Studying and Forecasting Trends for Cryptocurrencies Using a Machine Learning Approach

Johan Petter Möller

Bachelor's thesis
2018:K15

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

LUNDS UNIVERSITET

Lund University, Faculty of Science, Numerical Analysis

Bachelor thesis in Numerical Analysis

Studying and Forecasting Trends for Cryptocurrencies Using a Machine Learning Approach

Author :
Johan Möller

Supervisor :
Alexandros Sopasakis

October 16, 2018

Abstract

We consider a technique involving Neural Networks in order to try to predict trends for cryptocurrencies such as Bitcoin, Ethereum etc. In that respect the project involves construction, design and training of a deep learning Neural Network based on historical trading data for these cryptocurrencies. Subsequently we apply this trained network in order to better understand its ability to possibly forecast future trading trends.

Populärvetenskaplig sammanfattning

Genom att använda maskininlärning med fokus på artificiella neuronnät ska vi försöka preidktera trender för kryptovalutor, så som Bitcoin, Ethereum etc. För att genomföra detta krävs det att konstruera, designa och träna ett djupt neuronnät baserat på tidigare data. Sedermera används detta neuronnät för att bättre förstå maskininlärning och dess förmåga att prediktera framtida trender.

Acknowledgment

I would like to thank my supervisor Alexandros Sopasakis for all the help, encourage and enthusiasm.

Contents

1 Introduction

Forecasting trends for assets such as cryptocurrencies is not an easy task. In this thesis the goal is to understand how a machine learning method can be used for forecasting trends of time series data. Machine learning is, in some sense, smart algorithms that can draw conclusions from data, i.e learn data patterns. Further, a Brownian motion will be used for forecasting trends of the same data. This method will be used as a comparison.

The author does not believe, what so ever, that one can easily predict the future of cryptocurrencies or similar assets by constructing a machine learning algorithm or Brownian paths for forecasting trends. The author does believe that machine learning is a powerful tool for comprehending data pattern.

Cryptocurrencies are often in spotlight and then in context of wide volatility and high returns. Therefore, it seems interesting to see how a machine learning algorithm can be used for comprehend data pattern of such assets. The machine learning algorithm will involve neural networks and a rolling prediction forecast. Software such as: Keras, TensorFlow and Scikit-Learn are used for writing the algorithms. All code is written in Python.

In this thesis a regression method with supervised learning will be investigated. Regression meaning that the input map to an output and not a probability. Supervised learning meaning that the user must provide a label to each input so the algorithm can compare prediction output with true label. Subsequently, the algorithm will make predictions on data inputs which have no labels.

1.1 Problem Statement

In this thesis, a method for predicting the closing price using neural networks will be investigated. The prediction will be based on former closing prices. This method will be compared with another method, a Geometric Brownian Motion.

Let $\vec{X} = \big(X_1, X_2, X_3, ..., X_t\big)$ be a vector of closing prices at certain time points from $i = 1, ..., t$. Further, let $F(\cdot)$ denote a neural network. Can a neural network be used for forecasting trends of closing price of different cryptocurrencies, $\vec{\hat{X}} = \big(\hat{X}_{t+1}, \hat{X}_{t+2}, \hat{X}_{t+3}, ..., \hat{X}_{t+k}\big)$ where k is the prediction sequence length.

$$F(\big(X_1, X_2, X_3, ..., X_t\big)) = \big(\hat{X}_{t+1}, \hat{X}_{t+2}, \hat{X}_{t+3}, ..., \hat{X}_{t+k}\big) \tag{1}$$

2 Neural Network

Artificial neural networks (ANN) are designed to have layers. Layers consist of neuros, sometimes called nodes. In the first layer, the input layer, the neurons are sending the input further into the layers. The input is sometimes referred to as the input signal or just the signal. The signal is provided by the user. Each neuron in each layer is connected to each neuron in the next layer. Each connection is a weight. A weight is an unknown variable. The weight is combined with the signal obtained from the previous layer. In each neuron two operations are executed. One summoning junction which sum the signals from previous layer combined with a weight. The second operation is called a "wrapper". This is a function, called activation function. The activation function takes the signal as input The activation function is often a non-linear function which introduce non-linearity to the network. This is called for when the relation between input and output is non-linear. A smart choice of activation function may reduce the training time. Training is optimization of the difference between network output and desired output with respect to the weights. Training is discussed and explained in subsection "Training and Testing" (2.2). The signal is traversed through the hidden layers. If the network has more than one hidden layer, the network is considered a deep learning neural network. When the signal has traversed through the hidden layers it reaches the final layer. The output layer. In the output layer each signals are summed. The output from the output layer is the network output. The network output is compared to the desired output. There is no optimal number of neurons or hidden layers. A rule of thumb is if the problem the network is designed to solve is non-linear or complex, (i.e the relations between input and output), a multilayered network may decrease the training time and increase the test result. A design suitable for the problem is found by trial and error.

2.1 Activation Function

The activation functions role is to introduce non-linearity to the network. This is called for when the relationship between input variable and output label is non-trivial. A couple of different activation functions are the following:

$$ReLU = \max(0, x)$$

$$Sigmoid,\ \sigma(x) = \frac{1}{1 + e^{-x}}$$

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

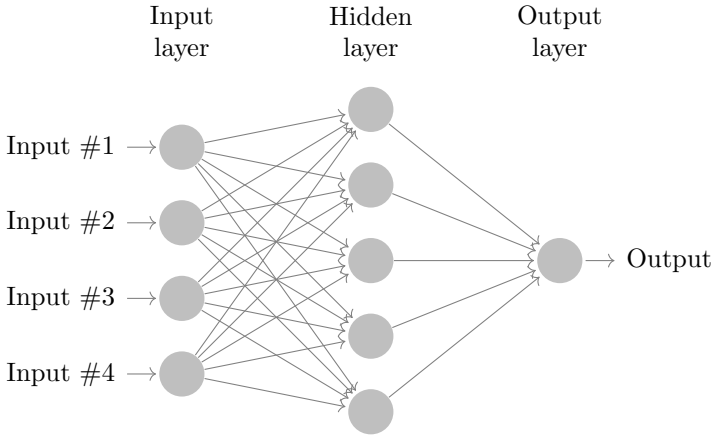
Different activation functions fit different tasks better even though they all can be used for all tasks. For example, $\sigma(x)$ fits classification problems better and ReLu fits better for a regression problem. It has been experimentally discovered the choice of activation function mostly affects training time (1).

2.2 Training and Testing

Training is done be evaluating the difference between network-output and target and subsequently minimizing the differences. Target is the desired value. The training data and test data must be provided by the user. The user has data points which are divided into train and test data. these are often called training-set and test-set. The test-set is unseen to the network and used to validate the network ability to approximate the data. The training set is used for training through backpropagation. Throughout training the goal is to modify the weights, so the network-output approximate the target as good as possible. The targets are also given by the user. The target is sometimes called label or desired value.

2.2.1 Example of Neural Network

In figure below an artificial neural network with design 4-5-1 is considered. This is a figure which is supposed to simplify the understanding of how a network is functioning. The neurons which are exemplified by circles are the functions and the connections (the lines between the nodes) are the weights. In this network only one hidden layer is used, therefore it is considered a single layer feedforward neural network. Feedforward refers to the input is traversed from right to left (in this case). The output for the last neuron, the output layer, as compared to the target. The difference referred to as the error. The error is then propagated backwards to measure the error contribution in each neuron. This is what the algorithm aims to minimize.



2.2.2 Backpropagation Algorithm

Backpropagation is the algorithm used for training networks, i.e minimize the difference between network output and target.

- $f(\cdot)$ denotes the activation function.

- x_i denotes the input. Provided by the user.
- $\hat{y}_i$ denotes neuron output.
- z_i denotes network output.
- y_i denotes target. Provided by the user.
- w_i denotes weights. Set at random and the updated.
- w_{j0} denotes biases.
- $E(\cdot)$ denotes the error function, i.e the difference between network output and target value.
- δ_i denotes the error.

Define a neuron with input signal as:

$$a_j = \sum_i w_{ji}\hat{y}_i + w_{j0} \tag{2}$$

Remark that if $\hat{y}_i = x_i$ if $a_j = a_1$ as $j = 1$ implies the algorithm is in the first layer.
Now, the activation function, $f(\cdot)$, is applied:

$$\hat{y}_j = f(a_j) \tag{3}$$

If $\hat{y}_j$ is located in the last layer, the output layer, $\hat{y}_j = z_j$. The error function[1] is defined, the error function measures the difference between network output and target.

$$E = \sum_j^n E^{(j)} \tag{4}$$

E is as function dependent on the signals a_j, hence:

$$E = E(a_1, a_2, ..., a_k) \tag{5}$$

Look at equation (2) we see that the only unknown variable is w_{ij}. Hence, $E = E(a(w))$. Now, differentiating the error function with respect to a particular weight, w_{ij}:

$$\frac{\partial E_j}{\partial w_{ij}} \tag{6}$$

Using the chain rule[2] the derivative is divided into two parts. Recall that E_j is a function depending on a_j which in its turn depends on w_{ij}, therefore:

$$\frac{\partial E^{(j)}}{\partial w_{ij}} = \frac{\partial E^{(j)}}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \tag{7}$$

[1]Error function can be defined in different ways, see Section "Error function"
[2]Recall chain rule: $y = f(x),\ x = f(t),\ \frac{\partial y}{\partial t} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial t}$

Define:

$$\delta_j = \frac{\partial E^{(j)}}{\partial a_j} \tag{8}$$

δ_j being the errors in neuron j.

Using Equation (2), one sees:

$$\frac{\partial a_j}{\partial w_{ij}} = \hat{y}_i \tag{9}$$

Hence:

$$\frac{\partial E^{(j)}}{\partial w_{ij}} = \frac{\partial E^{(j)}}{\partial a_j}\hat{y}_i \tag{10}$$

Recall $\delta_j = \frac{\partial E^{(j)}}{\partial a_j}$,

$$\frac{\partial E^{(j)}}{\partial w_{ij}} = \delta_j \hat{y}_i \tag{11}$$

In Equation (11) one can see that the derivatives are the product of the neuron output and the error from the previous layer (this is possible because we propagate backwards and neuron output is obtained from forward propagation done before). This can be calculated using the local derivative of the activation function in, $f'(\cdot)$ because $f'(a_j) = \hat{y}_i$ where $\hat{y}_i$ is a neuron output from previous layer.

