 3.3, we will also specify the input layer with 7 lags for training these networks.

3.2.3. Neural Network Training

For training the networks, we use the function `neuralnet()`, which is part of the R-package `neuralnet`. The advantage of this function is the utilization of the backpropagation algorithm presented in section 2.1.2. Various parameters must be passed to the function. We pass the 6-month in-sample data sets defined in section 3.2.1, which are specially tailored for our applications. In the previous section, we decided on a maximum delay of 7 and the structure of the input layer can be seen in table 3. The pink-colored values indicate that they are the same values but shifted. These are the input data of the 5th test period, which would start in 2020-05-01. However, since we shift the data by 7-time units, we also lose 7 days of data, since no values are known before 2020-05-01 to fill the delays with data (of course values exist before, but so the procedure can be applied to all possible time periods). If, for example, we considered a larger number of delays, then the data set would accordingly also become smaller and smaller.

Table 3: Adapted data of the 5th test/train split for training neural networks with the R package neuralnet. Lag0 corresponds to the original data while the rest are simply the corresponding lag of the original data. The color-coded values show that these are simply shifted values.

	lag0	lag1	lag2	lag3	lag4	lag5	lag6	lag7
2020-05-08	-0.01100	0.07108	0.02908	0.01009	0.00171	-0.01019	0.01387	0.02354
2020-05-09	-0.02560	-0.01100	0.07108	0.02908	0.01009	0.00171	-0.01019	0.01387
2020-05-10	-0.09134	-0.02560	-0.01100	0.07108	0.02908	0.01009	0.00171	-0.01019
2020-05-11	-0.01782	-0.09134	-0.02560	-0.01100	0.07108	0.02908	0.01009	0.00171
2020-05-12	0.02329	-0.01782	-0.09134	-0.02560	-0.01100	0.07108	0.02908	0.01009

In addition to the training data, we pass the function an equation to specify how the training data should be handled. In our application, we aim to find a reliable prediction for the original value from the lagged data and define the equation for the function as follows:

$$lag0 \sim lag1 + lag2 + lag3 + lag4 + lag5 + lag6 + lag7$$

The function thus uses the delayed data 1 to 7 as input layer and trains a network to get as close as possible to the true values of lag0. As discussed in section 3.2, we would like to investigate the impact of network architecture on network performance. Thus, for each split, we pass the corresponding architecture from (1) to (10,10,10) to the function.

In the `neuralnet`-function, we use the sigmoid logistic function as the activation function. As mentioned in section 2.1.2, the logistic function returns values between 0 and 1. According to the article by S. Heinz, neural networks benefit when the input layers are scaled according to the activation function [23]. Solving the numerical optimization problem is therefore simplified. If you look at the logistic function in figure 2, the activity of neurons close to zero is higher than at extreme values. Extreme positive or negative values would result in 0 or 1 according to the activation function and thus no longer make a large contribution to the model. The numerical problem would be much more complex to solve than having values within an active band. Thus, we scale the data to a range of [0, 1] that is convenient for numerical optimization. The lowest value in the data set is scaled to 0, while the highest value is scaled to 1. The listed values from table 3 are transformed and can be seen in table 4.

Table 4: Scaled data from the 5th test/train split. This is the same data as in the previous table, only here the data is scaled between 0 and 1.

	lag0	lag1	lag2	lag3	lag4	lag5	lag6	lag7
2020-05-08	0.45760	0.84468	0.64664	0.55707	0.51751	0.46141	0.57489	0.62048
2020-05-09	0.38874	0.45760	0.84468	0.64664	0.55707	0.51751	0.46141	0.57489
2020-05-10	0.07870	0.38874	0.45760	0.84468	0.64664	0.55707	0.51751	0.46141
2020-05-11	0.42544	0.07870	0.38874	0.45760	0.84468	0.64664	0.55707	0.51751
2020-05-12	0.61931	0.42544	0.07870	0.38874	0.45760	0.84468	0.64664	0.55707

The result from the `neuralnet`-function is the prediction of the network for the in-sample test data set, i.e. the data with the lag0. To compare the performance in the in-sample area with other networks, the results computed from the scaled data by the network are transformed back to the original form.

For the out-of-sample area, the `predict` function from the same package is used. One passes the trained network (optimized weights and biases) with the scaled out-of-sample data (lag 1 to 7). This result is also transformed back and compared to the original data (lag0) to analyze the performance of the neural network, ouf-of-sample.

3.2.4. 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 13 the middle one).

The plot in figure 15 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.

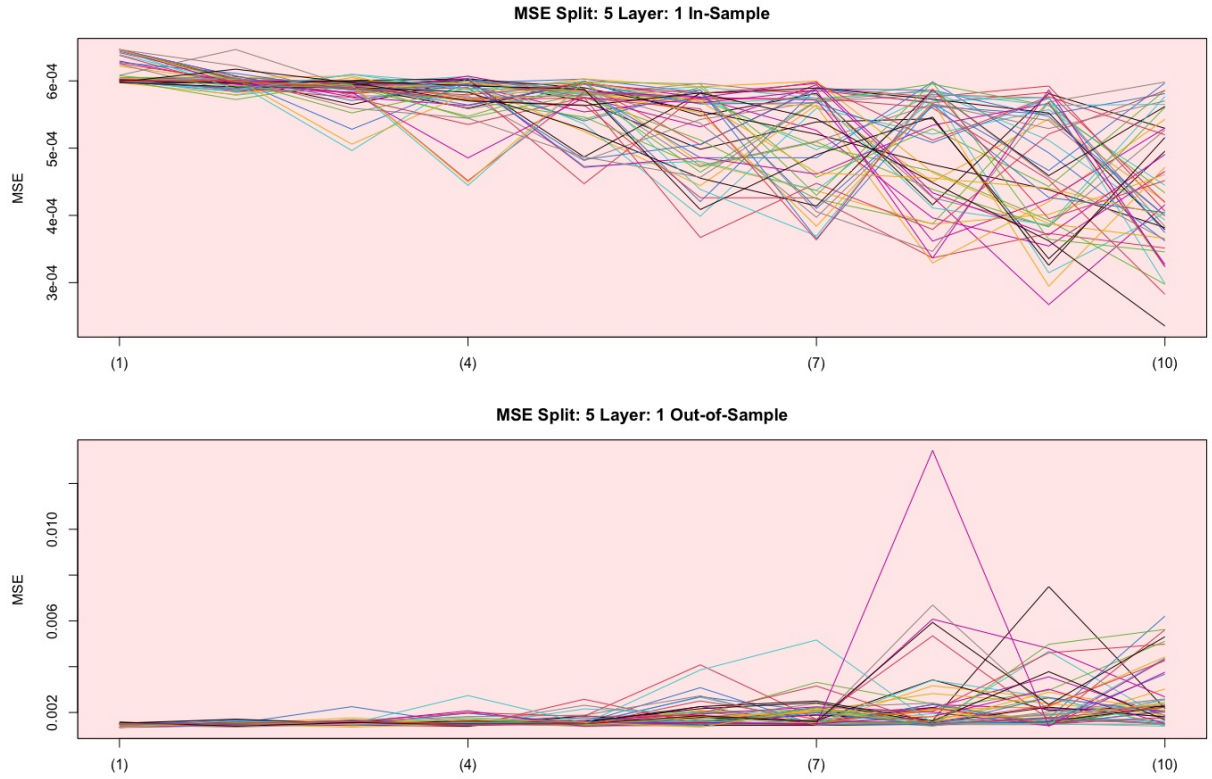

Figure 15: In- and out-of-sample MSE of the 5th time split. Only the networks with only one layer are shown here. Simplest network has 1 neuron (leftmost) and the most complex one has 10 neurons (rightmost), therefore we have the MSE's of 10 different neural networks. The in-sample MSE's tend to decrease, while the out-of-sample MSE's move the opposite direction.

What is already noticeable here is that with an 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 16, 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 15, the values for the MSE also fluctuate more and more with increasing complexity. Small in-sample MSE for more complex networks leads to rather high out-of-sample MSE. This leads us to the previously mentioned challenges in section 2.1.4.1, and that too many estimated parameters can lead to overfitting of the network.

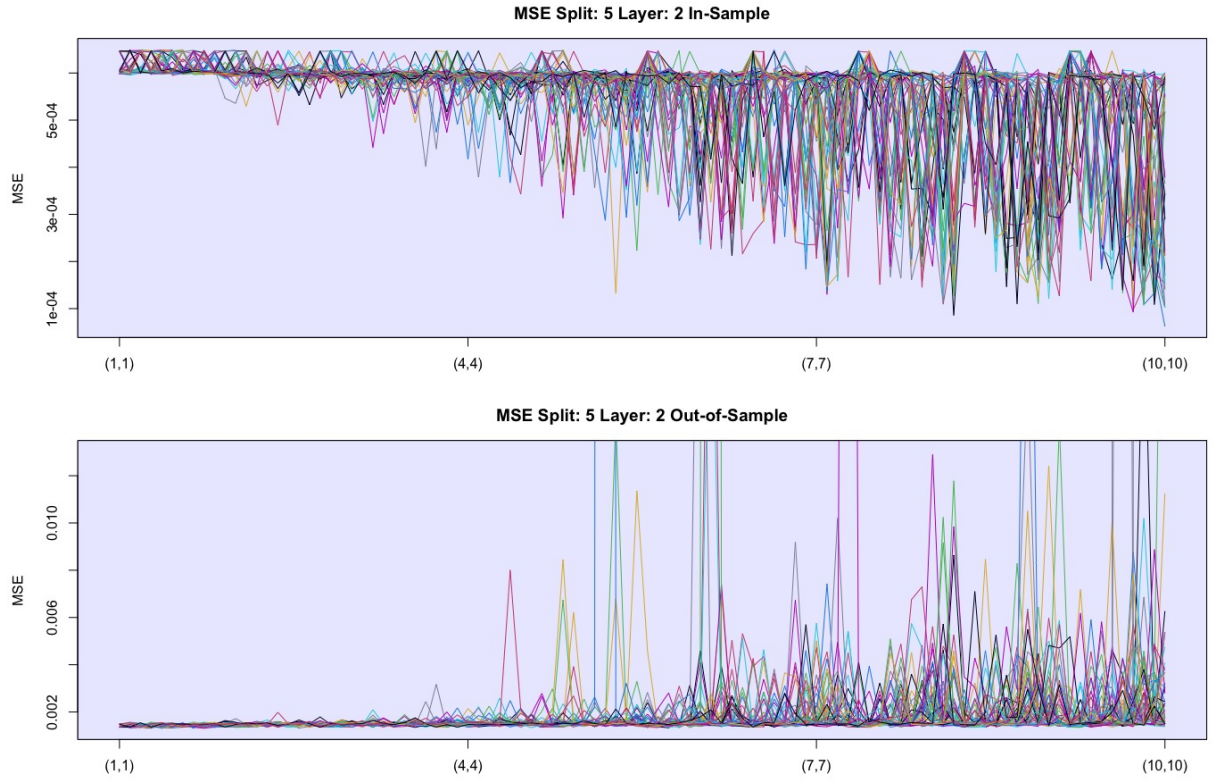

Figure 16: In- and out-of-sample MSE of the 5th split. Shown here are only the networks with two layers. The simplest network has an architecture of 1 neuron per layer (leftmost) and the most complex has 10 neurons per layer (rightmost), therefore we have the MSE's of 100 different neural networks. As in the previous graph, in-sample MSE's tend to decrease while out-of-sample MSE's tend to increase.

Looking at the out-of-sample MSE's in the graph below in figure 16, 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 17, 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.

