2. Theory

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

2.1. Neural network

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

2.1.1. Perceptron

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

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

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

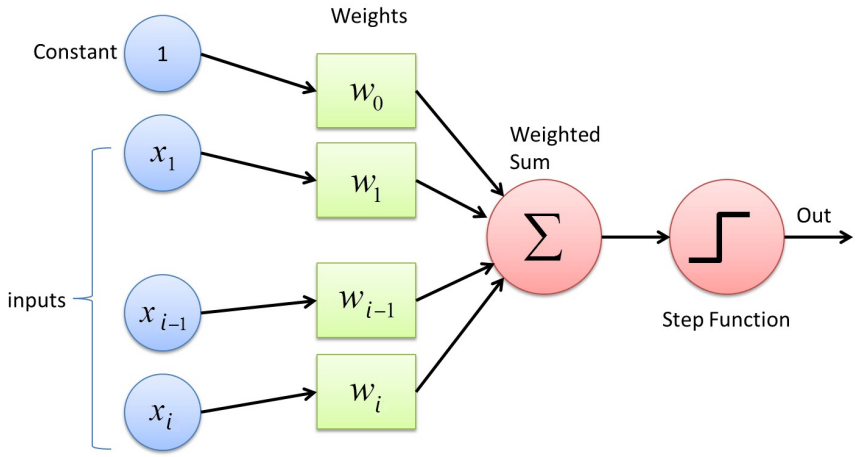

Figure 1: Schematic diagram of a perceptron.

The perceptron can also be represented as a function, which can be seen in equation 1. Analogous to the representation above, the inputs x_i are multiplied by the weights w_i in a linear combination. Then an error term is added so that the whole can be packed into the non-linear activation function $g(S)$. $\hat{y}$ is the binary output of this perceptron. With the aid of an activation function, binary output is obtained. The Heaviside step function shown in figure 1 is usually only used in single layer perceptrons, which recognize linear separable patterns. For the multi-layer neural networks presented later, step functions are not an option, because in the course of the backpropagation algorithm the gradient descent has to be minimized. This requires derivatives of the activation function, which in the case of this Heaviside step function equals 0. Because the foundation for the optimization process is missing, functions like the sigmoid function or the hyperbolic tangent function are used (Martin Anthony 1999). More about this topic is discussed in chapter 2.1.2.

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

As just mentioned, the aim is to feed the perceptron with the training set and change the weights w_i with each cycle so that the prediction becomes more accurate. The output value is compared to the desired value. Finally, the sign of the difference $y - \hat{y}$ determines whether the inputs of that iteration are added to or subtracted from the weights. Ideally, the weights will gradually converge and provide us with a usable model (Martin Anthony 1999).

2.1.2. Backpropagation algorithm

Finding the optimal weights of the neural network is achieved by finding the minimum of an error function. One of the most common methods for this is the backpropagation algorithm. This algorithm searches for the minimum of the error function by making use of a method called gradient descent. The gradient method is used in numerics to solve general optimization problems. In doing so, we progress (using the example of a minimization problem) from a starting point along a descent direction until no further numerical improvement is achieved. Since this method requires the computation of the gradient of the error function after each step, continuity and differentiability of this function must necessarily be given. The step function mentioned above in section 2.1.1. is therefore out of the question, but a non-linear function such as the logistic and the hyperbolic tangent functions (sigmoid) (Rojas 1996). Both activation functions are visible in figure 2. While the target range of the 'ordinary' sigmoid function (equation 2) is between 0 and 1, the $\hat{y}$ of the hyperbolic tangent function (equation 3) ranges between -1 and 1. v_i equals the weighted sum including bias term.

