# Predicting Bitcoin, Ethereum and XRP prices using a Neural Network

Anouk Montfoort (0690708)

January 31, 2025

## 1 Introduction

With Bitcoin reaching a price higher than 100.000 dollars in 2024, its all-time peak, it seems lucrative to invest in cryptocurrencies (Yahoo Finance, n.d.). Rather than buying low and selling high, a more strategic approach using mathematical models or so-called algorithmic trading is used in the report. This report discusses a trading strategy based on the prediction of cryptocurrencies prices using a neural network (NN). The advantage of NNs is their ability to analyze complex relationships without needing detailed knowledge of the underlying structure (Penm, Chaar & Moles, 2013). However, this ability also presents a disadvantage: deep learning models are less interpretable than classical mathematical models and are often considered "black-box" models, making it challenging to understand the specific relationships they learn between inputs and outputs.

The development of NNs began in 1943, when neurophysiologist McCulloch and mathematician Pitts created an early NN model using electrical circuits (Foote, 2021). This research was further developed by psychologist Hebb, who discovered that repeated stimulation of a neuron could strengthen its response (Foote, 2021). NNs are designed to mimic the human brain, which is a network consisting of interconnected neurons. The human brain learns by repetitively being exposed to the same stimulus and therefore altering the strength of synaptic connections between neurons. Similarly, NNs learn by adjusting the weights between neurons until the network's output closely matches the desired target value.

The overall goal of this research is to get a clear methodology to predict the price of cryptocurrencies based on a NN, so that insights into buying and selling opportunities are given. In this report, the cryptocurrency prices of Bitcoin, Ethereum and XRP are predicted while minimizing the error or loss of the prediction. The trading strategy focuses solely on if there is a trading opportunity (if it is attractive to buy or sell) and not how much should be traded.

The remainder of this report is structured as follows: Section 2 provides a detailed theoretical background on the methods and algorithms, Section 3 describes the design, implementation and tutorial of the algorithm, Section 4 summarizes the used data, Section 5 describes the main findings and Section 6 summarizes insights and conclusions drawn from the results.

## 2 Literature

### 2.1 Structure of the NN

This Section discusses the Neural Network (NN) in general form based on Hastie, Tibshirani and Friedman (2009), Nielsen (2015) and Taddy (2019). A neural network consists, similar to a human brain, of several neurons. In this report a multilayer NN, shown in Figure 1, consists of an input layer of $d$ normalized inputs, $L$ hidden layers and an output layer of 1 output.

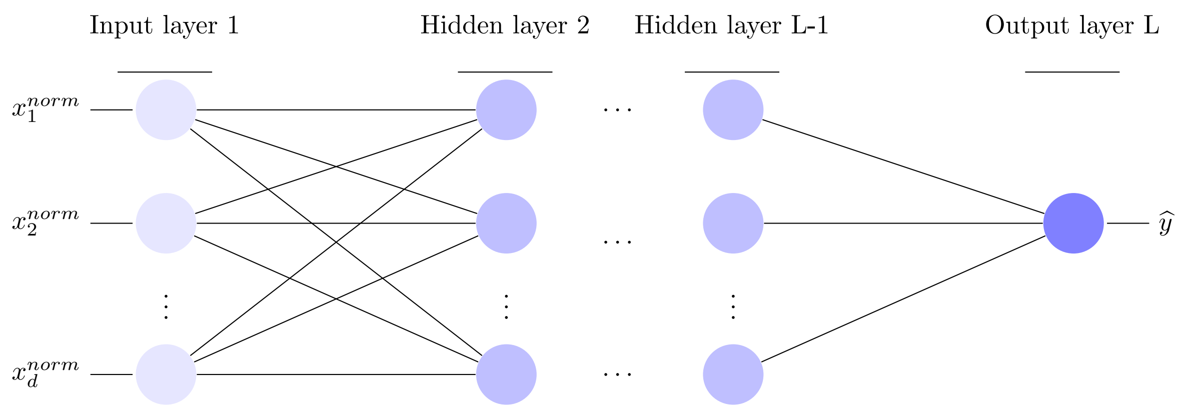


Figure 1: A multilayer NN having an input layer of $d$ inputs, $L$ hidden layers and an output layer of $t$ outputs.