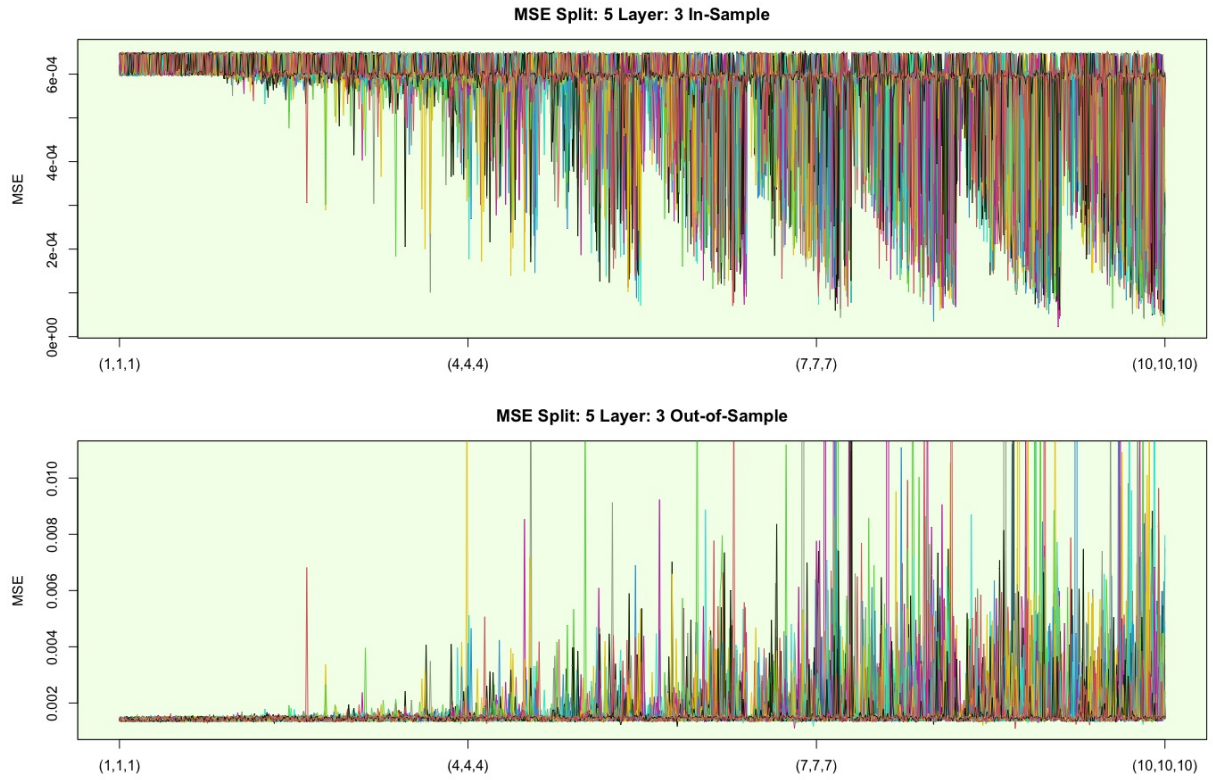

Figure 17: In- and out-of-sample MSE of the 5th split. Shown here are only the networks with 3 layers. Simplest network has an architecture of 1 neuron per layer (leftmost) and the most complex has 10 neurons per layer (rightmost), therefore we have the MSE's of 1000 different neural networks. Same pattern as for 1 layer and 2 layers: The in-sample the MSE's decreases, while the out-of-sample MSE's increase.

Figure 18 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 more towards the bottom as the complexity of the architecture increases. 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 the 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 section. Our primary goal is to compare the performances of different network architectures using MSE to find the optimal network that performs well for 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.

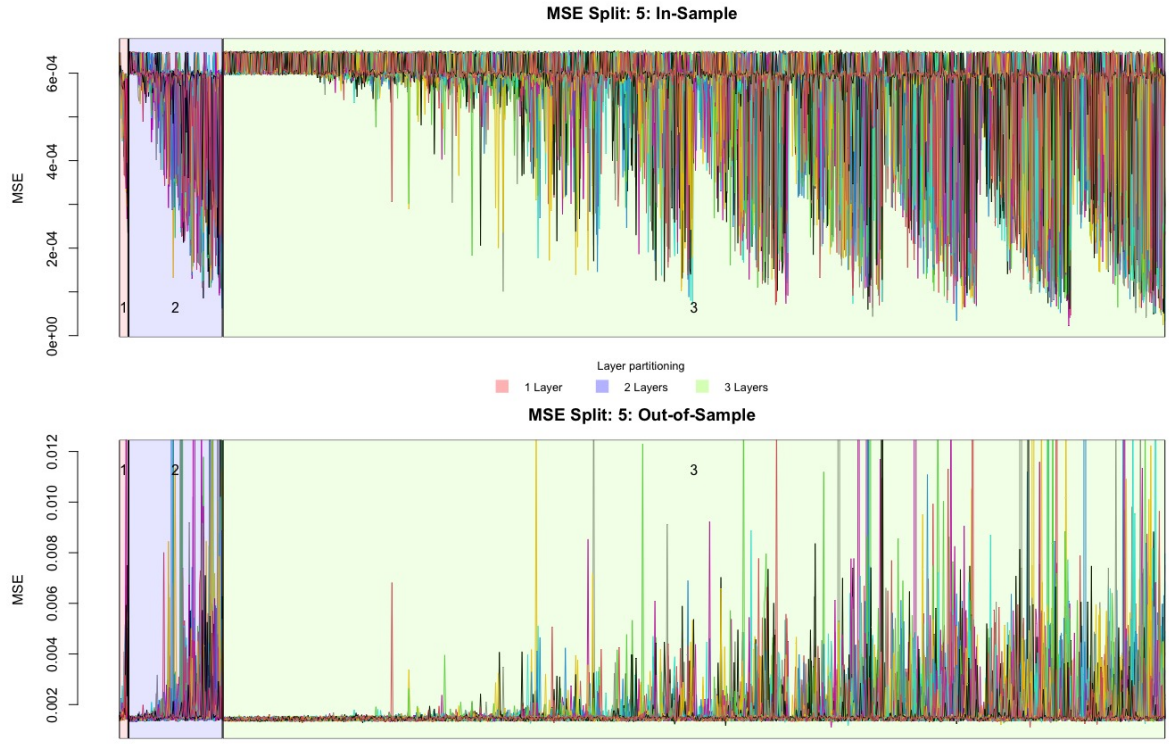

Figure 18: In- and out-of-sample MSE of the 5th split. The 3 previous graphs are summarized here and represents an overview of all 3 layers, which results in a total of 1110 neuron-layer combinations. Again, the inverse relationship is evident. The more complex a network, the smaller the in-sample MSE's become. At the same time, the out-of-sample MSE's increase.

What has not yet been studied is the dependence of the neural network's behavior on the different window splits we defined in figure 13. 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 across 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 18 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 across all time splits are plotted in figure 19. 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 across all time splits.

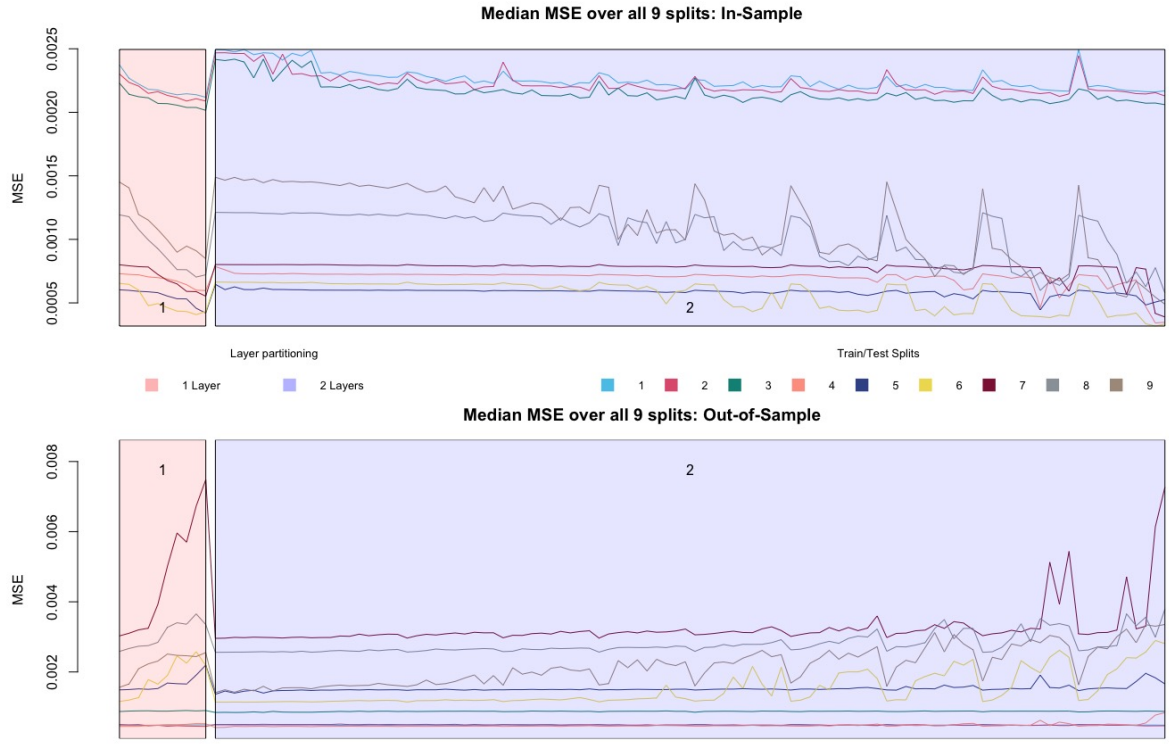

Figure 19: We take the median of all MSE's for a given network architecture. The 9 splits are visualized by different colors. For each network architecture, 50 neural networks were trained (55'500 models per time split). The nine colors illustrate how these models behave in the different time splits. Time periods with a strong trend (for example, the dark red line, split 7) lead to the same structure as already noted in the previous graphs. Again, the inverse relationship is evident. The more complex a network, the smaller the in-sample MSE's become. At the same time, the out-of-sample MSE's increase.

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 logarithmic prices can be found in the appendix starting with figure 33. 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 the last comparison, we visualize in figure 20 how well the different network architectures behave in sign trading. This is 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 forecast, 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.

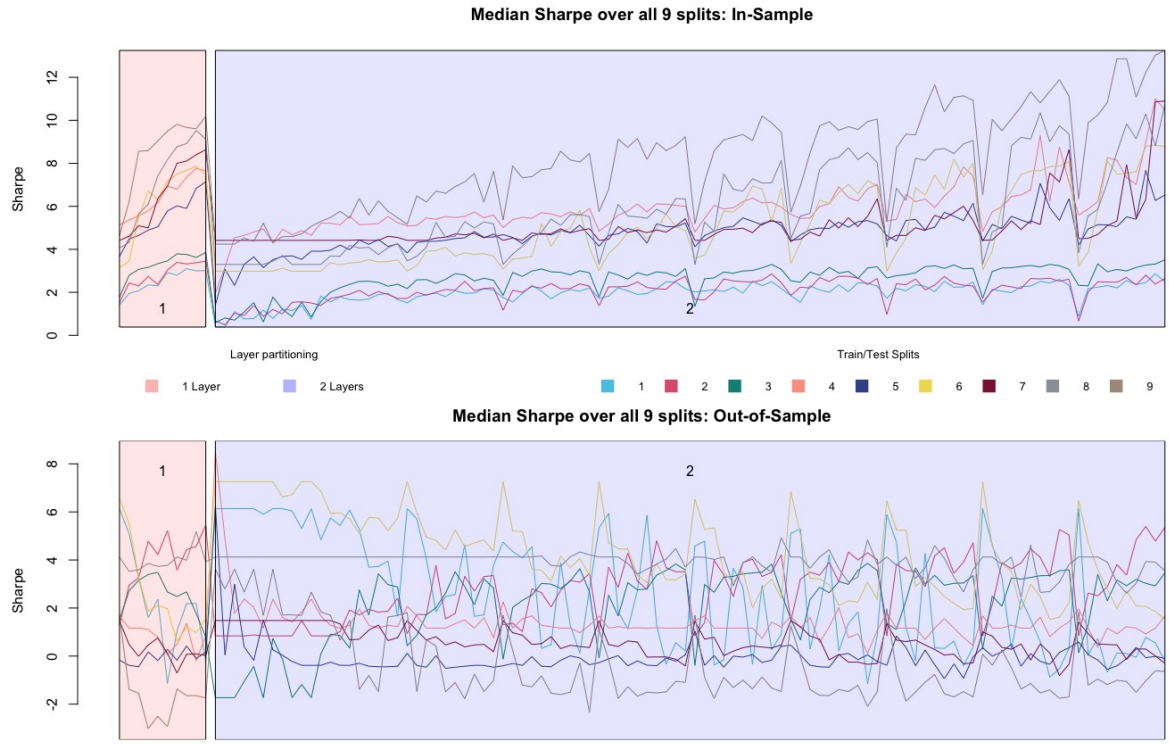

Figure 20: We take the median of all Sharpe ratios for a given network architecture. The 9 splits are visualized by different colors. The in-sample Sharpe ratios increase (upper plot) with increasing complexity. The out-of-sample Sharpe ratios tend to decrease with increasing complexity. In comparison to the previous plots using the MSE, the effect is much smaller here.