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

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

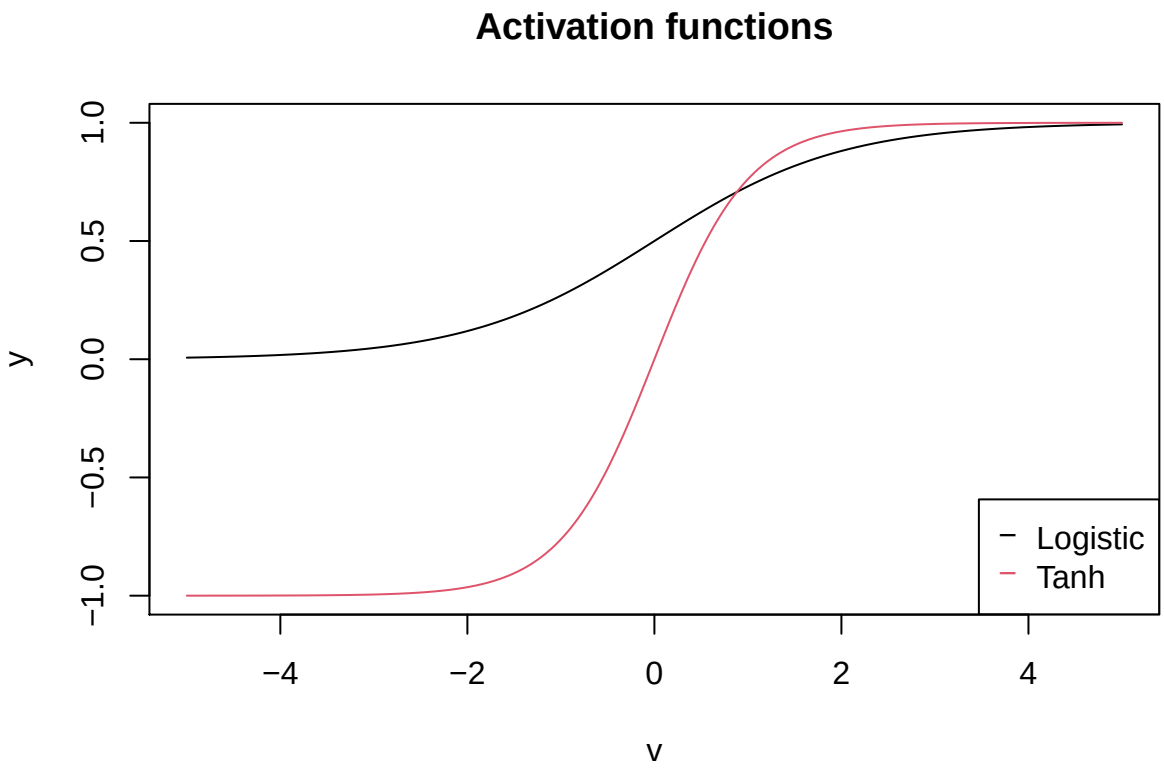

Figure 2: Two common sigmoid activation functions: logistic functions and hyperbolic tangent.

In the course of the error analysis, the output of the neural network respectively the result from the activation function in the output layer is compared with the desired value. The most commonly used error function E is the Mean Squared Error (MSE), which is seen in equation 4. y_i represents the actual value for the data point i, while $\hat{y}_i$ is the predicted value for data point i. The average of this error function is the average MSE, which is determined for a corresponding model. The learning problem is to adjust the weights w_i within the training sample so that $MSE(w)$ is minimized (Yann Lecun 1998).

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

As mentioned, this is searched for by the gradient descent method. The gradient of a function is a vector whose entries are the first partial derivatives of the function. The first entry is the partial derivative after the first variable, the second entry is the partial derivative after the second variable and so on. Each entry indicates the slope of the function in the direction of the variable to which it was derived. In this work, the notation ∇E is used when talking about the gradient for the error function E, which is displayed in equation 5 (Rojas 1996).

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

The weights get adjusted according to the following algorithm 6 where Δw_i is the change of the weight w_i and γ represents a freely definable parameter. In literature, this parameter is often called a learning constant (Hunsberger n.d.). 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 (Rosenblatt 1958). 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 (Hassan Ramchoun 2016). Figure 3 shows the structure of an MLP with n hidden layers. Compared to the perceptron, it can be seen that this neural network consists of an input layer, one or more hidden layers, and an output layer. In each layer, there is a different number of neurons, respectively nodes. These properties (number of layers and nodes) can be summarized with the term 'network architecture' and will be dealt with in this thesis.

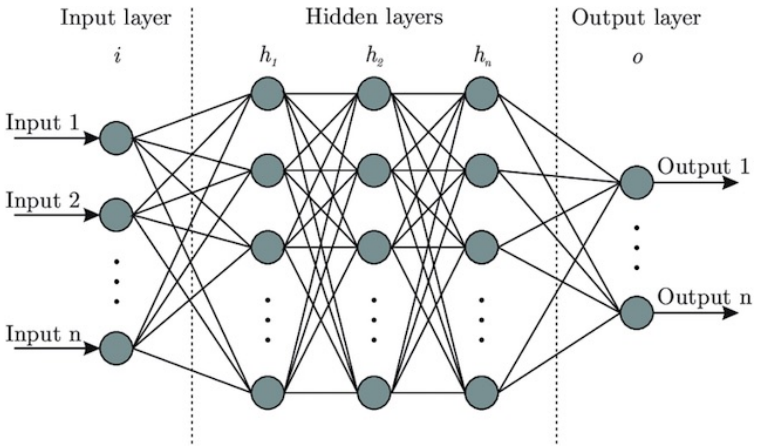

Figure 3: Schematic diagram of a multilayer perceptron

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

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

2.1.4. Recurrent neural networks (RNN)

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

exact location (coordinates) of a plane, it is of great advantage to know the location at $t-1$ and to derive the flight direction from it. With the inclusion of this information, the target area can be narrowed down, which optimally leads to more accurate results. The same principle is used in applications like machine translation and speech recognition, where the result (here possibly letter or word) of the last epoch plays a big role for the next prediction (Ke-Lin Du 2014).