Consider the signal traversed through the hidden layers ending up in the output layer. Here, the output is called z_k instead of $\hat{y}_i$. k indicates that the signals is located in the output layer. The signal is now referred to as a_k. The error function in current point is still referred to as $E^{(j)}$. The equation below (Equation. (12)) is formed.

$$\delta_k = \frac{\partial E^{(j)}}{\partial z_k} f'(a_k) \tag{12}$$

Equation (12) is the form to evaluate the deltas in the output layer. The deltas in the hidden layers are calculated using the formula:

$$\delta_j = f'(a_j) \sum_k w_{kj}\delta_k \tag{13}$$

This comes from splitting the derivatives of the error function using the chain rule once more, like:

$$\begin{aligned} \frac{\partial E^{(j)}}{\partial a_j} &= \delta_j \\ \frac{\partial E^{(j)}}{\partial a_j} &= \sum_k \frac{\partial E^{(j)}}{\partial a_k}\frac{\partial a_k}{\partial a_j} \end{aligned} \tag{14}$$

We end up in a formula for calculating the errors in each neuron, Equation (13).

3 Optimization of The Error Surface

The graph of the error function (Eq. (4)) is referred to as the error surface. Training a network consist of more than finding the error contributions in each neuron. Training a network consists of optimizing the error surface. Through optimization the training algorithm aims to find weights which represent a minima on the error surface and therefore the weight update have more impact on training. Backpropagation delivers the gradients of the network.

3.1 Error Functions

Measuring the error is executed by different error functions. Two of the are Mean Squared Error (MSE) is one (Eq. (15)) and Absolute Error is another (Eq. (16)).

$$E = \frac{1}{n} \sum_{i}^{n} \left(z_i - y_i \right)^2 \tag{15}$$

$$E = \frac{1}{n} \sum_{i=1}^{n} |z_i - y_i| \tag{16}$$

3.2 Optimization Algorithm

Different optimization algorithms are used. They are mostly different methods which originate from Newton's method. One optimization algorithm commonly used to optimize the error surface of a neural network is Adam. The name Adam is derived from *adaptive moment estimation*. Below an algorithm will be presented for using the Adam algorithm for calculating a minima of the error surface.

3.2.1 Pseudocode for Adam Optimization Algorithm

Algorithm 1 Algorithm optimizing $f(\cdot)$ using Adam algorithm for optimizing

Result: w which minimizes $f(\cdot)$

Require

η: stepsize

$\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

$f(w)$: Function to be optimized

w_0: Initial parameter vector

m_0: Initial first moment vector

v_0: Initial second moment vector

$t = 0$: First timestep

while w *not converged* **do**

$t = t + 1$ (1)

$g_t = \nabla f(w_{t-1})$ (2)

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (3)

$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (4)

$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$ (5)

$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$ (6)

$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}$ (7)

end

3.2.2 Algorithm Description

Below, a description of every step on Adam will be presented.

1. Timestep is updated
2. g_t is the gradient of $f(\cdot)$ at timestep t.
3. m_t is the moment estimation. That is because the algorithm is minimizing the function with respect to its expected value. I.e, m_t is the expected value of the gradient.
4. v_t is the second moment, the mean squared calculated with: $\nabla f\big((\frac{\partial f}{\partial w_1})^2, (\frac{\partial f}{\partial w_2})^2, ...\big)$
5. The estimation of the moment is biased and the term $1/1 - \beta_1^t$ is compensating for the bias-term.

$$m_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i$$

$$E[m_t] = E\left[(1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i\right]$$

$$E[m_t] = E(g_t)\left[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\right]$$

Consider $t = 2$. Then the above case ca be expandes as,

$$(1-\beta_1)\big(\beta_1^{2-1} + \beta_1^{2-2}\big) = (1-\beta_1)(\beta_1+1) = 1-\beta_1^2$$

This is true for all t, therefore the algorithm compensating with $1/1-\beta_1^t$ and becomes,

$$\frac{E(m_t)}{1-\beta_1^t} = E(g_t) \tag{17}$$

6. This is analogues to the case above.

7. Here, w_t is updated and a new iteration is starting.

Empirically it has been shown that good hyperparameter settings are: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\eta = 0.0001$. For further improvements, η may be updated each iteration as follows:

$$\eta_t = \eta\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \tag{18}$$

It has been shown that Adam is used preferably before other optimization algorithms in machine learning problems, due its efficiency in a high dimensional parameter space (9). Machine learning algorithms are often considered to often be in high dimensional parameter space since the connections between nodes quickly add up to many parameters.

Consider the Mean Squared Error,

$$MSE = 1/n\sum_i (z_i - y_i)^2 = 1/n\sum_i E(z_i, y_i).$$

The MSE would normally be a convex function. Our MSE however is depending on function values depending on more variables, such as weights. Therefore $E(F(\vec{X}, w), y)$ where $F(\cdot)$ is a neural network, w is a vector of all weights and $\vec{X}$ is the input vector. In general $F(\cdot)$is not convex and therefore $E(z, y)$ is not convex. This may cause a problem because the optimization algorithm does not guarantee a global minima. Research have shown that the importance to find a global minima decreases when the network structure becomes larger (6). If the structure is large the algorithm only needs to find a "big-enough-minima". With these explanations new questions arises. What is a "big-enough-minima"? Which structure is considered large? A "big-enough-minima" simply refers to a minima which as impact on training, i.e. lower the training and testing score. However, a structure is large or small is connected to the problem (6). These loosely defined problems and solutions may cause problems for the user to construct a network that is able to train well. To plot the errors each epoch is a

good indication if the network behaves as the user want. In Figure 1 a plot of a successfully trained network is shown. A common problem is overfitting. This can be detected if the train score is decreasing and test score is increasing. Remember, a low score is desirable.

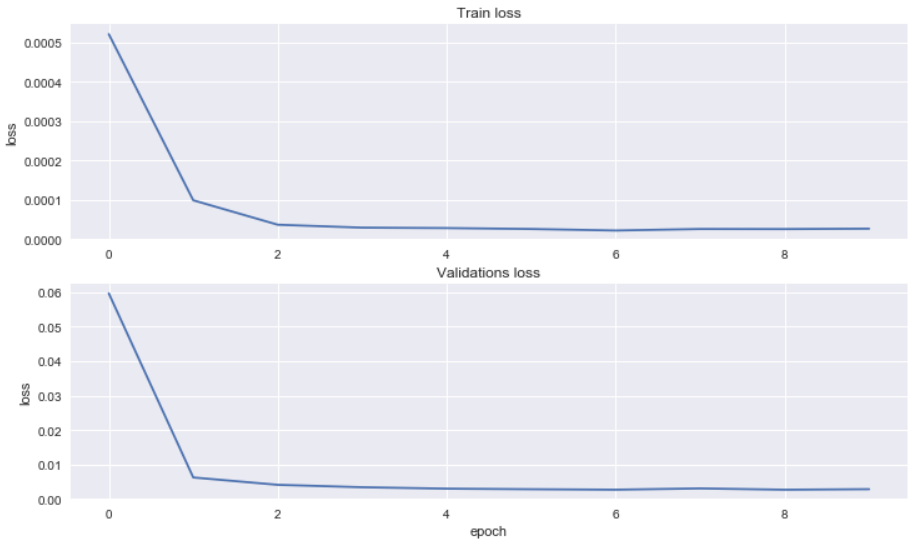


Figure 1: Top plot: plot of training error. Bottom plot: plot of test score. Error converging to a small error after five epochs. No overfitting. Successfully trained network. If train loss would converge to zero but validation loss would increase, it would indicate overfitting.
In the plot Train Loss and Validation Loss refers to train score and respectively test score. A low score is desirable.

3.3 Biases and Weights Exemplified

In this section, the use of weights and biases will be exemplified. Note that biases in this section do not refer to a bias estimation but a trainable parameter. A bias is added to make the neuron output more flexible.

Consider the input signal presented in Subsection 2.2.2.

$$a_j = w_{ji}\hat{y}_i + w_{j0} \tag{19}$$

The weight makes the approximation more or less steep and the bias has the ability to move the signal on the x-axis, similar to a degree one polynomial. First, initial bias and weights are set at random.

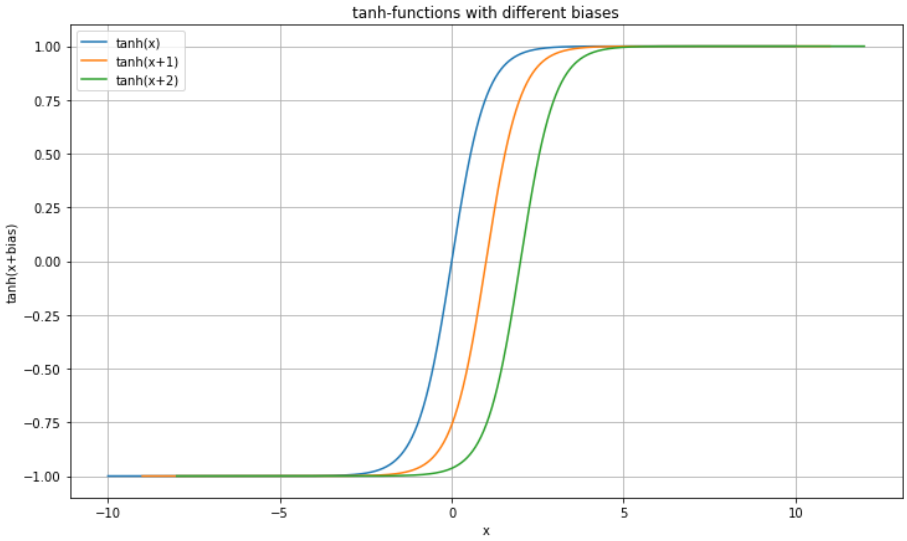


Figure 2: tanh-functions with different biases added. Weights are set to be one. Here one sees how the bias may shift the activation function output on the x-axis. Blue curve: $\tanh(x)$, Orange curve: $\tanh(x+1)$, green curve: $\tanh(x+2)$.