## 2.2 Training of the NN

In short, the definition of training a NN is teaching the NN to make accurate predictions by adjusting the parameters (weights and biases) of the NN. In the first epoch (one cycle trough training the dataset of a NN) the weights of each input are randomly assigned. In this report training of the NN is done by using forward propagation, Back-Propagation and Gradient Descent.

## 2.3 Forward propagation (prediction)

Forward propagation (FP) is performed to compute the predicted output given the inputs. In formula form, at each neuron in a input layer, hidden layer or output layer, information will be transformed via a weighted sum in layer $l$ over all inputs for that neuron plus a bias term:

$$Z^l = W^l A^{l-1} + b^l, \tag{1}$$

where $Z^l$ is the weighted sum of layer $l$, $W^l$ is the weight matrix for layer $l$, $A^{l-1}$ is the activation function of layer $l-1$ or the raw input data in the first layer, $b^l$ is the bias vector for layer l. Above equation 1 is passed through an activation function according to

$$A^l = \delta\big(Z^l\big), \tag{2}$$

where $A^l$ equals the output of layer $l$ for layers $l = 1, ...L-1$ and $\delta$ is the activation function. The output for output layer $L$ is equal to: $\widehat{y} = \delta\big(Z^L\big)$. An activation function is a non-linear function that is able to explain complex relations (Taddy, 2019). This report investigates activation functions hyperbolic tangent, sigmoid and ReLu. More on this in Section 2.7.

## 2.4 Back-Propagation

Back-Propagation (BP) is adjusting parameters based on the gradient or derivative of the loss function, meaning that BP computes the gradient of the loss function $LO$ with respect to the parameters of each layer. The chain rule is used to go backwards from the output layer $L$ through the hidden layers $L-1, ..., 2$ to the input layer 1.

In this report the loss function, which measures how well the predicted output matches the true output, is equal to the mean squared error (MSE):

$$LO = \frac{1}{N}\sum_{i=1}^{N}(\widehat{y} - y)^2 \tag{3}$$

where $N$ equals the total number of data points, $\widehat{y}$ equals the final predicted output of the NN in the output layer and $y$ equals the actual value.

The gradient of the loss $LO$ with respect to the weighted sum of output layer $L$ equals:

$$\frac{\partial LO}{\partial Z^L} = \frac{\partial LO}{\partial A^L}\frac{\partial A^L}{\partial Z^L} = (\widehat{y} - y)\delta'(Z^L), \tag{4}$$

The gradient of the loss $LO$ with respect to the weights and biases at the output layer $L$ equal:

$$\frac{\partial LO}{\partial W^L} = \frac{\partial LO}{\partial Z^L}(A^{L-1})^T, \tag{5}$$

$$\frac{\partial LO}{\partial b^L} = \frac{\partial LO}{\partial Z^L}, \tag{6}$$

Then, the gradients are backpropagated to previous hidden layers. This means that for hidden layers $l$ with $L-1$ to 1, the gradient of the loss $LO$ with respect to $Z^l$ equals (using the chain rule):

$$\frac{\partial LO}{\partial Z^L} = (W^{l+1})^T \frac{\partial LO}{\partial Z^{l+1}} \delta'(Z^l) \tag{7}$$

Then, at each hidden layer gradient for weights and biases are computed:

$$\frac{\partial LO}{\partial W^l} = \frac{\partial LO}{\partial Z^l}(A^{l-1})^T, \tag{8}$$

$$\frac{\partial LO}{\partial b^l} = \frac{\partial LO}{\partial Z^l}, \tag{9}$$

The gradients of the loss function with respect to parameters weights and biases tells us how much the loss will increase or decrease if the parameter is changed (how sensitive it is to the parameter). This means that if in the opposite direction of the gradient will be moved, the loss function will be minimized.

## 2.5 Gradient Descent

If the gradients are computed, the parameters are updated in the direction that minimizes the loss function $LO$ using Gradient Descent (GD) according to:

$$w^L = w^L - \eta \frac{\partial LO}{\partial w^L} \tag{10}$$

$$b^L = b^L - \eta \frac{\partial LO}{\partial b^L} \tag{11}$$

In above formulas $\eta$ is the learning rate, which determines the step size of each iteration of GD. It determines the speed at which will be moved in the direction of the negative gradient. If $\eta$ will be too small (too big), convergence will be small (the optimal solution may be passed). The procedure FP, BP and GD is repeated for multiple epochs until the parameters converge so that the loss function is minimized. By moving information in two directions, the NN learns and improves itself. The training procedure stops when the output is close enough to the desired value.