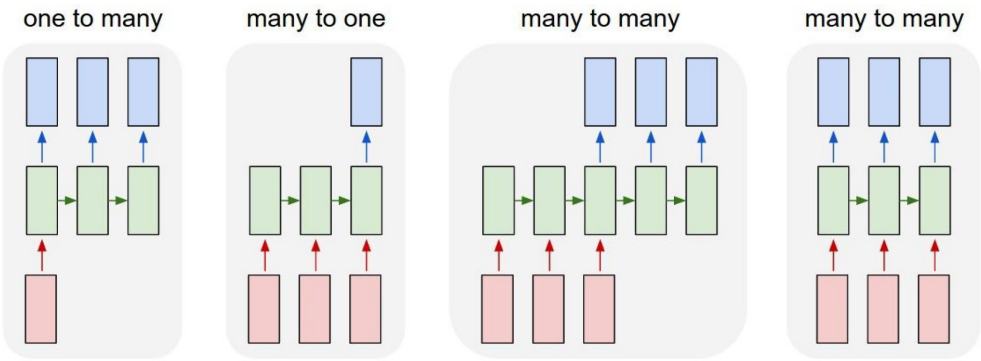

Figure 4: Process sequences of different applicances of RNN.

Figure 4 shows different process sequences of the RNN, which vary depending on the field of application. The red rectangles at the bottom represent the number of inputs. Similarly, the blue rectangles represent the outputs that come out of the RNN. The term 'many' refers to > 1 and is illustrated with three rectangles in the figure. The green ones represent the hidden states h_i of all time steps and thus can be seen as the memory of the neural network. The green arrows show that the previous hidden state is used as input for the current step. Starting from the left: one-to-many can be used for image captioning (extracting sequence of words from images), many-to-one for sentiment classification from sequence of words, many-to-many for machine translation (sequence of words in one language to sequence of words in another language) and many-to-many for video classification on frame level (Fei-Fei Li 2017). For the prediction of the BTC/USD exchange rate in this paper, we deal with the process many-to-one. This method combines information from inputs and hidden states into one single prediction value.

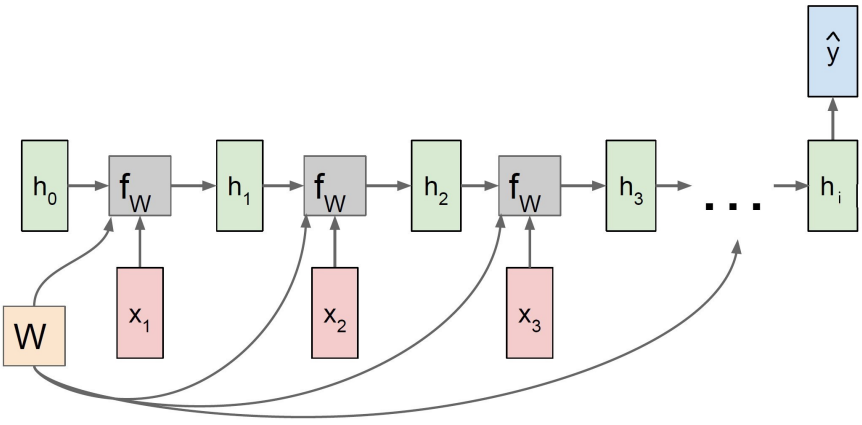

Figure 5: Computational graph of a many-to-one RNN.

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

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

$$\hat{y_i} = W_y h_i \tag{8}$$

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

2.1.5. Long-short term memory (LSTM)

2.1.6. Challenges

2.1.6.1 Overfitting

We have encountered several challenges that can occur when using neural networks. One of these possible problems is called overfitting. The goal of a neural network is to build a statistical model of the training set that is capable of generating the data. In overfitting on the other hand, the exact conditions of the training data including noise are reproduced. The focus is no longer on the underlying function. Last but not least, an unnecessarily large number of parameters or epochs can be 'consumed' for this, which makes the whole process relatively inefficient (Ke-Lin Du 2014).