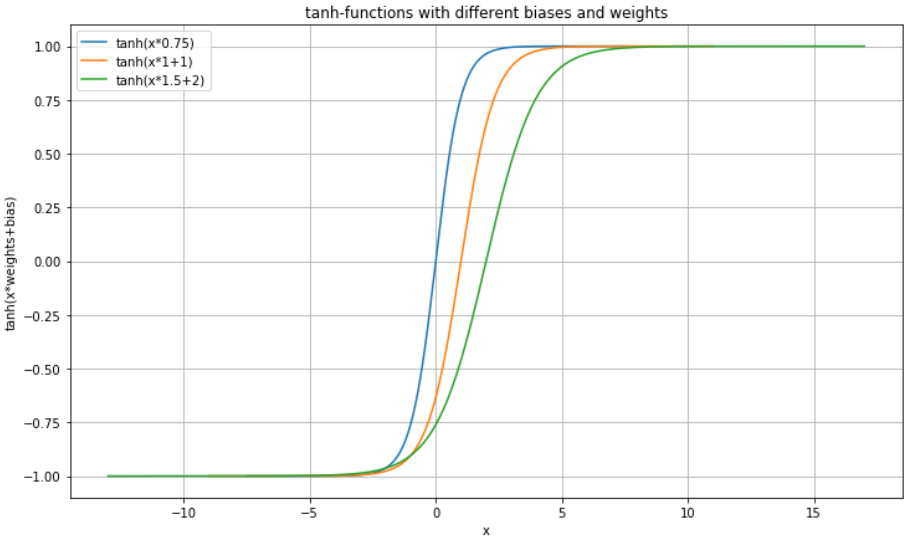


Figure 3: *tanh*-functions with different weights and biases, exemplifies the role of the weights. Here one sees how the weights make the activation function steeper and the biases shifting the curve on the x-axis. Blue curve: $\tanh(0.75x)$, Orange curve: $\tanh(x+1)$, green curve: $\tanh(1.5x+2)$.

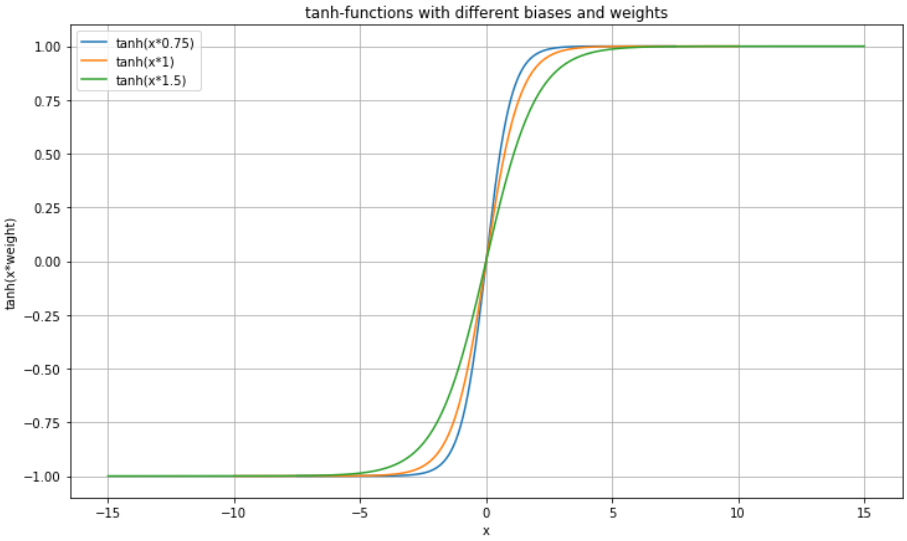
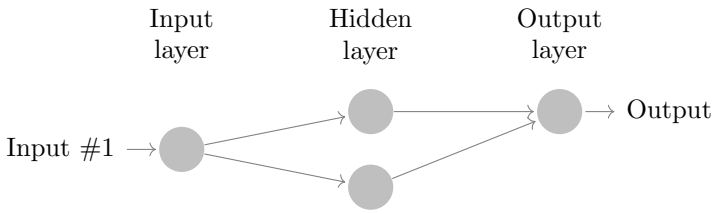


Figure 4: tanh-functions with different weights, exemplifies the role of the weights. Here one sees how the weights increases or decreases the steepness of the curve. Blue curve: $\tanh(0.75x)$, Orange curve: $\tanh(x)$, green curve: $\tanh(1.5x)$.

3.3.1 Example of Backpropagation

Consider a single layer network with one input neuron, two neurons in one hidden layer and one output neuron in the output layer (see network below).



- $f(\cdot)$ denotes the activation function.
- x_i denotes the input. Provided by the user.
- $\hat{y}_i$ denotes neuron output.
- z_i denotes network output.
- y_i denotes target. Provided by the user.

- w_i denotes weights. Set at random and the updated.

1. Forward propagate input x_1 to the two neurons in the next layer, the connections are weights. Here, only one operation is executed in each neuron. This is because there is only one input so no summation is needed for. For simplification the activation function used in the example is the identity.

$$\begin{aligned} f(x_1, w_1) &= x_1 w_1 = \hat{y}_1 \\ f(x_1, w_2) &= x_1 w_2 = \hat{y}_2 \end{aligned} \tag{20}$$

2. Each neuron output, $\hat{y}_1$ and $\hat{y}_2$ is now send forward to the output layer via two new connections, i.e w_3 and w_4.

$$\begin{aligned} f(y_1, w_3) &= \hat{y}_1 w_3 = \hat{y}_4 \\ f(y_2, w_4) &= \hat{y}_2 w_4 = \hat{y}_5 \end{aligned} \tag{21}$$

3. Now the network output is calculated. Here the summering operation is demonstrated. Also, notice that in this step no weights are combined with the output from this neuron. That is because the output is not traversed forward. The connections between neurons are weights.

$$f(\hat{y}_3, \hat{y}_4) = \hat{y}_3 + \hat{y}_4 = z_1 \tag{22}$$

4. In this step the output z_1 is compared with the target. Notice, y_1 is the target provided by the user.

$$z_1 - y_1 = \delta_5 \tag{23}$$

5. δ_5 is now propagated backwards and measure the error contribution in each neuron.

$$\begin{aligned} w_4 \delta_5 &= \delta_4 \\ w_3 \delta_5 &= \delta_3 \\ w_2 \delta_4 &= \delta_2 \\ w_1 \delta_3 &= \delta_1 \end{aligned} \tag{24}$$

6. In this step each weight, w_i, is updated. This is done by an optimization algorithm. Here η denotes the stepsize and $\frac{\partial f}{\partial \hat{y}_i}$ is the partial derivative of the activation function in each step.

$$\begin{aligned} w_1' &= w_1 + \eta \frac{\partial f}{\partial \hat{y}_1} x_1 \delta_1 \\ w_2' &= w_2 + \eta \frac{\partial f}{\partial \hat{y}_2} x_1 \delta_2 \\ w_3' &= w_3 + \eta \frac{\partial f}{\partial \hat{y}_3} \hat{y}_1 \delta_3 \\ w_4' &= w_4 + \eta \frac{\partial f}{\partial \hat{y}_4} \hat{y}_2 \delta_4 \end{aligned} \tag{25}$$

7. w_i' is the updated weight. This procedure is called an epoch. An epoch is an iteration of the algortihm.

3.3.2 Pseudocode for Backpropagation

Algorithm 2 Algorithm for finding δ's using backpropagation

Result: δ_j for $j = 1, 2...k$ layers
while $j \leq k$ **do**
  for *each* w_i, i = 1,2,...n. n *number of connections* **do**
    if $j = k$ **then**
      $\delta_j = f'(a_j)\frac{\partial E}{\partial z_j}$ **else**
        $\delta_j = f'(a_i)\sum_j w_{ij}\delta_j$
      end
    end
  end
end

4 Problems

Training a network consists of training many parameters and adjusting hyperparameters. A hyperparameter is a parameter which can not be trained and must be set by the user. Setting hyperparameter is done by trial and error. Example of hyperparameters are the number of layers and neurons. Others are stepsize, iterations and batch size. Problems that occur are vanishing gradient, overfitting and optimization problems. The optimization problem is discussed in "Optimization of The Error Surface".

4.0.1 Recall: Training and Testing Score

Training score and test core is the score obtained from training and testing (sometimes referred to as training and validation). Measured, in our case, with the Root Mean Squared Error-function. The train score is the score which measures how good the approximation is of the train-set (training data). This score is the mean difference between network output and target. The difference between network output and target is what the training algorithm aims to minimize, i.e. obtaining low train score. If the network successfully trained the test score will be low. To get an understanding of what is a low score one may have to try different designs to try to find the lowest possible. Another good indicator of a low score is to plot the approximation and the target to get a visual if the approximation meets the user requirements. One example of a function measuring the error is the Root Mean Squared, see Equation (26).

Here, X_i is the network approximation of the target, y_i.

$$Error = \sqrt{\frac{1}{n}\sum_{i}^{n}\left(X_i - y_i\right)^2} \tag{26}$$

4.1 Normalization

The train and test score will be lower if the data is normalized. Popular normalization ranges are: $[0, 1]$ and $[-1, 1]$. If the data is not normalized the error surface will range over a wide interval. The optimization will be easier and better with normalized data and therefore lowering the train and test score.

4.2 Vanishing Gradient

Vanishing gradient occurs when the gradient which is propagated backwards through the network is too small and therefore has no impact on training. The reversed problem, exploding gradient, is when the gradient is too big and have too much impact. Due to the fact that the gradient is traversed backwards through the network, it is crucial for the training to beat vanishing/exploding gradient. If it occurs it affects the whole network.