## 2.6 Training, validation and test dataset

The ability of NN to generalize or accurately predict the output on unseen test data is crucial for its performance. Overfitting, which occurs when the NN memorizes the training data but fails to generalize to new data, is a common issue (Nielsen, 2015). To prevent overfitting the normalized data set will be pre-processed by dividing the whole dataset into a training data set to adjust the weights of the NN, the validation data set to evaluate the performance based on different hyperparameters settings (namely: number of neurons in the input layer, number of hidden layers, number of neurons per hidden layer, the activation function and the learning rate) and the test data set to obtain the performance of the NN on new unseen (independent) data by using Time Series Splitting (Medium, 2021). After splitting the data into training, validation and test sets the hyperparameter setting that minimizes the validation error the most is determined on the validation data set via trial-and-error. Also, the algorithm contains early stopping procedure, which means that if the validation loss is not improving (so declining) in the following five epochs the training of the NN will stop earlier than the predefined number of epochs. Using the most optimal hyperparameter settings the NN is tested using the test data set and predictions of cryptocurrencies are made.

## 2.7 Activation function

As described in Section 2.3, this report investigates activation functions hyperbolic tangent (12), sigmoid (13) and rectified linear unit (14):

$$\delta(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{12}$$

$$\delta(x) = \frac{1}{1 + \exp(-x)} \tag{13}$$

$$\delta(x) = \max(0, x) \tag{14}$$

Rectified linear unit (ReLU) is often preferred over sigmoid and hyperbolic tangent (Tanh) because it is faster and avoids some issues (Nwankpa, Ijomah, Gachagan & Marshall, 2018). Tanh and sigmoid saturate, meaning that large values go to 1 for Tanh and small values go to $-1$ or 0 for sigmoid, and show only sensitivity for changes around their midpoints, which is 0.5 for sigmoid and 0 for Tanh (Goodfellow, Bengio & Courville, 2017). ReLU does not have this issue that weights in NN training barely change compared to the previous epoch and that the NN training can be completely stopped due to this. However, ReLu can suffer from a so-called "dying ReLu problem" (Leung, 2021) In some cases neurons can become inactive and can give 0 for every output. This actually means that the network is not learning anymore.

# 3 Python code

The multilayer NN to predict the cryptocurrency prices is programmed from scratch and no simplifying packages are used for the structure of the NN.

The Python code asks the user for input, regarding which cryptocurrency the user wants to investigate, the number of input neurons of the NN, the number of hidden layers, the number of neurons per hidden layer, the activation function, the learning rate and the number of epochs. This means that if the user gives these inputs, the Python code starts to run and the NN starts to train. The Python code stops if the maximum number of epochs is reached or due to early stopping if there is no improvement in validation loss. Based on the most optimal hyperparameter settings, NN will predict the cryptocurrency price, give the test loss and shows a graph of the actual data together with the predicted price.

## 4 Data

In this project *yfinance* library of Python is used to download the closing prices of cryptocurrencies in dollars from Yahoo Finance (Aroussi, 2020). More, specifically "BTC-USD", "ETH-USD" and "XRP-USD" are investigated, e.g. bitcoin price in US dollars, ethereum price in US dollars and XRP price in US dollars.

The input and output data in this project are simple. The input, which is the number of neurons in the intput layer, consists of a sequence of cryptocurrency prices over a period of X days, which is used to predict the price on the following X+1 day (the output). Because a NN is a black-box model and is challenging to interpret, it is chosen to have a simple data framework in this project. In Section 5.1, the results on different number of neurons in the input layer will be investigated.

## 5 Results

### 5.1 Obtaining hyperparameters on validation set and training set

After normalization of the inputs, the network is trained on several hyperparameter settings using the train and validation data sets. The search is limited to search within a maximum of four hidden layers and a maximum of 30 neurons per hidden layer. Adding more hidden layers and/or having more neurons in the network did not improve the performance substantially and overfitting is prevented. To obtain the hyperparameter setting with the lowest validation loss, the NN is trained on a maximum of 20.000 epochs and using early stopping.