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, a one-time investment in the S&P 500 ten years ago would have resulted in a Sharpe ratio of 0.83 [24].

3.2.5. Model Selection

With the above procedure, we have been able to identify the characteristics of neural networks. In time series applications, complex networks lead to overfitting. We could detect an inverse relationship between in- and out-of-sample MSE. We trained many networks, yet no distinct optimal neuron-layer combination can be identified. Determining the architecture based on the minimum MSE would not yield good results. There would be too great a risk that the chosen architecture would be too complex and lead to overfitting. Additionally, the chosen architecture should lead to trustworthy predictions over different time periods (in our example over the 9 splits).

For the rest of the procedure, we decide to use a neural network architecture of (7,7). According to the 2-layer structure visualized in figure 16, the architecture we chose looks very balanced. In-sample, the MSE is rather in a more optimal range (smaller MSE) and out-of-sample, any overfitting is rather minimal compared to even more complex networks. In the attachment in figure 32, you can see the average MSE across all 9 splits. The neuron layer architecture of (7,7) is exactly in the range where the in-sample MSE decreases and out-of-sample increases. The selected network lies exactly between the volatile networks, thus this architecture does not seem to be as prone to overfitting as the one next to it. The (7,7) network is not too complex, tends to have a good in-sample MSE and the out-of-sample MSE does not tend to overfit as much.

3.2.6. Benchmark

The trained networks and applied methods are compared with each other using a benchmark. For this purpose, we use the buy-and-hold, which is a long-term passive investment strategy. Investing with this strategy often leads to better returns in the long run than active trading [25]. This strategy is hard to beat, especially when the market is trending upwards. In such a trend, an active strategy would hardly recommend anything other than buy-and-hold.

We, therefore, compare our developed active strategies with the cumulative log returns. Our benchmark includes all green out-of-sample returns from figure 13 cumulated.

3.3. Trading Strategy

This section describes the trading strategy in more detail. In basic terms, the findings of the previous sections 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 21 shows a broad overview of how the different factors are combined to generate the final trading signal.

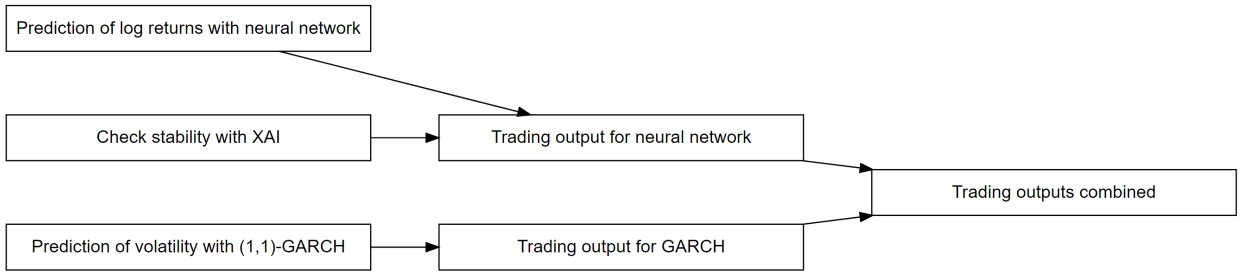

Figure 21: This flowchart illustrates an overview of the trading strategy applied in this section. After finding the best network architecture for accurate forecasts of log returns, we check the stability of the neural network with XAI. The GARCH-model addresses the phenomenon of volatility clusters. Finally, all these information are combined in a trading strategy in order to be compared with our benchmark consisting of buy-and-hold.

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 section 3.2.4. 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 sections 2.4.1 and 2.4.3. That is why buy-and-hold is frequently used as a comparison.

Before we trade the returns, we first need to define the environment and make some assumptions. Normally, trading costs are crucial in high-frequency trading. Since the fee structure for cryptocurrencies is more attractive than for stocks, transaction costs are waived for the sake of simplicity. Further, we assume the possibility of entering short positions for BTC. These assumptions allow us to either stay in the market (signal = 1), to exit the market (signal = 0) or to sell tomorrow's return (signal = -1). In the latter two positions, negative returns can be avoided, or we can even profit from the downward movement. By exiting the market with signal 0, there are several opportunities to invest the money elsewhere or just stay out of the market and therefore eliminate market exposure.

3.3.1. Trading with Neural Networks and LPD

The following consecutive sections 3.3.1.1 and 3.3.1.2 complement each other. We describe, how the neural network and LPD (theory explained in section 2.2.2) is used to create trading signals.

3.3.1.1. Neural Network

In 3.2., we use an average of only 50 realizations of feedforward networks to reduce computation time. However, 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.

For a better understanding, we will take a closer look at how we handle the predictions in order to generate a trading signal. Assuming we calculate an amount of N_{total}=100 networks in total, try to predict just 1 day, and use the sign of the forecast as a trading signal. If 49 nets predict a positive forecast and 51 a negative, we end up with a negative trading signal, which is actually decided by just one net. This means that the resulting trading signal occurs randomly. A positive trading signal that results from 60 positive predictions and 40 negative ones may be more accurate. Therefore, the following a function is introduced that takes this aspect of majority voting into account.

Let $\underline{\hat{p}}$ be the predicted values $\{\hat{p}_1 \dots \hat{p}_{N_{total}}\}$ of the neural networks in a certain time t. With the help of function f seen in equation 16, we derive a trading signal for every t, while we pay attention to majority decision. The parameter κ influences how large the percentage of votes must be for a vote to win. For further evaluation of neural networks, κ can be used as an optimizing factor.

$$f(\underline{\hat{p}}) = \begin{cases} 0 & \text{if } \frac{1}{N_{total}} \sum_{i=1}^{N_{total}} sign(\hat{p}_i) < \kappa \\ sign(\sum_{i=1}^{N_{total}} sign(\hat{p}_i)) & \text{else} \end{cases} \tag{16}$$

N_{total} = Total number of neural networks

κ = Ratio of majority decision

$\hat{p}_m$ = Predicted value from neural network, $m \in \{1 \dots N_{total}\}$

One can observe in function f how the uncertain decisions result in a 0 signal. Another aspect that could be included is position sizing and to adjust the signal depending on the size of κ. In scope of this thesis, these aspects are omitted due to resource constraints.

3.3.1.2. LPD Signal

To explain the following as understandable as possible, we recall chapter 2.2.2 where the discrete derivatives of each input layer lag_j were calculated with respect to the forecast at each time step. This procedure is now applied to the in-sample as well as the out-of-sample area. From the in-sample, the mean and the standard deviation are calculated for each lag_j so that we can determine which values are considered extreme. These are now used for the out-of-sample area. An example with out-of-sample 2021-01-01 - 2021-02-08 clarifies these procedure:

Figure 22 represents the 7 lags, when looking at lag 4, the top line shows the previously calculated mean value from the in-sample (black) and the in-sample standard deviation upwards and downwards (dashed red). For the visualization the lines are only drawn for one lag. Let us take a look at the time step 2021-01-25, where we can see that the pink line exceeds the standard deviation. Next, we count how many other lags also overshoot their standard deviation in this time step. This procedure is described by formula 20. We recognize that 4 lags have exceeded their standard deviation on 2021-01-25 and therefore indicate an unstable state of the neural network. In this example, we state that if 4 or more lags exceed the upper limit the final signal is 0. If the the number is between 2 and 4, we assign the signal to 0.5. For our example date 2021-01-25 the signal would be 0. Correctly concluded, we get a trading signal for each date.

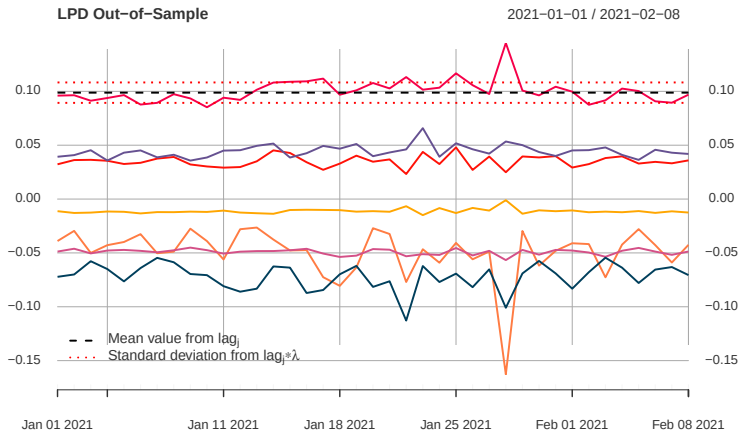

Figure 22: The LPD is used to generate the trading signal. Higher volatility indicates that the neural network may provide unreliable outputs. For the top LPD time series, the mean value and the upper and lower decision bands are plotted. Each time this LPD exceeds the upper or lower decision band, a signal is realized for this lag. In the most extreme phases it can happen that several lags exceed the decision bands and thus generate several signals. The number of all exceedances will be considered in the next plot in order to generate the XAI trading signal.

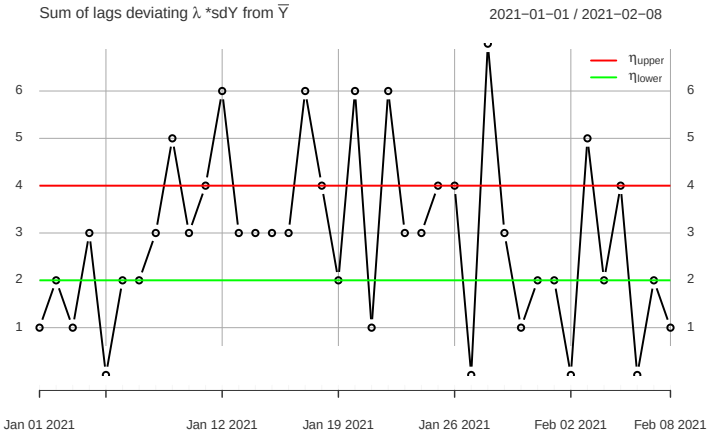

Figure 23: The black line shows how many lags exceed the threshold defined above at one point in time. The more extreme lags are counted, the more unstable the neural network is. Red is the upper decision limit and the green is the lower one. Values below the green line indicate realiability and lead to a signal = 1. Values above the red line indicate instability and lead to a signal = 0. Values between the red and the green line lead to a signal = 0.5.

To use this behavior for optimizing our model, a more general approach is required. The standard deviation is therefore scaled with parameter λ. With the help of formula 19, it is decided in every lag_j whether the predicted LPD $\hat{\Delta}_{j,t}$ exceeds $\mu_j \pm \lambda\sigma_j$.