To solve this problem one can utilize an activation function whose derivative is bounded and therefore the gradient will not blow up or explode. This can affect the network ability to solve a problem due to the fact that such activation function may obstruct the training. Instead of using such activation function one may try to design the network differently. More clever neurons have been developed to solve this problem. The most popular is a "Longs Short Term Memory"-network (LSTM-network). These networks are, while back propagating, using adding operations and where multiplication is needed the network is using the identity. Which derivative is 1 by construction. These networks are hard to understand due to every neuron in an LSTM-network is in its turn a small network itself. We encourage the reader to seek information elsewhere because these networks are very powerful and do not need as much data as a feedforward network.

One detects vanishing gradient when training does not lower the test score.

4.3 Overfit

Overfit occurs when the network memorizes the data pattern and not comprehend the data pattern. Overfit is detected by training score decreases while the test score increases. The best solution to this is to feed more data to the network. More data is in some cases not available and therefore other methods have to be selected. One may have to change the network design. A simpler

network may beat overfitting. This must be balanced well so the reversed problem does not occur. This is the problem underfitting.

4.4 Dropout

Dropout simply refers to drop random neurons in an epoch. This implies that each neuron do not get updated every epoch and therefore the risk of overfitting decreases.

4.5 Regularization

Another way to combat overfitting is to add penalties to too strong signals. Consider again the MSE cost function with added a penalty:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}((z_i - y_i)^2) + \lambda \sum_{i=1}^{n} w_i^2$$

where λ is our penalty, w is our corresponding weight and z is the network output. Adding a penalty to too big signals and therefore their feature is not taken in too large consideration. This method is called regularization.

5 Universal Approximation Theorem

One classic theorem when it comes to the theory of neural networks is the *"Universal Approximation Theorem"*. Universal Approximation Theorem was proven by George Cybenko in 1989 (10). George Cybenko stated that the *Universal Approximation Theorem* only is true for Sigmoid activation function. In 1991 Kurt Hornik showed that *Universal Approximation Theorem* is true for an arbitrary activation function (10). *Universal Approximation Theorem* states that a neural network is a universal approximator, i.e. it can approximate an arbitrary function (10). The theorem is presented in this thesis for motivating the use of a neural network as an approximator in our case.

Theorem 5.1 *(Universal Approximation Theorem (10)) Let $g(\cdot)$ be an arbitrary activation function. let $X \subseteq \mathbb{R}^m$ and X compact. The space of continuous functions on X is denoted $C(X)$. Then $\forall f \in C(X)$, $\forall \epsilon > 0 : \exists n \in N, w_{ij}, b_i, \phi_i \in \mathbb{R}$, $i \in \{1, .., n\}$, $j \in \{1, .., m\}$, such that:*

$$F_{(n)}(x_1, x_2, ..., x_m) = \sum_{i=1}^{n} \phi_i g(\sum_{j=1}^{m} w_{ij}x_j + b_i) \tag{27}$$

as an approximation of the function $f(\cdot)$;

$$||f - F_{(n)}|| < \epsilon$$

With a compact space we mean a space closed and bounded. Here subscript n on the network, $F(\cdot)$. n denotes the number of neurons the network consists of.

Again, consider the input signal:

$$a_j = \sum_i w_{ji}\hat{y}_i + w_{j0}$$

If we wrap this signal in the activation function, $g(\cdot)$ we get:

$$g(\sum_i w_{ji}\hat{y}_i + w_{j0})$$

and then sum all of these with a scalar we obtain:

$$\sum_j \phi_j g(\sum_i w_{ji}\hat{y}_i + w_{j0})$$

This is the same approximator used in *Universal Approximation Theorem*. Here, i and j have changed places because of earlier notation, (see Equation (2)).

6 Comparison Model

For comparison a method involving a Monte Carlo simulation of a Brownian Motion is used below. A Geometric Brownian motion is used for modeling uncertainty and one of the fundamental equation for asset prices (2). The use of this method was inspired of the article "Modeling the price of Bitcoin with geometric fractional Brownian motion: a Monte Carlo approach" by Mariusz Tarnopolski.

6.1 Geometric Brownian Motion:

A Geometric Brownian Motion is described as follows:

$$\begin{aligned} dX_t &= \mu X_t dt + \sigma X_t dW_t \\ X_0 &= x_0 \end{aligned} \tag{28}$$

The solution to this equation is known,

$$X(t) = x_0 \exp\left\{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)\right\} \tag{29}$$

$X(t)$ denotes the stock price at time point t, μ and σ are constants, μ is called drift parameter and σ is called volatility parameter. μ is the expected value and approximated as the arithmetic mean of the data ($\hat{\mu}$) and σ is the standard deviation calculated as:

$$\sigma = s = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \hat{\mu})^2}{n-1}}$$

x_i is the observed value obtained from data. W_t is a sample from the normal distribution with variance σ^2 and expected value zero.

μ and σ^2 are calculated from the variation in data, i.e the return between data points. Calculated as follows:

$$Return = \log\big(\frac{x_i}{x_{i-1}}\big)$$

Where x_i is the observed data.

Equation (29) is used for pricing Bitcoin using a Monte Carlo-approach. Meaning, the Brownian motion is simulated N times, where N denotes a constant that the user choose, i.e the number of paths. The expected value of the predicted stock is calculated as the median of the values estimated at desired time step. The expected value is evaluated as the median comes from the fact that the return levels are log-normal distributed (2).

6.2 Pseudocode for Brownian Simulation

Algorithm 3 Algorithm for producing n paths of a Brownian motion.

Result: n paths for stock pricing
X_0 =initial value
number of simulation = N #set to an integer n.
simulation = 0
Last T = a # last time step for prediction.
while *simulation* $\leq$ *number of simulations* **do**
  predicted prices = []
  T = 0
  t = 1 #time step

  while $T \leq$ *last* T **do**
    $X_T = X_0 exp\left[\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma\epsilon\sqrt{t}\right]$
    X_T append to predicted prices
    T= T + 1

  end
end

7 Rolling Prediction Forecast

Neural networks are, today, used for stock prediction based on its well ability for non-linear mapping between input and output (5). This property makes a neural network suitable for time series analysis.

Rolling prediction forecast comes from using the predictions recursively, as follows:

$$\begin{aligned} F(X_1, X_2, ...X_t) &= X_{t+1} \\ F(X_1, X_2, ...X_t) &= \hat{X}_{t+1} \\ F(X_2, ...X_t, \hat{X}_{t+1}) &= \hat{X}_{t+2} \\ F(X_3, ...X_t, \hat{X}_{t+1}, \hat{X}_{t+2}) &= \hat{X}_{t+3}... \end{aligned} \tag{30}$$

7.1 Algorithm

Below a proposed algorithm for making a rolling prediction forecast will be presented. Here, $F(\cdot)$ is the network which performs the prediction. The network is written in Keras and TensorFlow. Keras and TensorFlow are a machine learning software available for the programming language Python.

Algorithm 4 Algorithm for a rolling prediction forecast

Result: Prediction for k time points
Require:
Data, $\vec{X}$
Prediction length, k
$t = 1$
while $t \leq k$ **do**
  $\hat{X}_{t+1} = F(\vec{X})$
  $\vec{X} =$ *append* $\hat{X}_{t+1}$ *to* $\vec{X}$
  $\vec{X} =$ *Drop first entry in* $\vec{X}$
  $j = j + 1$
end
Return last k entries of $\vec{X}$

8 Result

Training, testing and prediction result will be presented for the different methods and afterwards compared to the true outcome. Training score is the score the network achieves on training data, i.e. the data where the network is performing the training. This means that in the rolling prediction forecast the algorithm iterates over target data and not predicted data. Test score refers to the score achieved on testing, sometimes called validation. This measures how well the network performs on unseen data. This data still has a target value, meaning we can measure how well the network performs. During testing the network still iterates over true data. First when making forecasts about the future the algorithms iterates over predicted data points.

The score is measured using the function Root Mean Squared Error, "RMSE".

$$RMSE = \sqrt{\frac{1}{n}\sum_{i}^{n}(\hat{X}_i - y_i)^2} \tag{31}$$

Where $\hat{X}_i$ denotes network output. y_i denotes the target.

8.1 Train and Test Score

Below the different score will be presented. Further, plots will be presented to indicate if the regression performs well or not.

Table 1: Score, the square root of the MSE.

Currency	**Epochs**	**Train Score**	**Test Score**	**Samples**
Bitcoin	10	0.00321	0.0436	1811
Bitcoin	50	0.00285	0.0490	1811
Ethereum	10	0.01175	0.0774	980
Ethereum	50	0.00913	0.0596	980
Litecoin	10	0.00555	0.0525	1811
Litecoin	50	0.00443	0.0502	1811

Looking at the result from table 1, we see the score obtained from training and testing using different numbers of epochs, 10 and 50. Worth noticing is that the score increases with less data (score obtained from Ethereum). This seems intuitively correct due to the network has less training experience.