The most optimal hyperparameter settings are selected by chosing the settings with the lowest validation loss. A lower validation loss means the NN generalizes better on unseen data. Also, the training loss is calculated in each epoch. If the training loss gets lower in every epoch, it means the NN is learning. To get the most optimal hyperparameter settings, first the number of neurons in the input layer is chosen. Thereafter, using the settings with the most optimal number of neurons in the input layer, the number of hidden layers and neurons per hidden layer are selected. Using this result, the best two hyperparameter settings are used to select the activation function. Finally, using the best two hyperparameter settings, the most optimal learning rate is selected. The two hyperparameter settings with the lowest validation loss over all different hyperparameter settings are selected to predict the cryptocurrency price on the test data set.

Via trial-and-error, there is found for BTC that:

1. The less neurons in the input layer, the lower the validation loss. This means that it is more optimal to predict the next days Bitcoin price using a smaller number of days. It seems best to predict the next day Bitcoin price on the prices of the previous 15 or 25 days.
2. Table 1 shows that more hidden layers is not always better. Too much neurons in each hidden layer makes the NN slow and does not always gives a small loss. The performance of the NN is best when having four hidden layers with each five neurons ([5,5,5,5]) or having two hidden layers with each twenty neurons ([20,20]).
3. Hyperbolic tangent with a learning rate of 0.01 has the lowest validation loss. Table 1 highlights the two NNs that will be used for prediction.

Table 1: Different hyperparameter settings for BTC.

| | Neurons in input layer | Neurons per hidden layer | Activation function | Learning rate | Validation loss | Train loss | Epochs |
|---|---|---|---|---|---|---|---|
| **Input layer** | 15 | [20,20,20] | Tanh | 0.01 | 0.0080 | 0.0087 | 2039 |
| | 25 | [20,20,20] | Tanh | 0.01 | 0.0163 | 0.0219 | 2149 |
| | 40 | [20,20,20] | Tanh | 0.01 | 0.4005 | 0.4872 | 41 |
| | 50 | [20,20,20] | Tanh | 0.01 | 0.1065 | 0.2036 | 80 |
| | 75 | [20,20,20] | Tanh | 0.01 | 0.1369 | 0.1874 | 303 |
| | 100 | [20,20,20] | Tanh | 0.01 | 0.4564 | 0.2738 | 76 |
| **Hidden layers** | 15 | [5,5] | Tanh | 0.01 | 0.0195 | 0.0276 | 1042 |
| | 25 | [5,5] | Tanh | 0.01 | 0.0035 | 0.0096 | 3058 |
| | 15 | [5,5,5] | Tanh | 0.01 | 0.2576 | 0.3164 | 66 |
| | 25 | [5,5,5] | Tanh | 0.01 | 0.0025 | 0.0054 | 10.573 |
| NN1 | 15 | [5,5,5,5] | Tanh | 0.01 | 0.0012 | 0.0024 | 20.000 |
| | 25 | [5,5,5,5] | Tanh | 0.01 | 0.0080 | 0.0112 | 1650 |
| | 15 | [10,10] | Tanh | 0.01 | 0.0024 | 0.0027 | 20.000 |
| | 25 | [10,10] | Tanh | 0.01 | 0.0035 | 0.0053 | 20.000 |
| | 15 | [10,10,10] | Tanh | 0.01 | 0.0063 | 0.0112 | 3709 |
| | 25 | [10,10,10] | Tanh | 0.01 | 0.0037 | 0.0053 | 4197 |
| | 15 | [10,10,10,10] | Tanh | 0.01 | 0.2363 | 0.1429 | 44 |
| | 25 | [10,10,10,10] | Tanh | 0.01 | 0.0160 | 0.0349 | 416 |
| NN2 | 15 | [20,20] | Tanh | 0.01 | 0.0012 | 0.0024 | 20.000 |
| | 25 | [20,20] | Tanh | 0.01 | 0.0024 | 0.0042 | 20.000 |
| | 15 | [20,20,20,20] | Tanh | 0.01 | 0.0326 | 0.0601 | 97 |
| | 25 | [20,20,20,20] | Tanh | 0.01 | 0.0212 | 0.0258 | 232 |
| | 15 | [30,30] | Tanh | 0.01 | 0.0322 | 0.0139 | 1936 |
| | 25 | [30,30] | Tanh | 0.01 | 0.4742 | 0.4829 | 7 |
| | 15 | [30,30,30] | Tanh | 0.01 | 0.0409 | 0.0572 | 190 |
| | 25 | [30,30,30] | Tanh | 0.01 | 0.4839 | 0.5340 | 10 |
| | 15 | [30,30,30,30] | Tanh | 0.01 | 0.7471 | 0.6254 | 13 |
| | 25 | [30,30,30,30] | Tanh | 0.01 | 0.5169 | 0.4797 | 7 |
| **Activation function** | 15 | [5,5,5,5] | Sigmoid | 0.01 | 0.0762 | 0.0655 | 20.000 |
| | 15 | [5,5,5,5] | ReLu | 0.01 | 0.0069 | 0.0216 | 134 |
| | 15 | [20,20] | Sigmoid | 0.01 | 0.0024 | 0.0034 | 20.000 |
| | 15 | [20,20] | ReLu | 0.01 | 0.1364 | 0.1454 | 5 |
| **Learning rate** | 15 | [5,5,5,5] | Tanh | 0.001 | 0.6313 | 0.5794 | 115 |
| | 15 | [20,20] | Tanh | 0.001 | 0.1651 | 0.1973 | 406 |
| | 15 | [5,5,5,5] | Tanh | 0.005 | 0.0018 | 0.0029 | 10.449 |
| | 15 | [20,20] | Tanh | 0.005 | 0.0034 | 0.0027 | 20.000 |
| | 15 | [5,5,5,5] | Tanh | 0.02 | 0.0038 | 0.0096 | 1090 |
| | 15 | [20,20] | Tanh | 0.02 | 0.0791 | 0.2095 | 13 |
| | 15 | [5,5,5,5] | Tanh | 0.05 | 0.0152 | 0.0235 | 220 |
| | 15 | [20,20] | Tanh | 0.05 | 0.1169 | 0.0926 | 14 |