With function g seen in equation 19, we count the lags which exceed their bands, the output of g is between 0 and lag_n. In equation 20, g is further applied to generate a signal. With η_{lower} (green) and η_{upper} (red), it is decided which final output for the time step t should be assigned. In the last part of section 3.1., we observed that larger negative returns are more likely than positive ones, therefore the proposed signal is 0. For values between η_{lower} and η_{upper} we propose a signal 0.5. Function 20 is then applied to every time step t, therefore the final LPD signal is computed.

$$\mu_j = \frac{1}{n_{in}} \sum_{t=1}^{n_{in}} \Delta_{j,t} \tag{17}$$

$$\sigma_j = \sqrt{\frac{1}{n_{in}} \sum_{t=1}^{n_{in}} (\mu_j - \Delta_{j,t})^2} \tag{18}$$

$$g(\hat{\Delta}_t) = \sum_{j=1}^{lag_n} \Big((\hat{\Delta}_{j,t} > \mu_j - \lambda\sigma_j) \vee (\hat{\Delta}_{j,t} < \mu_j + \lambda\sigma_j) \Big) \tag{19}$$

$$\text{signal}(\hat{\Delta}_t) = \begin{cases} 0.5 & \text{if }, \eta_{lower} < g(\hat{\Delta}_t) \leq \eta_{upper} \\ 0 & \text{if }, \eta_{upper} < g(\hat{\Delta}_t) \\ 1 & \text{else} \end{cases} \tag{20}$$

$\Delta_{j,t}$ = Matrix of in-sample LPD $\{\underline{\Delta}_1 ... \underline{\Delta}_{n_{in}}\}$ observed for every t in insample

$\hat{\Delta}_{j,t}$ = Matrix of out-of-sample LPD $\{\underline{\hat{\Delta}}_1 ... \underline{\hat{\Delta}}_{n_{out}}\}$ predicted for every t in out of sample

λ = Scaling parameter for standard deviation

lag_n = Maximum number of lags

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

n_{in}, n_{out} = Maximum time in-sample / out-of-sample

g = Function of lags exceeding band, output $\in \{0 ... lag_n\}$

$signal$ = Function of trading signal

μ_j = Arithmetic mean of in-sample LPD for every lag_j

σ_j = Standard deviation of in-sample LPD for every lag_j

3.3.2. 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 42.

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 proposed 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:

$$\text{Signal}_{GARCH}(\hat{\sigma}_t) = \begin{cases} 0 & \text{if}, \hat{\sigma}_t \geq 1.64 * \sigma_{historical} \\ 1 & \text{else} \end{cases} \tag{21}$$

$\hat{\sigma}_t$ = predicted volatility for every t

$\sigma_{historical}$ = historical volatility

In other words, the function 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 24: 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.

Figure 24 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 sign strategy misjudges the situation at key points in time and thus hurts the overall performance. Only during the Covid-19 crash in March 2020, the GARCH sign 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.

3.3.3. Neural Network and LPD Trading

In section 3.2.5 we have decided to fix the architecture to 2 layers with 7 neurons each, all the 9 splits which were introduced in section 3.2.1, are now used for trading. In every 6 month-insample split a new model is estimated with the schema from 3.3.1. The out-of-sample signals of the neural network and LPD from every split are summarized.

Our previous findings from the GARCH model were not very promising, nevertheless, we try to use those signals with the neural net output and the signals from LPD. That leaves us with a total of three signals for every timestep:

$\hat{NN}_t$ = Neural net signals $\in \{-1, 0, 1\}$

$\hat{LPD}_t$ = LPD signals $\in \{0, 0.5, 1\}$

$\hat{GARCH}_t$ = GARCH signals $\in \{0, 1\}$ from section 3.3.2

With the previously established architecture, all combinations of the signals for different values for the parameters: λ, $\eta_{lower}, \eta_{upper}$ and κ are computed and the performance is compared to each other as well as to the benchmark from 3.2.6. To find a balance between precision and computation time, we found N_{total}= 1000 Neural networks in every split, should be sufficient for the evaluation.

The most promising performance is found with $\lambda = 1$, $\eta_{lower} > \eta_{upper}$ (no 0.5 signals were used) $\eta_{upper} = 3$ and $\kappa = 0.2$. In figure 25 different combinations are visualized.

- NN is simply trading the $LogReturn_t$ with $\hat{NN}_t$
- NN+LPD is the product of NN and $\hat{LPD}_t$
- NN+LPD+GARCH takes into account the GARCH signals $\hat{GARCH}_t$ by multiplying them with NN+LPD.

By investigating the plot, it seems that some combinations are performing better than the others. The GARCH application appears to worsen the performance. The ligthblue and the yellow line catches our eye, their performance seems to be better than the rest and as we can see, those are combinations of NN+LPD. Notice that signals with zeroes are not yet invested elsewhere, therefore they represent staying out of the market, mitigate risk and wait for a new signal to come.

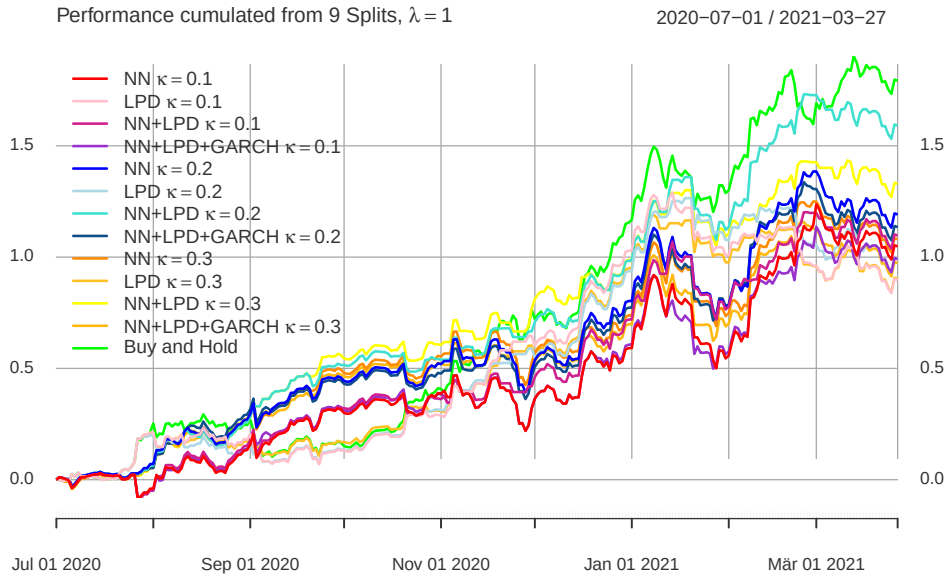

Figure 25: Different combinations of trading signals are compared to each other. The best performing one is buy-and-hold (green). It is closely followed by the light blue NN+LPD combination with a κ of 0.2. The behavior of the individual rules is often very similar, few different decisions can influence the performance. Especially at the end of July, beginning of August the rules behave differently, but then usually behave very similarly.

When we have a closer look at the Sharpe ratio in 26, our findings from before are confirmed. The NN+LPD with $\kappa = 0.2$ has even a better Sharpe ratio than buy and hold. But another observation is seemingly more interesting, the LPD is always better than only NN. By investigating the other calculations we have done (plots found in Attachement), we find the same pattern, therefore we conclude that LPD brings us a real benefit for this particular application.

Figure 27 gives an insight on how the individual combinations performed in each of the 9 splits. The NN+LPD with $\kappa = 0.2$ is predominantly better in the first three splits, in the middle section its worse and towards the end the Sharpe is again better than buy and hold. But we have to take this plot with a grain of salt because these splits are only one month out of sample, meaning the Sharpe Ratio is sensitive to outliers, this effect is even stronger when there are few datapoints present.

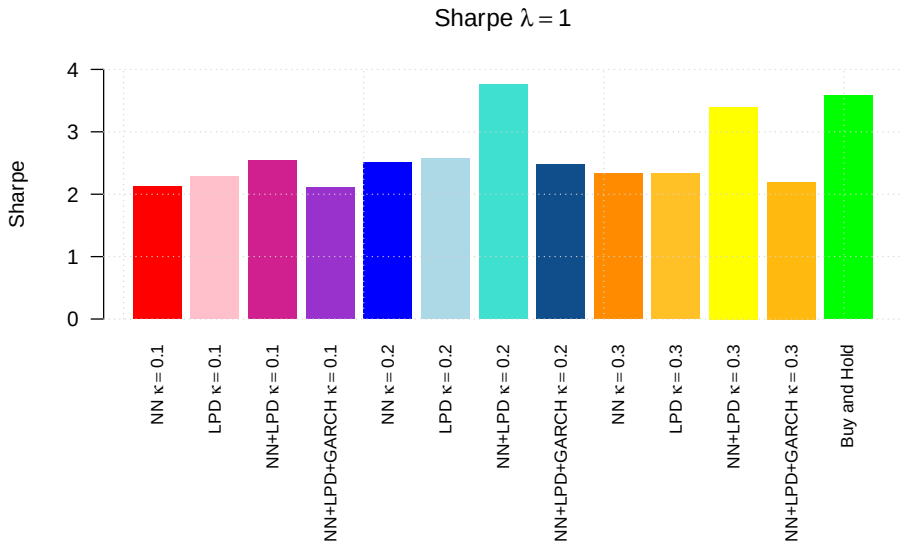

Figure 26: Sharpe ratios of different signaling combinations. Based on the performance measure Sharpe ratios, the LPDs lead to a better performance. According to the Sharpe ratio, the performance of the NN+LPD signaling rule with a κ of 0.2, is better than Buy and Hold.

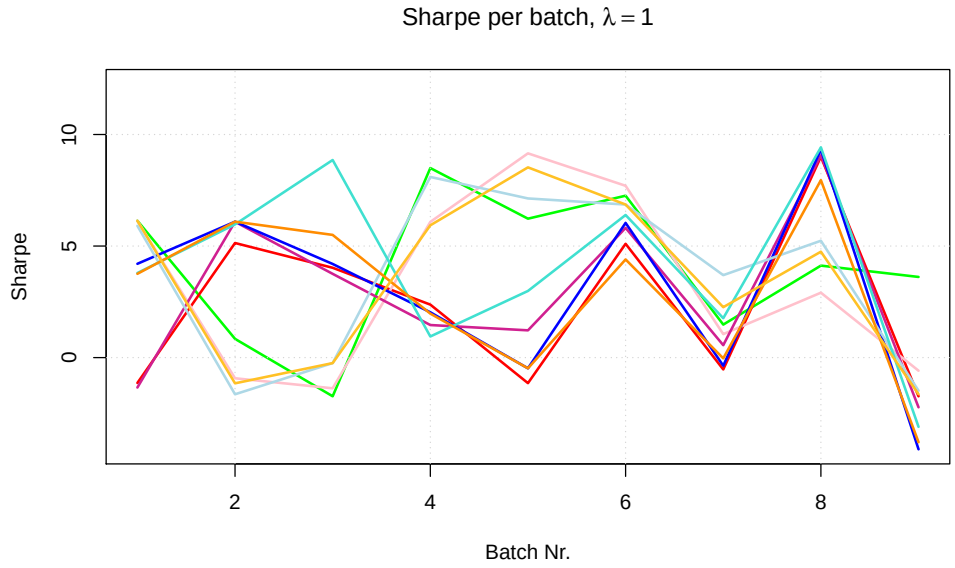

Figure 27: Sharpe ratios from the same different signaling combinations as in the last two graphs, but each for the 9 different time splits. Certain combinations have very similar decisions and thus lead here also to similar trajectories of the Sharpe ratios.

3.3.3. Adding Ether

In the performance plot 25, we have still left out what to do with 0 signals. One opportunity as in 3.3 shortly mentioned, is to invest the money in another asset. By reason of staying in the same asset class we choose Ether [29] to invest in. The coin Ether from Ethereum is with $61.48B [30] the second largest cryptocurrency according to market capitalisation.

According to [31] the correlation between the two assets is 0.91. This is not a braking factor to us, because the 0 signal of the LPD is just indicating that something is changing in the data or the neural net is not stable, thus we can not exactly know what the 0 signal means.

The structure Ether is very similar to Bitcoin [32]. The series is imported in the same way as BTC e.g. the log returns are derived.

That said, we use a simple rule to integrate Ether. If the signal of the combination NN+LPD is 0 then we go long on Ether. As soon as the signal from LPD changes back to 1 or -1, Ether is sold and the Bitcoin position is entered again.