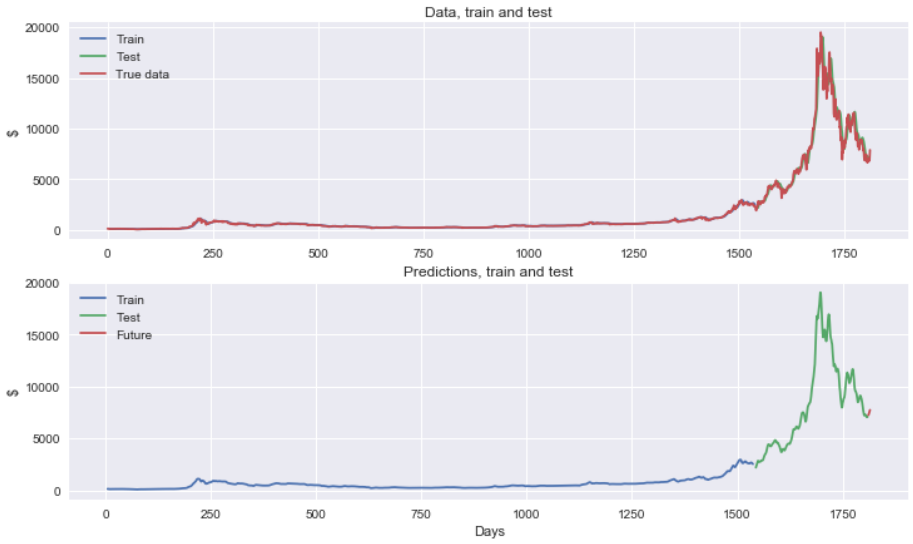


Figure 5: Network output on Bitcoin data (1811 samples). Top plot: network output compared with true data. Red curve is the true data. Blue curve is the network output on the train set (85 % of the data). network obtained train score, 0.00321 after ten epochs (measured using the root mean squared error). Indicating that the network trained well. Green curve: network output on test set. Test score: 0.0436. Another indication that the network performed well is that we almost do not see the network output, it is almost exactly looking as the red curve. Bottom plot: Only network output. Blue curve: network output on train set. Green curve: network output on test set. Orange curve: predictions for the five following days (see Section "Predictions").

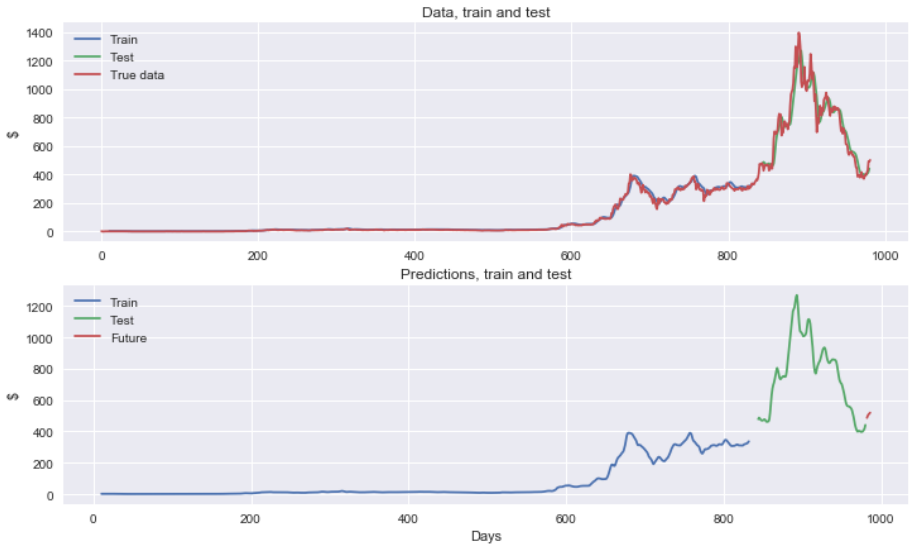


Figure 6: Network output on Ethereum data (980 samples). Top plot: network output compared with true data. Blue curve is the network output on the train set (85 % of the data). network obtained train score 0.00913 after ten epochs (measured using the root mean squared error). Green curve: network output on test set. Test score: 0.0596. An indication that the network performed well is that we almost do not see the network output, it is almost exactly looking as the red curve. Bottom plot: Only network output. Blue curve: network output on train set. Green curve: network output on test set. Orange curve: predictions for the five following days (see Section "Predictions").

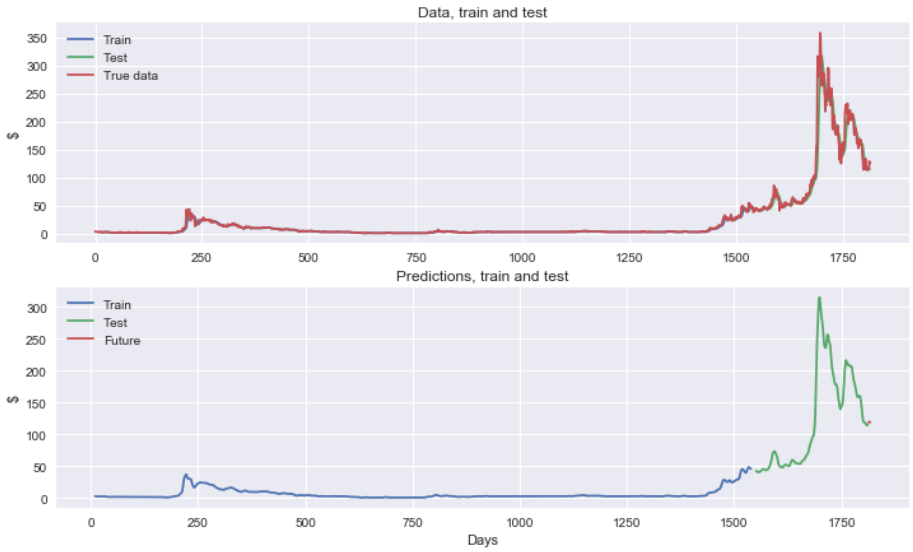


Figure 7: Network output on Litecoin data (1811 samples). Top plot: network output compared with true data. Blue curve is the network output on the train set (85 % of the data). Obtained train score, 0.00443 after ten epochs (measured using the root mean squared error). Green curve: network output on test set. Test score: 0.0502. An indication that the network performed well is that we almost do not see the network output, it is almost exactly looking as the red curve. Bottom plot: Only network output. Blue curve: network output on train set. Green curve: network output on test set. Orange curve: predictions for the five following days (see Section "Predictions").

9 Predictions

From Section "Rolling Prediction Forecast" an algorithm for prediction was presented. In this section, the result will be presented. The predictions are for five days ahead. Here, as stated in section "Rolling Predictions Forecast" the algorithm is iterating over predicted values.

9.1 Result from Rolling Prediction Forecast

Table 2: True and predicted prices from 18-04-14 to 18-04-18. In this table we only take the neural network-method predictions in consideration.

Currency	Price (18-04-14), USD	Price (18-04-18), USD
Bitcoin	7984.24	8164.42
Ethereum	501.48	524.79
Litecoin	126.29	140.00
Currency	Predicted price (18-04-14), USD	Predicted price (18-04-14), USD
Bitcoin	8000	8710
Ethereum	442.5	464
Litecoin	122.5	129

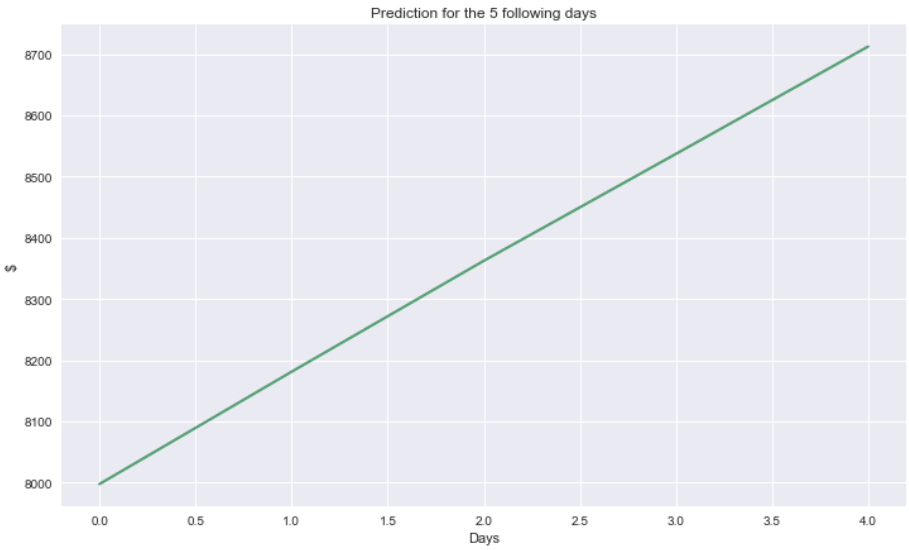


Figure 8: Bitcoin: Predictions for the five following days, (April 14 to April 18, 2018). Network predict an increase of 10.89%. True change: 2.21%.

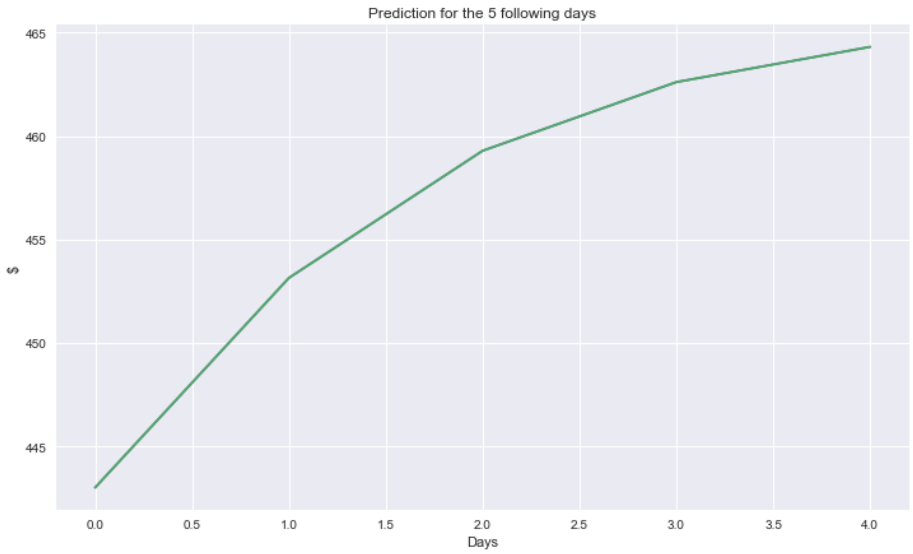


Figure 9: Ethereum: Predictions for the five following days, (April 14 to April 18, 2018). Network predict an increase of 4.90%. True change: 4.60%