Via trial-and-error, there is found for ETH that:

- For ETH 25 and 40 neurons in the input layer seem most optimal. It seems best to predict the next day Ethereum price on the prices of the previous 25 or 40 days.
- Table 2 shows that having five neurons in each hidden layers gives the lowest validation loss. Both two hidden layers and four hidden layers with five neurons per hidden layer have the lowest validation loss.
- Hyperbolic tangent with a learning rate of 0.01 gives the lowest validation loss in most cases. A lower or higher learning rate does not affect the validation loss substantially. A NN with 25 neurons in the input layer and two or four hidden layers with five neurons per hidden layer seem the most optimal. Also, ReLu with 25 input neurons and five neurons in two hidden layers shows a low validation loss (0.0094). Because ReLu often suffers from ”dying ReLu problem” and gets stuck in a local optimum, three instead of two NNs will be used for prediction. Table 2 highlights the three NNs that will be used for prediction.

Table 2: Different hyperparameter settings for ETH.

| | Neurons in input layer | Neurons per hidden layer | Activation function | Learning rate | Validation loss | Train loss | Epochs |
|---|---|---|---|---|---|---|---|
| **Input layer** | 15 | [20,20,20] | Tanh | 0.01 | 0.9844 | 1.0017 | 5 |
| | 25 | [20,20,20] | Tanh | 0.01 | 0.0590 | 0.2538 | 29 |
| | 30 | [20,20,20] | Tanh | 0.01 | 0.5471 | 0.3838 | 17 |
| | 40 | [20,20,20] | Tanh | 0.01 | 0.0265 | 0.0544 | 524 |
| | 50 | [20,20,20] | Tanh | 0.01 | 0.5533 | 0.4711 | 5 |
| | 60 | [20,20,20] | Tanh | 0.01 | 0.5467 | 0.5807 | 5 |
| | 75 | [20,20,20] | Tanh | 0.01 | 0.5122 | 0.4410 | 5 |
| | 100 | [20,20,20] | Tanh | 0.01 | 0.3122 | 0.3762 | 30 |
| **Hidden layers** NN3 | 25 | [5,5] | Tanh | 0.01 | 0.0097 | 0.0297 | 655 |
| | 40 | [5,5] | Tanh | 0.01 | 0.0138 | 0.0124 | 6574 |
| | 25 | [5,5,5] | Tanh | 0.01 | 0.3855 | 0.3912 | 11 |
| | 40 | [5,5,5] | Tanh | 0.01 | 0.0185 | 0.0264 | 403 |
| NN1 | 25 | [5,5,5,5] | Tanh | 0.01 | 0.0064 | 0.0140 | 423 |
| | 40 | [5,5,5,5] | Tanh | 0.01 | 0.8473 | 0.6619 | 6 |
| | 25 | [10,10] | Tanh | 0.01 | 0.5835 | 0.4431 | 63 |
| | 40 | [10,10] | Tanh | 0.01 | 0.0650 | 0.2138 | 23 |
| | 25 | [10,10,10] | Tanh | 0.01 | 0.1334 | 0.2488 | 39 |