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

2.1.6.2. Vanishing gradient problem

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

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

2.2. Model comparison

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

2.2.1. Sharpe ratio

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

$$SharpeRatio = \frac{R_p - R_f}{\sigma} \tag{9}$$

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

2.2.2. Diebold Mariano

The second method used is the Diebold Mariano test, which compares the predictive accuracy between two forecasts. First, the loss differential d_i between two forecasts is defined in equation 11 where a loss function L of one model is subtracted from another model. The proposed loss functions include absolute errors (AE) and squared errors (SE) (Francis X. Diebold 1995). Given an expected value of $d = 0$, both forecasts are assumed to have the same accuracy. If the expected value differs from zero, the null hypothesis can be rejected. This would mean that the two methods have different levels of accuracy.

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

with

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

and

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

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

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

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

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

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

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

2.3. Bitcoin

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

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

2.3.1. Historical analysis

The story of bitcoin began with a paper published by the name of Satoshi Nakamoto (Nakamoto 2008). The publisher of the document cannot be assigned to a real person, therefore the technology inventor remains mysteriosly unknown until today. In 2009 the first bitcoin transaction was executed. On account of the opensource technology of bitcoin, lots of alternative currencies were created.

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

Nevertheless in 2014 more companys, such as: Zynga, D LasVegas Casinos, Golden Gate Hotel & Casino, TigerDirect, Overstock.com, Newegg, Dell, and even Microsoft (Chohan 2017), began to accept bitcoin as a payment method. In 2014 the first derivative with bitcoin as an underlying was approved by the U.S.Commodity Futures Trading Commission. 2015 an estimated 160000 merchants used bitcoin to trade. It is observed that the value of bitcoin is very volatile, we will discuss this in a FURTHER XYXY section.

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

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

From this point in time, things began to change, more volume was flushed in the market and the price of BTC began to ascend and the real rally began ,the BTC rose up to 20k USD / BTC on 17th September 2017 **B**. After the rise comes the fall and BTC lost value for more than a year until **C** 2018-12-15 the trend reverted and found its peak after 6 months in **D** 2019-06-26, but oncemore it was not lasting for long as bitcoin lost **D** 2020-03-12 nearly half its value in 4 days. But the story wasn't over by now, after the drop the price of the cryptocurrency regained value, passed previous levels and shortly after exploded, after companies like tesla and signal bought a big chunk of bitcoins, into a maximum of 58000 USD per bitcoin.

Figure 6: Schematic diagram of a perceptron.

Figure 7: Schematic diagram of a perceptron.

2.3.2. SHA256 Hash

- Block
- Blockchain
- Distributed Blockchain
- Token
- Coinbase Transaction
- Public/Private Key -> Signing
- Signature (sign, verify)
- Transaction

Chohan, Usman W. 2017. *A History of Bitcoin.* University of New South Wales, Canberra.

Edwards, John. n.d. "Bitcoins Price History." Accessed March 1, 2021. https://www.investopedia.com/articles/forex/121815/bitcoins-price-history.asp.

Fei-Fei Li, Serena Yeung, Justin Johnson. 2017. *Lecture 10: Recurrent Neural Networks.* Stanford University.

Francis X. Diebold, Roberto S. Mariano. 1995. *Comparing Predictive Accuracy.* University of Pennsylvania.

Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou. 2016. *Multilayer Perceptron: Architecture Optimization and Training.* International Journal of Interactive Multimedia; Artificial Intelligence.

Hunsberger, Luke. n.d. "Back Propagation Algorithm with Proofs." Accessed March 21, 2021. https://www.cs.vassar.edu/~hunsberg/cs365/handouts-etc/backprop.pdf.

Karpathy, Andrej. n.d. "A Recipe for Training Neural Networks." Accessed March 24, 2021. https://karpathy.github.io/2019/04/25/recipe/.

Ke-Lin Du, M. N. S. Swamy. 2014. *Recurrent Neural Networks.* Springer London.

Martin Anthony, Peter L. Bartlett. 1999. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press.

Nakamoto, Satoshi. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System.* online: www.bitcoin.org.

Rojas, Raul. 1996. *The Backpropagation Algorithm.* Springer Berlin Heidelberg.

Rosenblatt, Frank. 1958. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review.

Triacca, Umberto. n.d. *Comparing Predictive Accuracy of Two Forecasts: The Diebold-Mariano Test.* Universita dell Aquila.

Yann Lecun, Genevieve B. Orr, Leon Bottou. 1998. *Efficient Backprop.* Image Processing Research Department AT& T Labs.