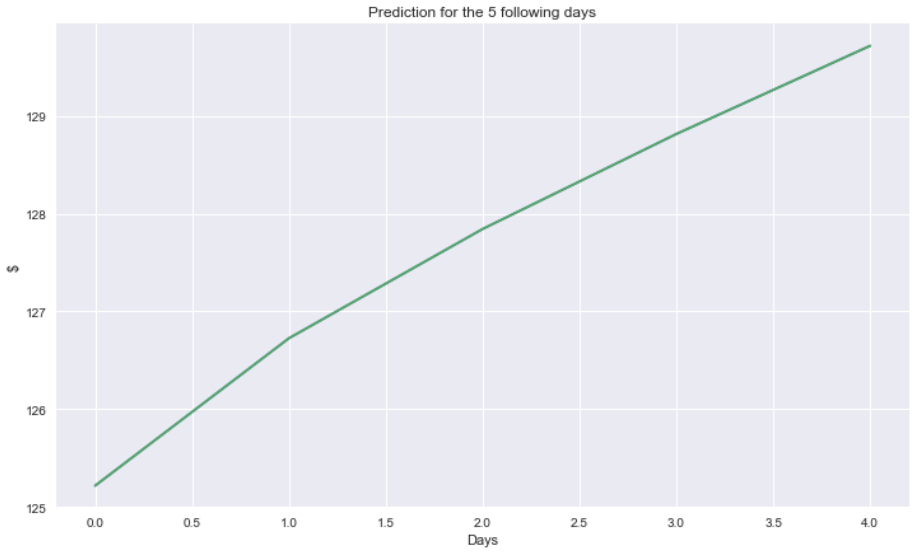


Figure 10: Litecoin: Predictions for the five following days, (April 14 to April 18, 2018). Network predict an increase of 5.31%. True change: 10.85%.

9.2 Comparison Method

15000 simulated Geometric Brownian Motions are simulated and then the median is calculated. The median is the expected price. Below a plots and result will be presented.

9.2.1 GBM Result

Table 3: Predicted price (pred.), predicted change (pred. change) and true change.

Currency	Pred. (18-04-14)	Pred. (18-04-18)	Pred. change (%)	True change (%)
Bitcoin	7986.00	8052.41	0.832 %	2.21 %
Ethereum	501.000000	508.823070	1.5614 %	4.60 %
Litecoin	126.00	125.873274	-0.1005 %	10.85 %

9.2.2 Bitcoin GBM Plots

Below the paths, prediction and histogram of the Bitcoin will be presented.

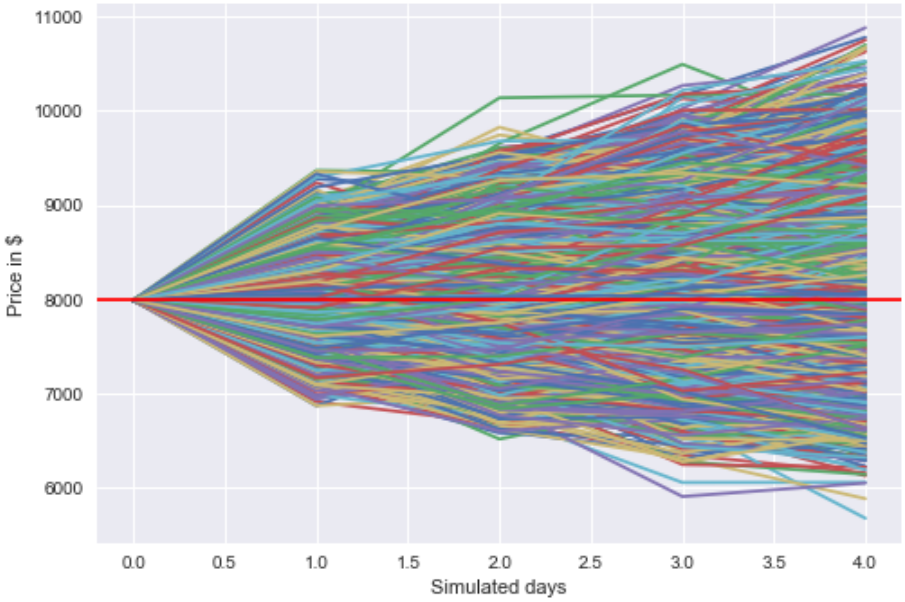


Figure 11: Bitcoin: 15000 paths of GBM. Red line is last known price

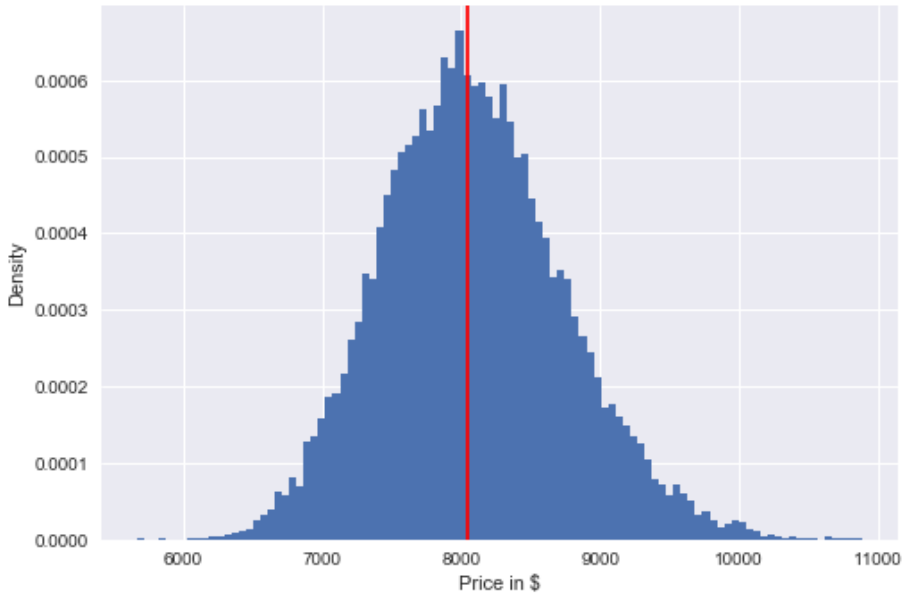


Figure 12: Bitcoin: Histogram over last price. Red line is most likely price 18/4.

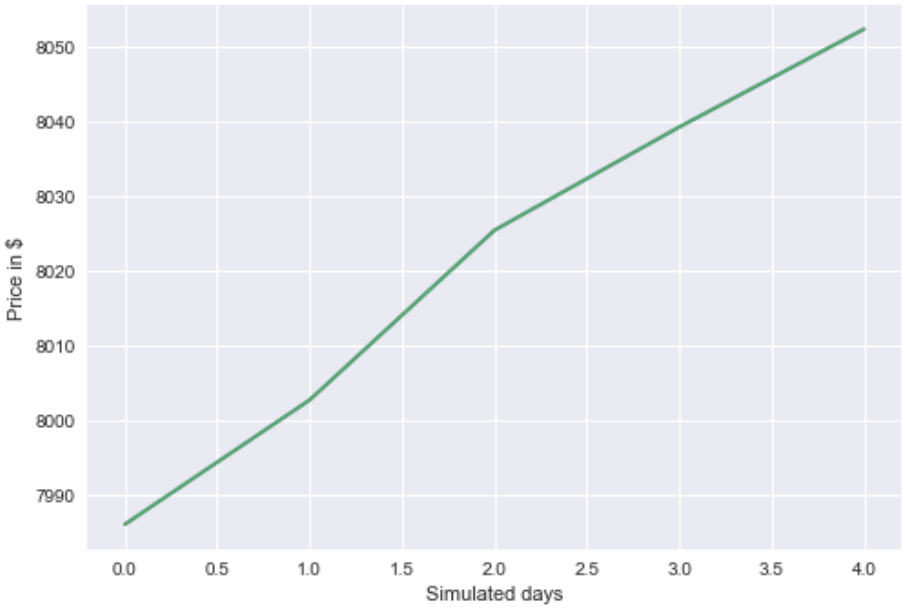


Figure 13: Bitcoin: Prediction. Most likely prices from 14.04 - 18.04

9.2.3 Ethereum GBM Plots

Below the paths, prediction and histogram of the Ethereum data will be presented.

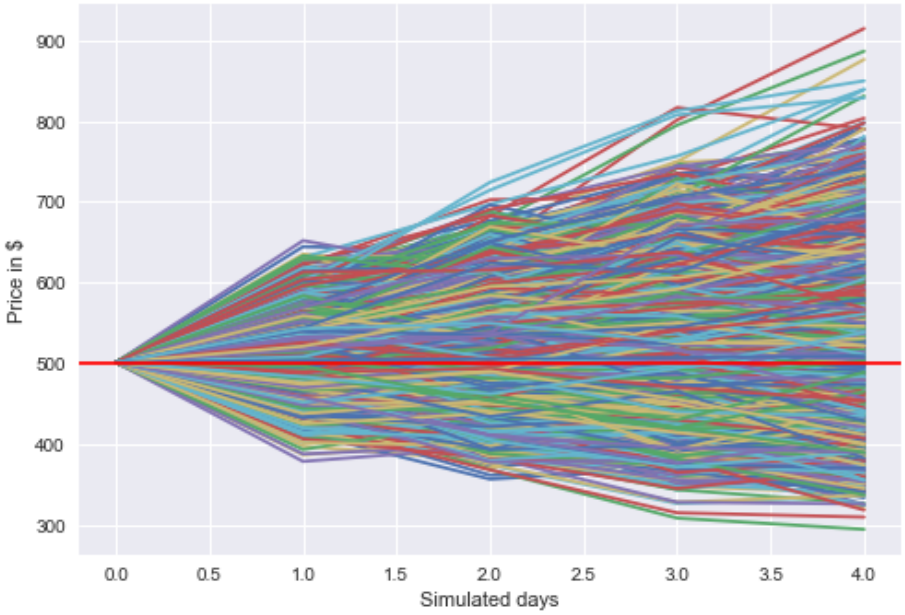


Figure 14: Ethereum: 15000 paths of GBM.