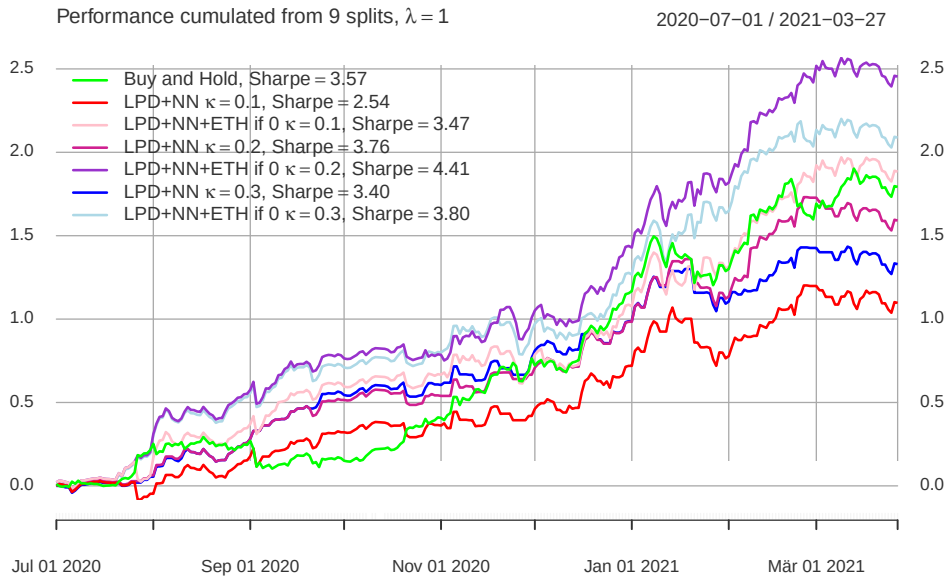

Figure 28: As described in this section, we investigate whether the addition of an alternative investment can lead to better performance. The trading signals with different kappas resulting from the neural networks and LPD (LPD+NN) are supplemented with occasional investments in ETH. The combination LPD+NN+ETH with a kappa of 0.2, clearly performs best here. Overall, even 3 combinations are better than buy and hold.

In figure 28 the NN+LPD's from figure 25 and the adding of Ether is visualized. In the lower part, we can observe the signals which are 0 in the NN+LPD model. These zeroes are now complemented with the log returns from Ether.

The one performance with NN+LPD+ETH with $\kappa = 20\%$ seems promising. Also for the other two NN+LPD the adding of Ether seems benefital.

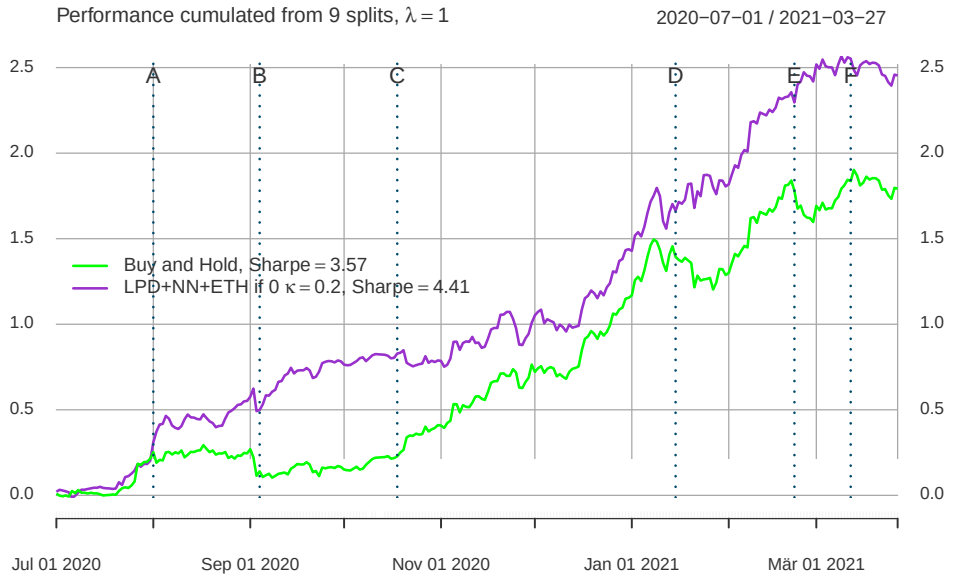

Figure 29: Visualization of the best performing signaling and the benchmark buy and hold. One can clearly see that as of August 2020, the purple signaling strongly contrasts with the green benchmark. There are some decisions which lead to a better performance than simple buy and hold.

For a closer analysis, let us have a look at the cumulated log-returns of the best performing model vs. buy-and-hold in Figure 29. The following analysis is related to the violet chart and starts from left to right, beginning with the letter **A**. Immediately after the line by letter **A**, several decisions differ from buy-and-hold and force a shift to the series. In the letter **B**, we observe similar behavior for a short period. The unsatisfying trend change in the letter **C** is stopped around the first of November. Thereafter unto the point in letter **D** the decisions of the model behave quite similar to the green line. Right after **D** the net handles the negative trend very well and climbs further up. The same accounts for the letter **E**, where the trend once more goes in the opposite direction than the benchmark. Shortly after **F** the model does exactly the same as buy-and-hold.

Let us have a simple theoretical comparison example under the assumptions in 3.3 at last.

With summarizing the logreturns from 14 with Formula 22, one can calculate the value S_{tn} which we end up with, when an Amount S_{t0} is invested at the beginning.

$$\begin{aligned} S_{tn} &= S_{t0} \prod_{k=1}^{t_n} e^{LogReturn_k} \\ &= S_{t0} e^{\sum_{k=1}^{t_n} LogReturn_k} \end{aligned} \tag{22}$$

Lets say, we invest 1000 US-Dollars at the beginning of the timeperiod in 29. With buy-and-hold we would end up at 6003.25 US-Dollars by the end of March. However if we invest the 1000 USD in the best performing strategy we will end up with **11'633.70** USD, that would be as nearly as double the amount of buy-and-hold. Comparing the Sharperatios **4.41** to 3.57 of the LogReturns, the used strategy does also significantly better.

4. Results

In this section, we summarize our results from the individual parts.

Regarding the network architecture, we find that it is not worthwhile to use a feedforward network with as many layers and neurons as possible, since this leads to overfitting.

Furthermore, we found out, that if the correlation between in-sample MSE and out-of-sample MSE is positive, this does not necessarily mean that this also accounts for the performance. From this finding, we can conclude that a less complex network in our case leads to equal or better results.

For our purpose, a two-layer with each 7 neuron net is therefore sufficient.

In terms of trading, we tested all combinations of three approaches: Neural net forecast, stability check with LPD, and volatility predictions with GARCH.

After forecasting the volatility with a rolling window GARCH we conclude: the performance is not satisfying enough. The combination of the GARCH volatility with neural net and LPD also brings no further benefit.

The combination of neural net and the LPD approach with parameter $\lambda = 1$ and $\kappa = 20\%$ lead to a good performance.

We also notice the improvement in the performance of the neural net through LPD in almost all cases.

Finally, with the addition of the asset ether and the use of the 0 signals, we have outperformed the benchmark buy and hold.

5. Conclusion

In this bachelor thesis, we deal with three different fields. First, the network architecture of a neural network is investigated, then the model is analyzed using an XAI approach. In the last step, the information found in this paper and general time series analysis is combined to form a trading strategy. The methodology is based on the guiding questions defined in chapter 1.:

1) What influence does the selection of the network architecture have on the quality of predictions? What influence can be found concerning the Sharpe ratio of a simple trading strategy?

Pursuing the first guiding question, all possible network architecture combinations between one layer with one neuron and three layers with ten neurons each are quantitatively compared. The plots of the error function MSE in chapter 3.2.4. show that a complex model is more prone to overfitting. The Sharpe ratio is used to assess trading performance. There it can be seen that the inverse relationship between the in-sample and out-of-sample indicates overfitting characteristics. In conclusion, no rule of thumb can be found that works well without testing and comparing. The model should be complex enough for the sake of accuracy, but should not have overfitting. Ultimately, it seems to make sense that the number of neurons should be in the range of the number of inputs.

2) What is the added value of enhancing this neural network with aspects of explainable artificial intelligence (XAI)? Again, the impact on trading performance will be investigated.

Of the methods presented, the linear parameter data approach (LPD) turns out to be the most useful for financial time series. Important reasons for this are that the data should remain chronologically ordered and the dependency structure should remain in place. The derivatives after the intercept and the weights reveal important points. On one hand, there is an analogy to linear regression, which suggests that some explanatory variables are more important than others at a point in time. Furthermore, analogous to the autocorrelation of the bitcoin log returns, it shows which lags are important and should be fed into the neural network. LPD cannot explain the operation of the network, the role of weights and error terms, and the development of the backpropagation algorithm. In summary, the derivations do a good job of revealing when the neural network is unstable, and thus inaccurate outputs are to be expected.

3) How can the acquired information about neural networks, XAI, and general time series analysis be used to define an efficient trading strategy?

The selected neural network provides the predictions of the log return of the next day. Referring to the central limit theorem, 1000 neural networks are optimized per prediction. With the help of a majority decision, which depends on a parameter β, the definite signal is derived from the neuronal network. At the same time, LPD shows in which time periods the model is unstable and therefore the output should be taken with caution. Extreme events of the individual derivatives in the LPD plot are combined to generate an additional trading signal. With the addition of a GARCH model, future volatility is estimated and if estimates are high, the market is exited as a precaution. Combining this information, it is noticeable that over the period tested, the buy-and-hold benchmark beats all methods. However, the addition of the signals from the LPD lead to better returns than the same model without LPD. Therefore, for the tested period, the addition of XAI seems to offer an effective added value.

Investing is about having your money invested in the market. Therefore, the thought process is continued by making an alternative investment in Ether during phases where the previous methods recommend going out of the market. Surprisingly, such a strategy leads to an outperformance of buy-and-hold for the chosen period.

5.1. Outlook

There are thousand different ways, to implement a trading strategy, due to the time limitation, this thesis could only investigate a small part of the applications of feed forward nets in trading. Recent events in the cryptocurrency market, which just occured at the end of the writing phase of this thesis, made us curios how the model would perform with the recent drawdown. The model handled the drawdown phase properly as seen in Figure 30. The massive loss in the beginning of may was unfortunately even amplified but shortly after damped and the trend reversed, comparing to buy-and-hold.

The GARCH model is only one of the many classical approaches for timeseries. Based on this thesis other tools such as different moving averages or ARMA-APARCH Models in combination with feed forward nets. Also interesting might be the inputsmoothing of the neural net with an MA.

The Explainability aspect of neural networks is still in development. Further work could try to improve the application of XAI in the context of financial timeseries.

Future work based on this thesis, also could investigate the real time application test of the model for trading. Since this thesis is only valid under the assumptions taken, a practical application needs to deal with these issues.

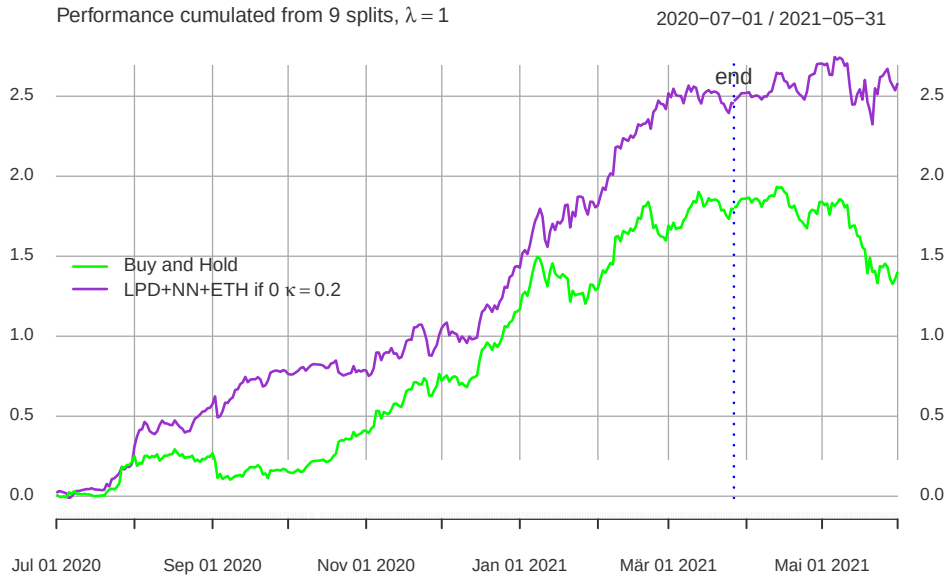

Figure 30: Data collected at the end of the writing phase of this thesis

References

[1] A. F. Bariviera, M. J. Basgall, W. Hasperué, and M. Naiouf, "Some stylized facts of the bitcoin market," *Physica A: Statistical Mechanics and its Applications*, vol. 484, pp. 82–90, 2017, doi: https://doi.org/10.1016/j.physa.2017.04.159.