| | 40 | [10,10,10] | Tanh | 0.01 | 0.1225 | 0.1921 | 24 |
| | 25 | [10,10,10,10] | Tanh | 0.01 | 0.0403 | 0.0684 | 361 |
| | 40 | [10,10,10,10] | Tanh | 0.01 | 0.0903 | 0.0138 | 5100 |
| | 25 | [20,20] | Tanh | 0.01 | 0.4224 | 0.4398 | 38 |
| | 40 | [20,20] | Tanh | 0.01 | 0.2080 | 0.4275 | 76 |
| | 25 | [20,20,20,20] | Tanh | 0.01 | 0.4161 | 0.4404 | 9 |
| | 40 | [20,20,20,20] | Tanh | 0.01 | 0.2798 | 0.3673 | 52 |
| | 25 | [30,30] | Tanh | 0.01 | 0.5527 | 0.4424 | 5 |
| | 40 | [30,30] | Tanh | 0.01 | 0.0656 | 0.0939 | 1046 |
| | 25 | [30,30,30] | Tanh | 0.01 | 0.6143 | 0.3722 | 35 |
| | 40 | [30,30,30] | Tanh | 0.01 | 0.3835 | 0.3870 | 29 |
| | 25 | [30,30,30,30] | Tanh | 0.01 | 0.7087 | 0.4157 | 21 |
| | 40 | [30,30,30,30] | Tanh | 0.01 | 0.6827 | 0.4900 | 10 |
| **Activation function** | 25 | [5,5] | Sigmoid | 0.01 | 0.0661 | 0.0501 | 20.000 |
| NN2 | 25 | [5,5] | ReLu | 0.01 | 0.0094 | 0.0295 | 2950 |
| | 25 | [5,5,5,5] | Sigmoid | 0.01 | 0.0747 | 0.0559 | 20.000 |
| | 25 | [5,5,5,5] | ReLu | 0.01 | 0.1048 | 0.1348 | 5 |
| **Learning rate** | 25 | [5,5,5,5] | Tanh | 0.001 | 0.4606 | 0.4431 | 101 |
| | 25 | [5,5] | ReLu | 0.001 | 0.0173 | 0.0325 | 6316 |
| | 25 | [5,5,5,5] | Tanh | 0.005 | 0.0776 | 0.0648 | 144 |
| | 25 | [5,5] | ReLu | 0.005 | 0.1048 | 0.1349 | 5 |
| | 25 | [5,5,5,5] | Tanh | 0.02 | 0.6898 | 0.8036 | 39 |
| | 25 | [5,5] | ReLu | 0.02 | 0.1328 | 0.1320 | 442 |
| | 25 | [5,5,5,5] | Tanh | 0.05 | 0.0189 | 0.0167 | 138 |
| | 25 | [5,5] | ReLu | 0.05 | 0.1048 | 0.1349 | 5 |

Via trial-and-error, there is found for XRP that:

- For XRP 15 and 25 neurons in the input layer are most optimal. It seems best to predict the next day XRP price on the prices of Ethereum of the previous 15 or 25 days.
- Table 3 shows that having four hidden layers with each five neurons or two hidden layers with each ten hidden layers is best: [5,5,5,5] and [10,10]
- Hyperbolic tangent in combination with 0.01 gives the lowest validation loss in most cases. Table 3 shows also that a NN with fiveteen input neurons, two hidden layers with each ten neurons and hyperbolic tangent with a learning rate of 0.01 gives the lowest validation loss of 0.0011. Also, a NN with the same structure but with a learning rate of 0.05 gives a low validation loss, namely 0.0012. Table 3 highlights the two NNs that will be used for prediction.