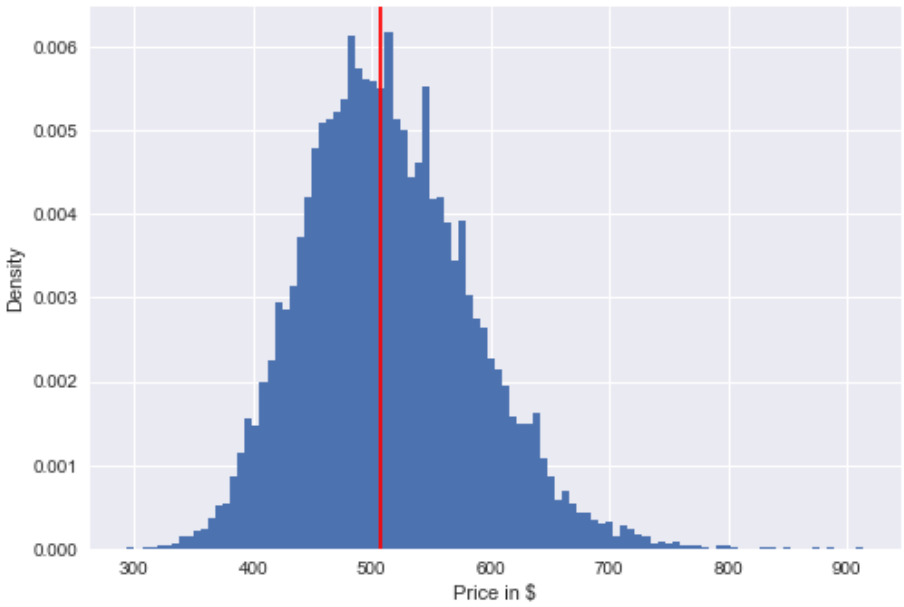


Figure 15: Ethereum: Histogram over last price. Red line is most likely price 18/4.

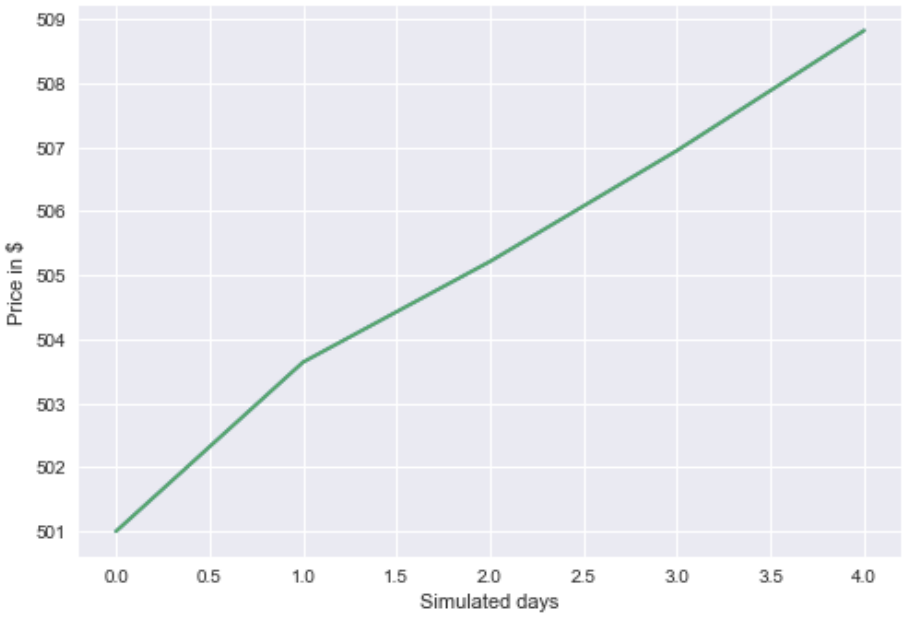


Figure 16: Ethereum: Prediction. Most likely prices from 14.04 - 18.04

9.2.4 Litecoin GBM Plots

Below the paths, prediction and histogram of the Litecoin data will be presented.

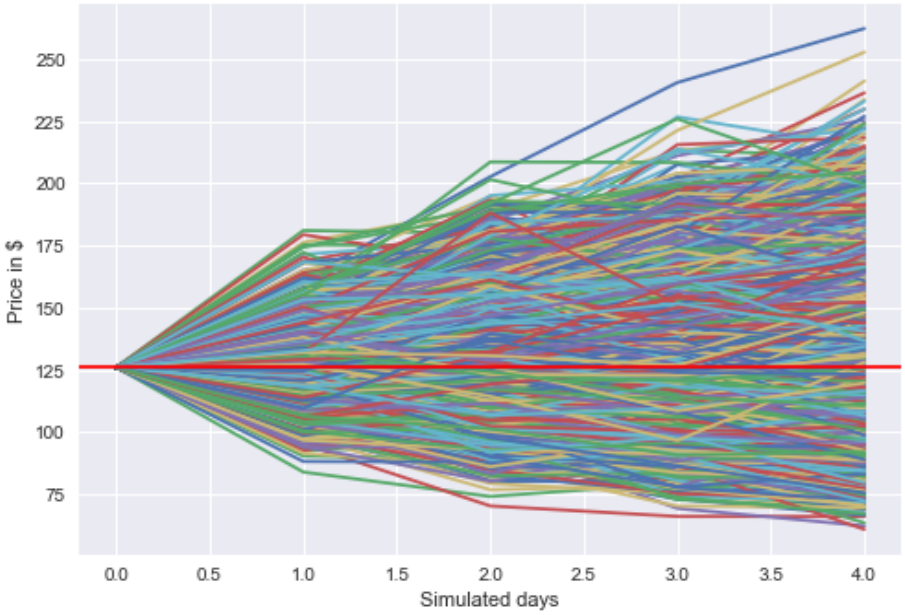


Figure 17: Litecoin: 15000 paths of GBM.

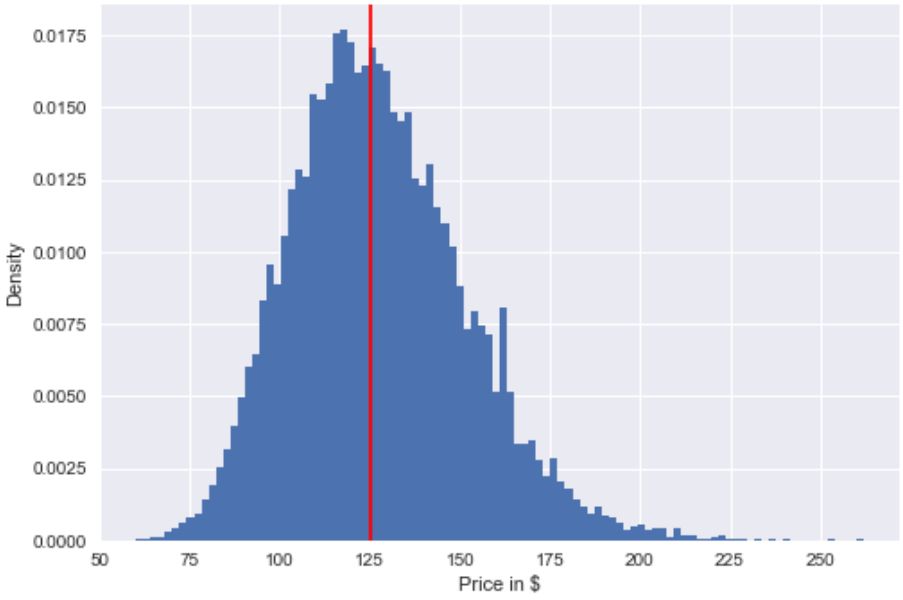


Figure 18: Litecoin: Histogram over last price. Red line is most likely price 18/4.

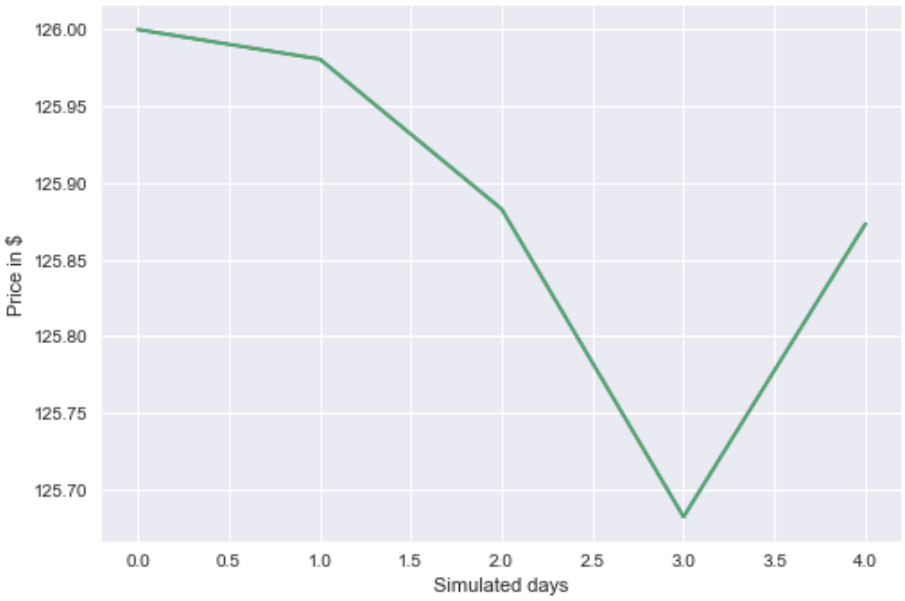


Figure 19: Litecoin: Prediction. Most likely prices from 14.04 - 18.04

9.3 Neural Network Prediction versus True Change

Below a table of predicted changes and true changes will presented.

Table 4: True and predicted prices from 18-04-14 to 18-04-18. Only the neural network-method predictions in consideration.

Currency	Price (18-04-14), USD	Price (18-04-18), USD
Bitcoin	7984.24	8164.42
Ethereum	501.48	524.79
Litecoin	126.29	140.00
Currency	Predicted price (18-04-14), USD	Predicted price (18-04-14), USD
Bitcoin	8000	8710
Ethereum	442.5	464
Litecoin	122.5	129

Table 5: True and predicted change. Only the neural network-method in consideration.