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

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

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

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

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

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

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

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

[10] 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).

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

[12] 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).

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

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

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

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

[17] 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).

[18] 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).

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

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

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

[22] 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.

[23] S. Heinz, "A simple deep learning model for stock price prediction using TensorFlow." https://blog.mlreview.com/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877%0A (accessed Jun. 03, 2021).

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

[25] B. Beers, "Buy and hold definition." https://www.investopedia.com/terms/b/buyandhold.asp (accessed May 27, 2020).

[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.

[29] coinmarketcap, "Ether." https://coinmarketcap.com/de/currencies/ethereum/%0A%0A (accessed May 27, 2020).

[30] ethereum main page, "Ethereum." https://ethereum.org/en/%0A%0A (accessed May 28, 2021).

[31] Y. Gola, "These three crypto assets are least correlated to bitcoin, data shows." https://www.newsbtc.com/news/bitcoin/these-three-crypto-assets-are-least-correlated-to-bitcoin-data-shows/%0A (accessed May 21, 2020).

[32] P. R. Pascal Buehler Ken Geeler, *The impact of network architecture and network type of neuralnetworks on trading performance.* econometrics3 project, 2021, pp. 5–7.

Attachment

This bachelorthesis is created with R-4.0.2 , RStudio Version 1.4.904 and RMarkdown in collaborative working via Git / Github https://github.com/phibry/BA

Note that all figures similar to Figure 43: parameter $\beta = \kappa$ from thesis

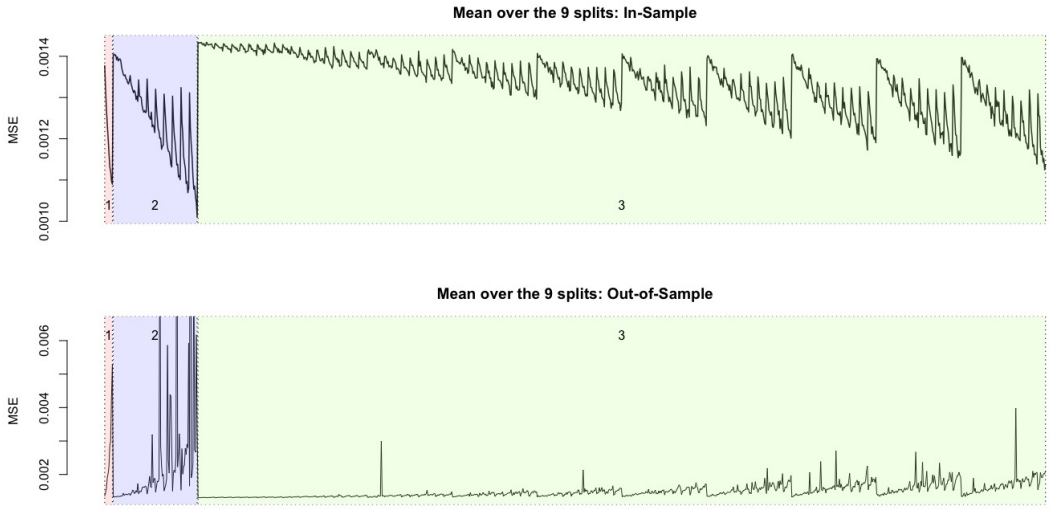

Figure 31: MSE mean over all 9 splits with all 3 layers.

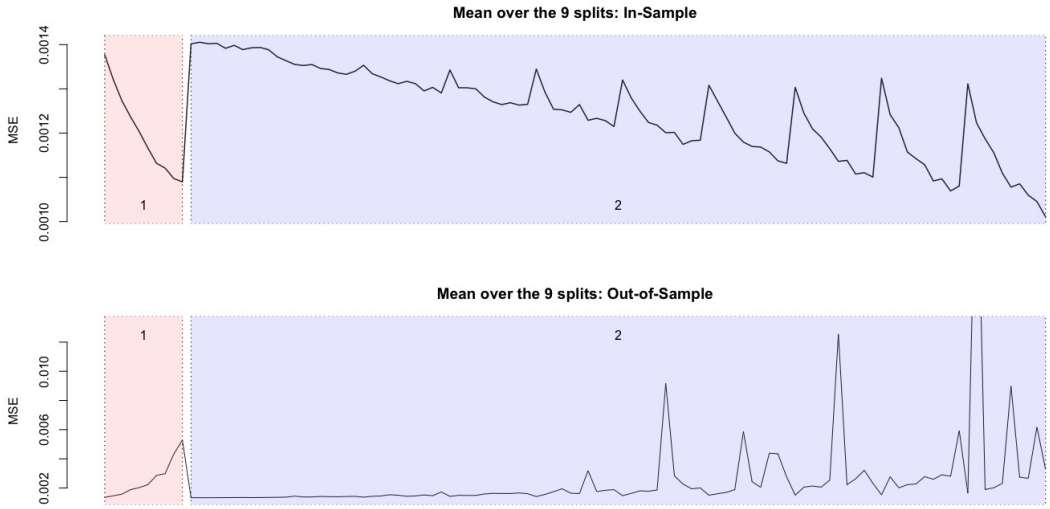

Figure 32: MSE mean over all 9 splits with only 2 layers.

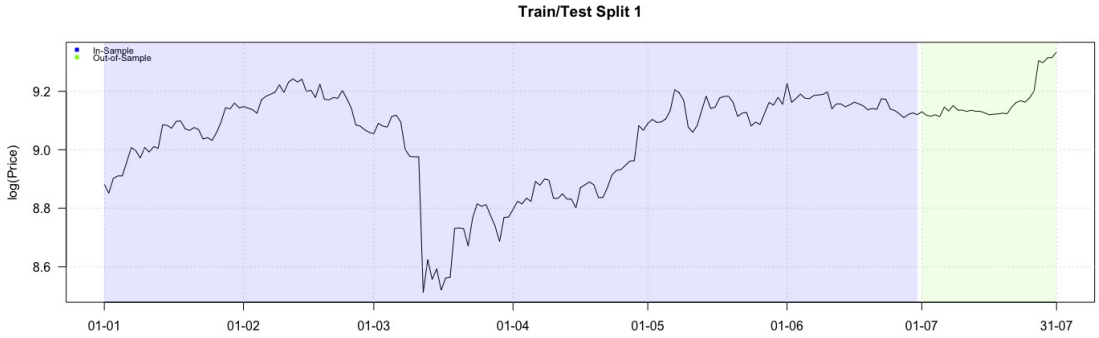

Figure 33: Split 1.

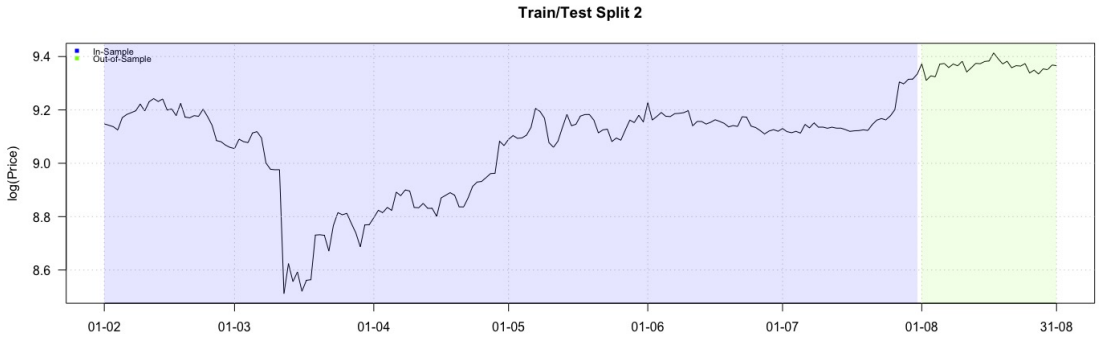

Figure 34: Split 2.

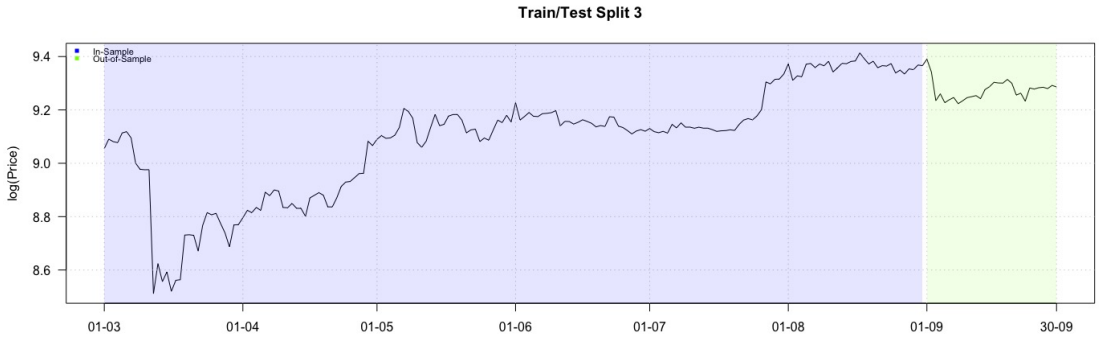

Figure 35: Split 3.

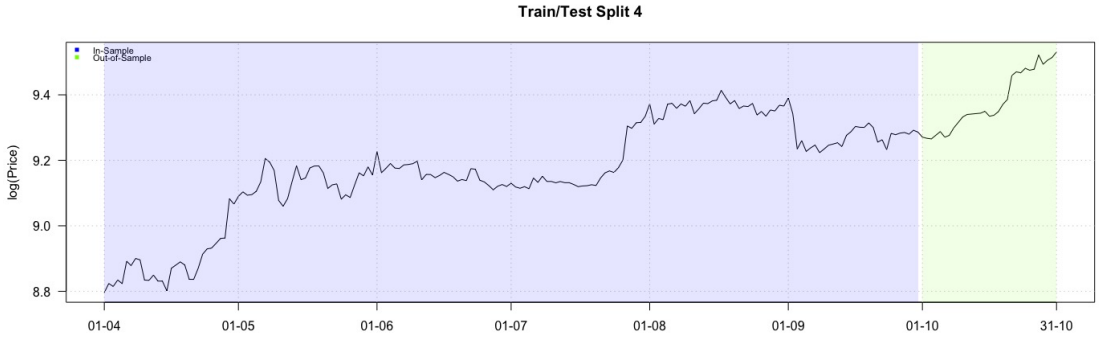

Figure 36: Split 4.

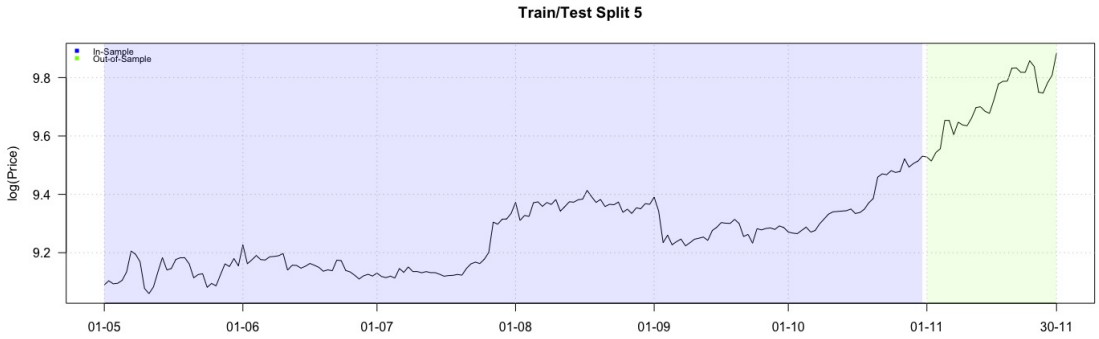

Figure 37: Split 5.