Currency	True change (%)	Predicted change (%)
Bitcoin	2.21%	10.89%
Ethereum	4.60%	4.90%
Litecoin	10.85%	5.31%

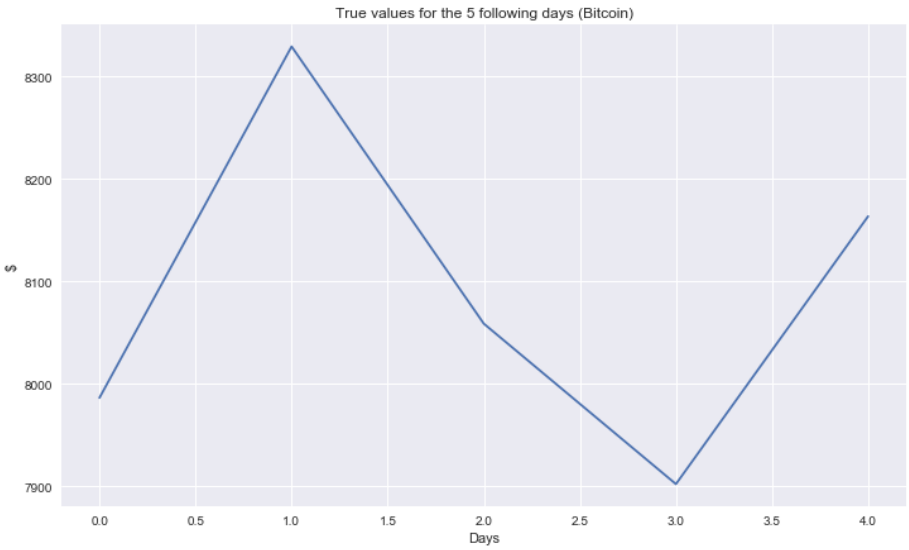


Figure 20: Bitcoin: True change for the five following days, (April 14 to April 18, 2018). True change: 2.21%. Predicted 10.89%. The network could not predict the drop after one day. Looking at the plots in Section "Train and test score" the predicted values have a more smooth curve which indicates that the network seem to have problem detecting daily trends.

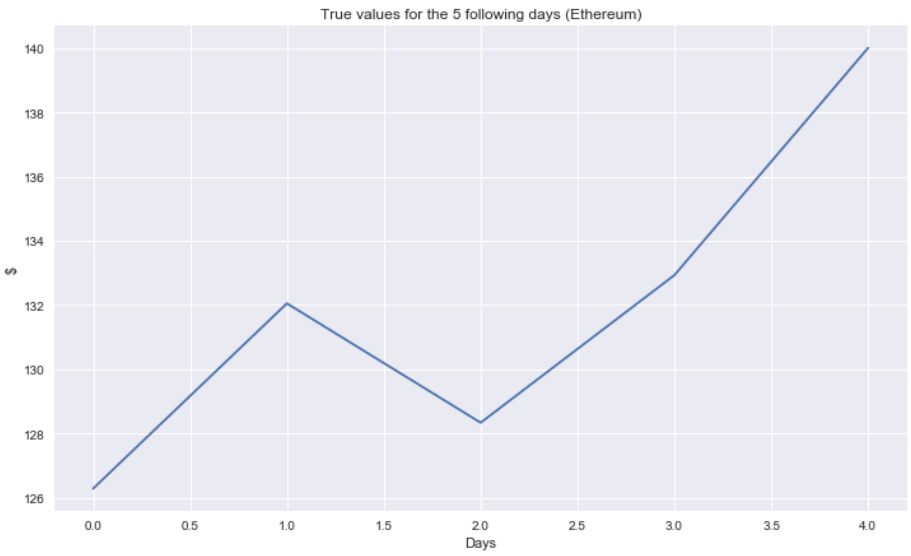


Figure 21: Ethereum: True change for the five following days, (April 14 to April 18, 2018) True change: 4.60%. Predicted change: 4.90%. The network has problem detecting daily trends.

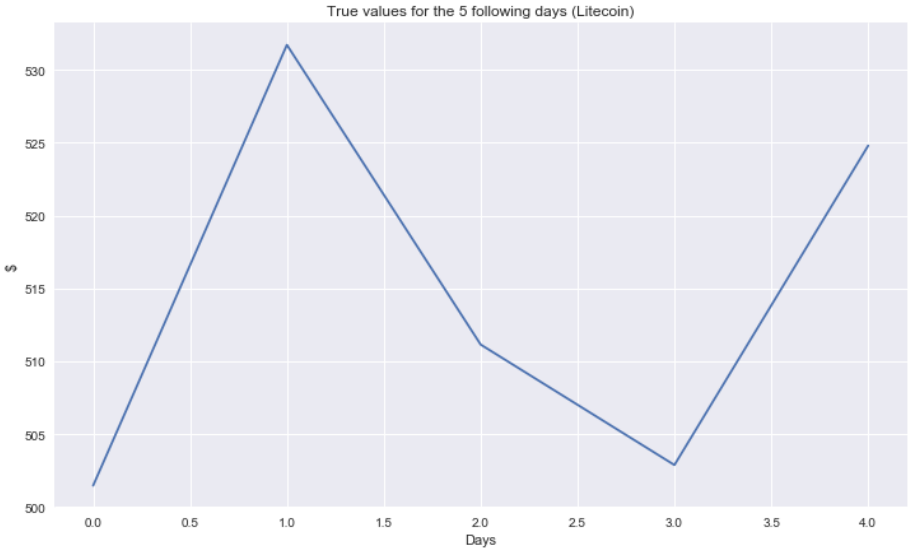


Figure 22: Litecoin: True change for the five following days, (April 14 to April 18, 2018). True change: 10.85%. Predicted change 5.31%. The network has problem detecting daily trends.

10 Comparison

Here a table will be presented were the predicted change will be compared to each other, i.e the GBM prediction and neural network prediction, and the true change.

Table 6: Neural network (NN) and Geometric Brownian Motion (GBM) change and true change from April 14 to April 18.

Currency	NN change (%)	GBM change (%)	True change (%)
Bitcoin	10.89%	0.832 %	2.21 %
Ethereum	4.90%	1.5614 %	4.60 %
Litecoin	5.31%	-0.1005 %	10.85 %

In Table 6 the predicted changes from both methods and also the true changes from April 14 to April 18, 2018 are presented. In all cases the GBM made a too low prediction but predicted an increase with success in two cases. In one case the GBM predicted a negative change, although the currency did increase with 10.85%. The neural network prediction predicted an increase in all cases, even though the predictions are too small or too big.

11 Conclusion

From this result, we can tell that a machine learning algorithm involving neural networks may be used to comprehend the data of a time series such as the closing price of a cryptocurrency. If the algorithm can be used for forecasting the price for five days ahead is too early to say. The algorithm must be evaluated over time to investigate the ratio of true prediction versus false prediction. If the algorithm can predict an increase or decrease with an accuracy larger than 50% we believe it is a successful algorithm. The author does not believe that just because the algorithm predicted an increase in all cases and it came true in all cases it is not an indication that the algorithm may be used successfully for trading cryptocurrencies. It may just be, and probably is, a lucky shot.

11.1 Further work

The algorithm can be improved. Now, the algorithm only takes the closing price as an input variable but, more variables should be used for evaluating the closing price. For example, market volume and open price. During the work with the algorithm, a classification algorithm was developed to predict a probability which indicated an increase or decrease. Perhaps that would be more interesting than an algorithm constructed to predict the exact price. In this thesis a regression method was used based on the possibility to visualize the approximation.

Also, it is important to notice that the price of an asset such as cryptocurrencies depends on much more variables than earlier data points. Some of these variables seem to be impossible to use for modeling the closing price. Such as political instability etc.

Such a algorithm will be developed and actual trading will be investigated with more common assets such as stocks.

12 References

References

[1] Jeff Heaton. *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks.* Heaton Research Inc, St. Louis, 2015.

[2] Tomas Björk. *Arbitrage Theory in Continuous Time* Oxford University Press, 1998

[3] Mariusz Bernacki and Przemysław Włodarczyk. *Principles of training multi-layer neural network using backpropagation.* `http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html`, September 2004.

[4] Xiao Ding, Yue Zhang, Ting Liu and Junwen Duan. *Deep Learning for Event-Driven Stock Prediction.* Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China, and Singapore University of Technology and Design, Singapore. 2015. p. 2327- 2333.

[5] Takashi Kimoto, Kazuo Asakawa, Morio Yoda and Masakazu Takeok *Stock Market Prediction System with Modular Neural Networks.* INVESTMENT TECHNOLOGY RESEARCH DIVISION. Tokyo, Japan.

[6] Grzegorz Swirszcz, Wojciech Marian Czarnecki and Razvan Pascan *Local Minima in Training a Neural Network*, London, UK

[7] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn TensorFlow* O'REILLY, 2017

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2009. p. 225-284

[9] Diederik P. Kingma and Jimmy Lei B.
ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. University of Amsterdam, OpenAI and University of Toronto. 2015

[10] Balázs Csanád Csáji. *Approximation with Artificial Neural Networks.* Eötvös Loránd University, Hungary

[11] Mariusz Tarnopolski, *Modeling the price of Bitcoin with geometric fractional Brownian motion: a Monte Carlo approach.* Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Krakow, Poland Available at: `https://arxiv.org/pdf/1707.03746.pdf`

[12] Brown, Brewer, Kevin D.; Feng, Yi; and Kwan, Clarence C. Y. (2012) *Geometric Brownian Motion, Option Pricing, and Simulation: Some Spreadsheet-Based Exercises in Financial Modeling, Spreadsheets in Education (eJSiE): Vol. 5: Iss. 3, Article 4.* Available at: `http://epublications.bond.edu.au/ejsie/vol5/iss3/4`

Bachelor's Theses in Mathematical Sciences 2018:K15
ISSN 1654-6229

LUNFNA-4020-2018

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/