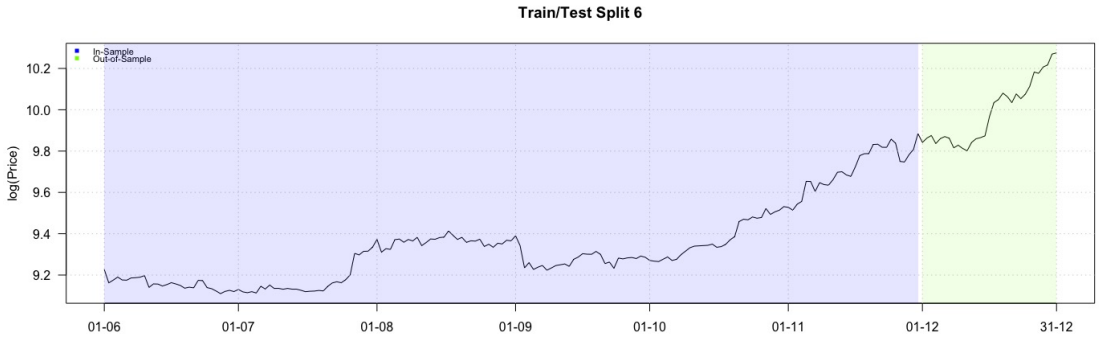

Figure 38: Split 6.

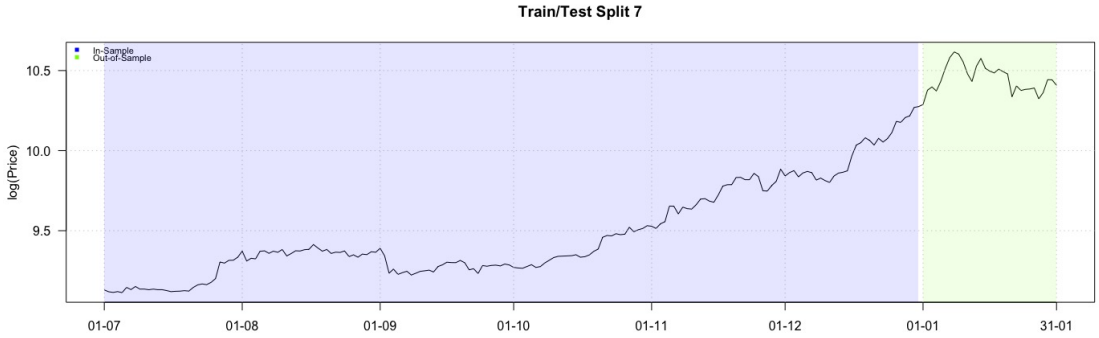

Figure 39: Split 7.

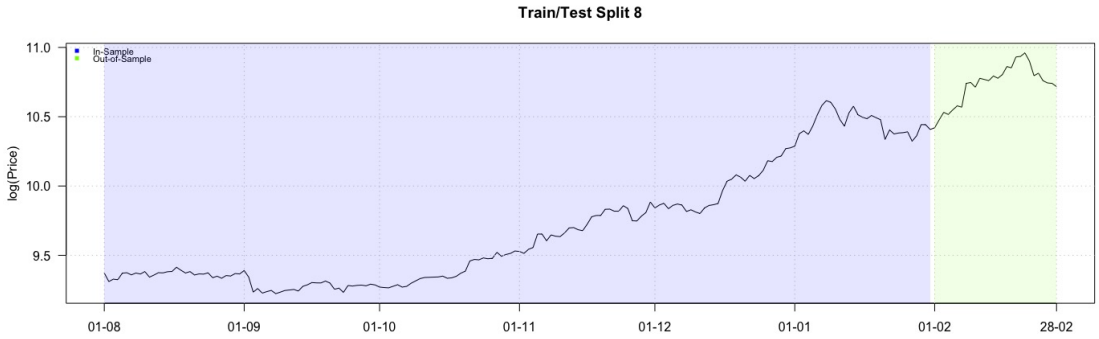

Figure 40: Split 8.

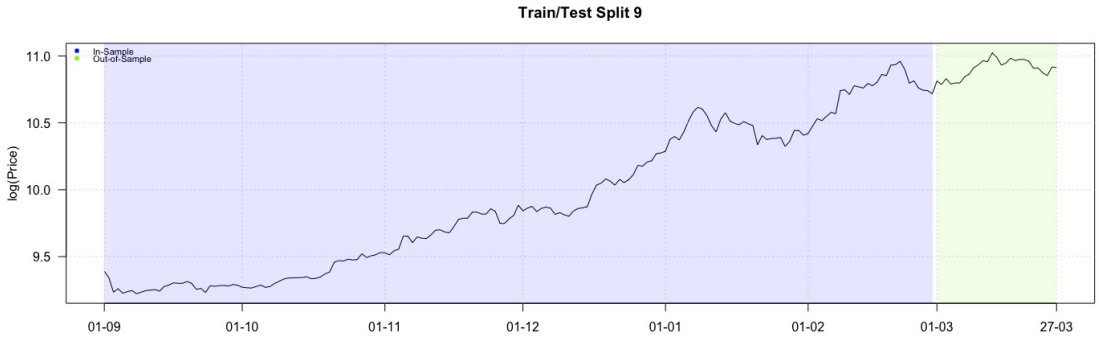

Figure 41: Split 9.

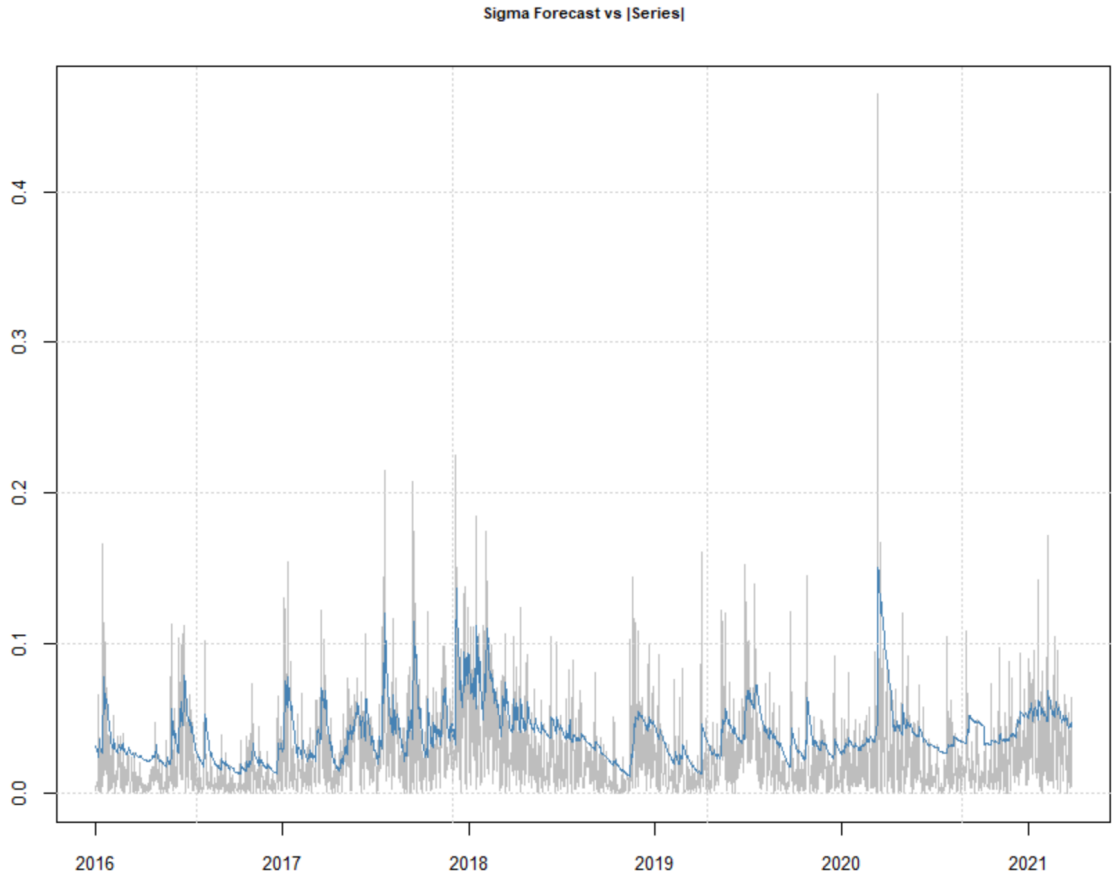

Figure 42: One-step-ahead forecasts of volatility using rolling window of size 365. Refitting model after every months.

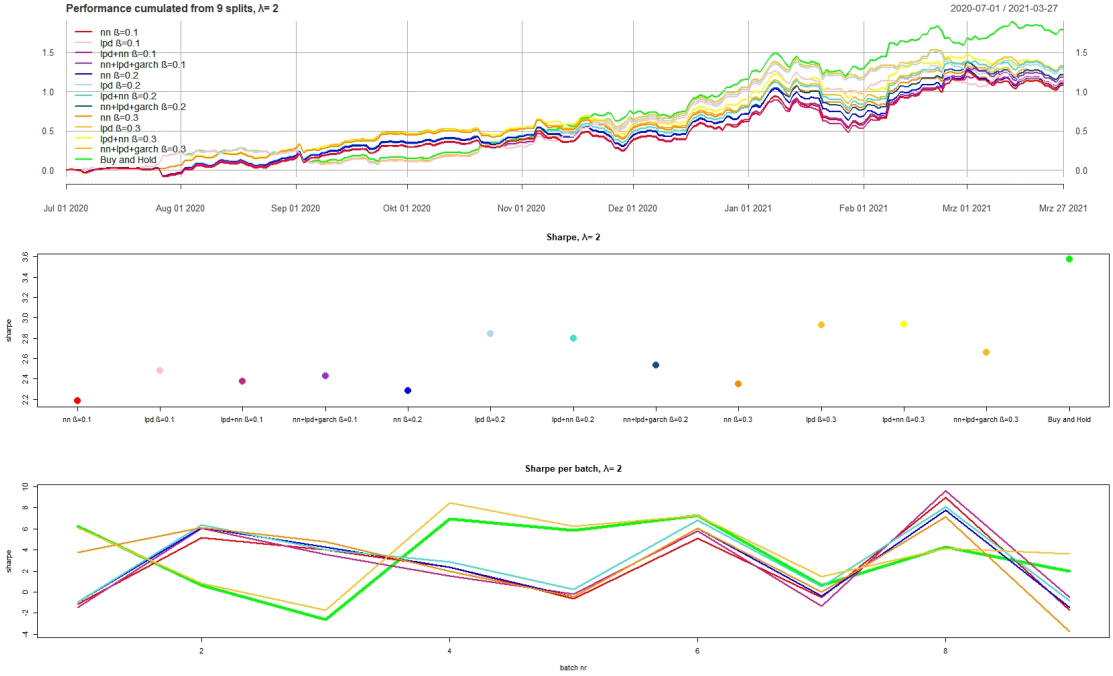

Figure 43: Split 9.

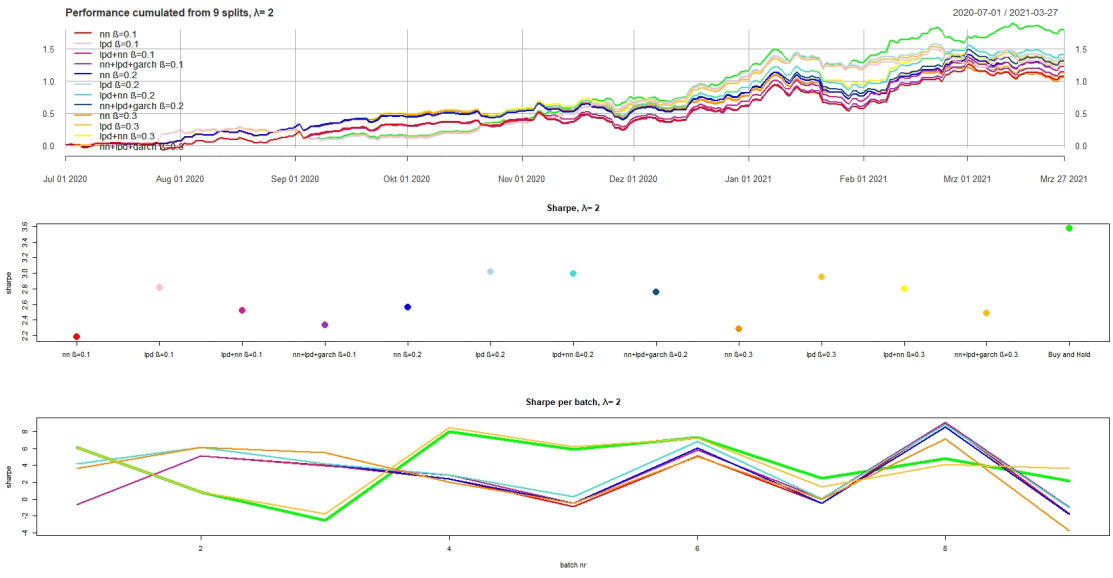

Figure 44: Split 9.

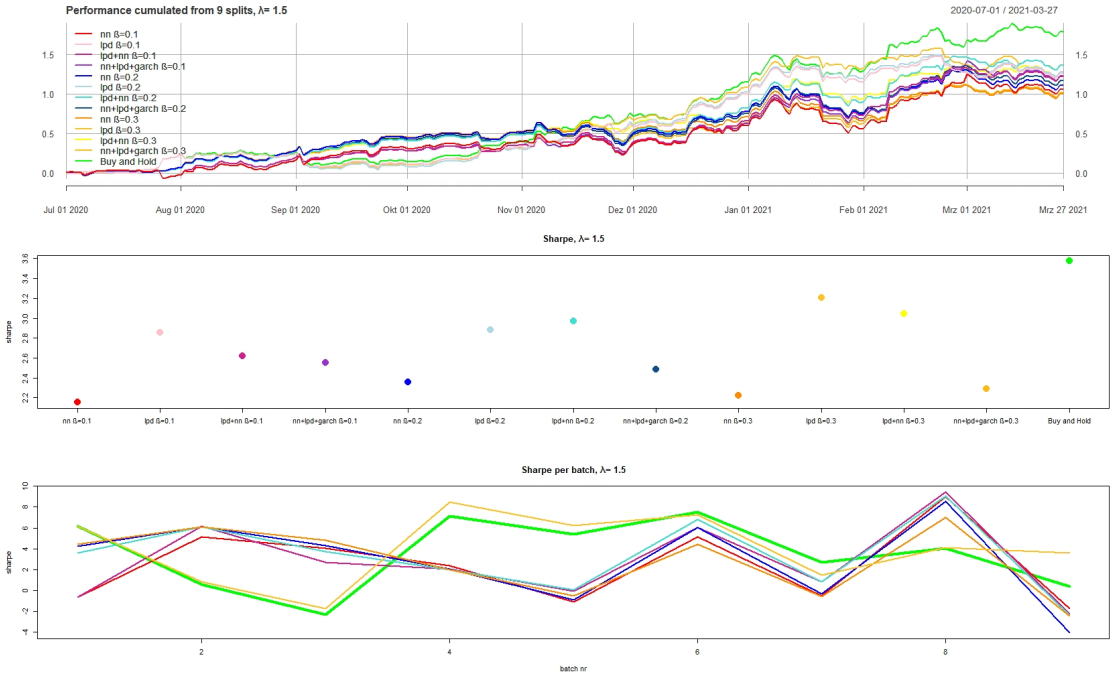

Figure 45: Split 9.

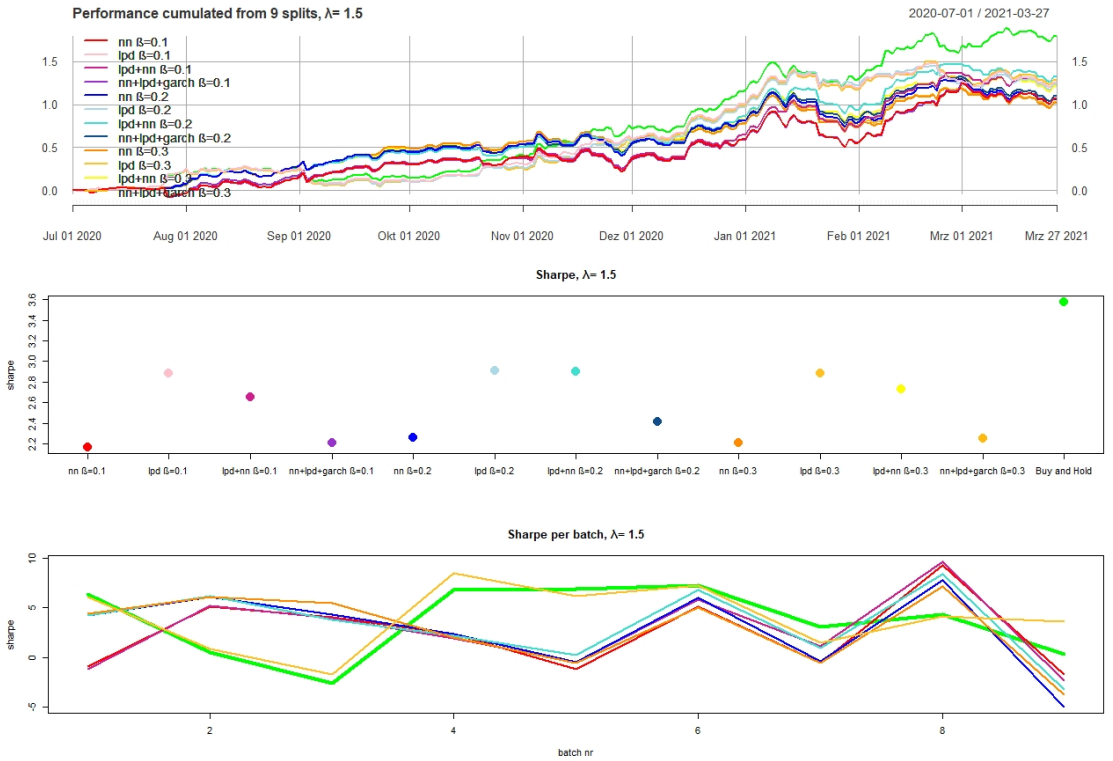

Figure 46: Split 9.

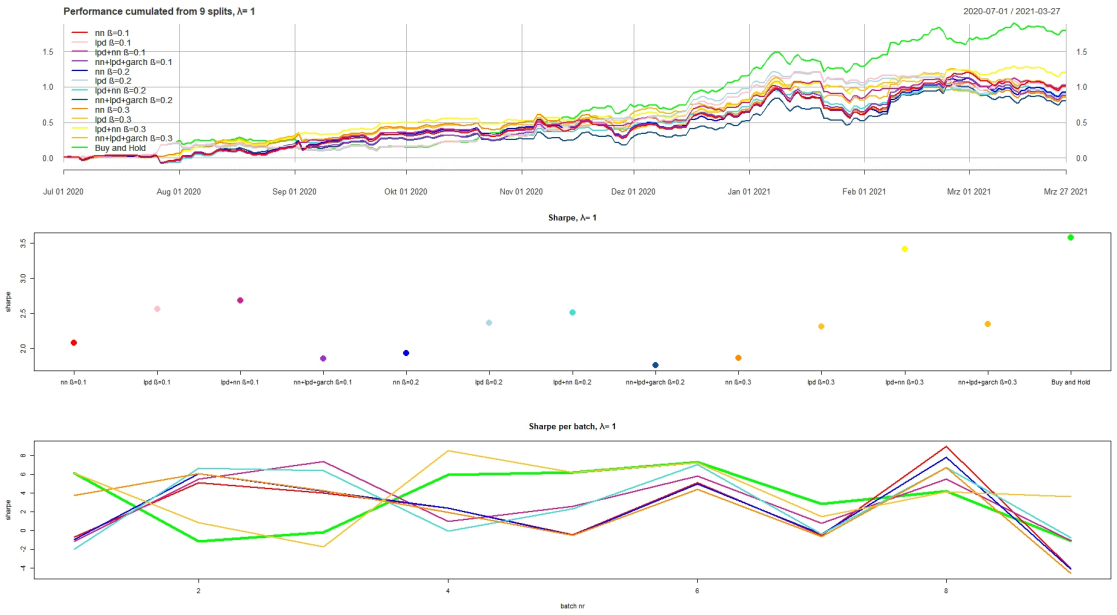

Figure 47: Split 9.

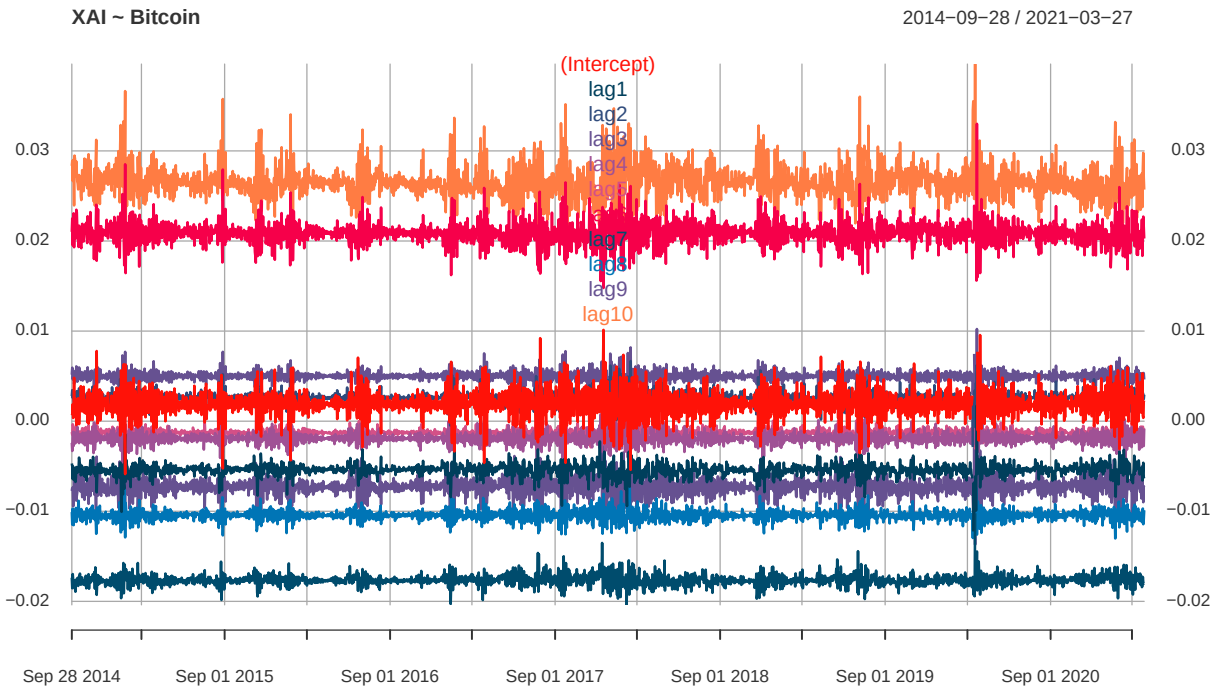

Figure 48: LPD of BTC.