Table 3: Different hyperparameter settings for XRP.

| | Neurons in input layer | Neurons per hidden layer | Activation function | Learning rate | Validation loss | Train loss | Epochs |
|---|---|---|---|---|---|---|---|
| **Input layer** | 15 | [20,20,20] | Tanh | 0.01 | 0.1041 | 0.2414 | 43 |
| | 25 | [20,20,20] | Tanh | 0.01 | 0.0125 | 0.0107 | 5648 |
| | 40 | [20,20,20] | Tanh | 0.01 | 0.2398 | 0.2500 | 97 |
| | 50 | [20,20,20] | Tanh | 0.01 | 0.7786 | 0.7047 | 5 |
| | 75 | [20,20,20] | Tanh | 0.01 | 0.3347 | 0.3338 | 219 |
| | 100 | [20,20,20] | Tanh | 0.01 | 0.6095 | 0.6736 | 8 |
| **Hidden layers** | 15 | [5,5] | Tanh | 0.01 | 0.0042 | 0.0145 | 4217 |
| | 25 | [5,5] | Tanh | 0.01 | 0.0263 | 0.1477 | 231 |
| | 15 | [5,5,5] | Tanh | 0.01 | 0.0177 | 0.0561 | 44 |
| | 25 | [5,5,5] | Tanh | 0.01 | 0.0055 | 0.0306 | 86 |
| | 15 | [5,5,5,5] | Tanh | 0.01 | 0.0014 | 0.0028 | 20.000 |
| | 25 | [5,5,5,5] | Tanh | 0.01 | 0.0042 | 0.0065 | 6744 |
| NN1 | 15 | [10,10] | Tanh | 0.01 | 0.0011 | 0.0027 | 20.000 |
| | 25 | [10,10] | Tanh | 0.01 | 0.0299 | 0.0087 | 7639 |
| | 15 | [10,10,10] | Tanh | 0.01 | 0.0250 | 0.0327 | 126 |
| | 25 | [10,10,10] | Tanh | 0.01 | 0.0433 | 0.0750 | 318 |
| | 15 | [10,10,10,10] | Tanh | 0.01 | 0.4179 | 0.4284 | 6 |
| | 25 | [10,10,10,10] | Tanh | 0.01 | 0.0185 | 0.0256 | 1698 |
| | 15 | [20,20] | Tanh | 0.01 | 0.0742 | 0.1934 | 63 |
| | 25 | [20,20] | Tanh | 0.01 | 0.0059 | 0.0054 | 9246 |
| | 15 | [20,20,20,20] | Tanh | 0.01 | 0.3338 | 0.1367 | 9 |
| | 25 | [20,20,20,20] | Tanh | 0.01 | 0.2648 | 0.1794 | 12 |
| | 15 | [30,30] | Tanh | 0.01 | 0.0014 | 0.0023 | 20.000 |
| | 25 | [30,30] | Tanh | 0.01 | 0.0222 | 0.0268 | 2527 |
| | 15 | [30,30,30] | Tanh | 0.01 | 0.8258 | 0.6324 | 5 |
| | 25 | [30,30,30] | Tanh | 0.01 | 0.2994 | 0.3314 | 77 |
| | 15 | [30,30,30,30] | Tanh | 0.01 | 0.4327 | 0.2816 | 7 |
| | 25 | [30,30,30,30] | Tanh | 0.01 | 0.4953 | 0.4316 | 14 |
| **Activation function** | 15 | [5,5,5,5] | Sigmoid | 0.01 | 0.0148 | 0.0834 | 9149 |
| | 15 | [5,5,5,5] | ReLu | 0.01 | 0.0297 | 0.1285 | 5 |
| | 15 | [10,10] | Sigmoid | 0.01 | 0.0087 | 0.0440 | 301 |
| | 15 | [10,10] | ReLu | 0.01 | 0.0288 | 0.1292 | 5 |
| | 15 | [30,30] | Sigmoid | 0.01 | 0.2939 | 0.4771 | 6 |
| | 15 | [30,30] | ReLu | 0.01 | 0.0297 | 0.1285 | 5 |
| **Learning rate** | 15 | [5,5,5,5] | Tanh | 0.001 | 0.0091 | 0.0467 | 265 |
| | 15 | [10,10] | Tanh | 0.001 | 0.0444 | 0.0829 | 311 |
| | 15 | [30,30] | Tanh | 0.001 | 0.0879 | 0.1061 | 4860 |
| | 15 | [5,5,5,5] | Tanh | 0.005 | 0.0020 | 0.0036 | 20.000 |
| | 15 | [10,10] | Tanh | 0.005 | 0.0042 | 0.0086 | 20.000 |
| | 15 | [30,30] | Tanh | 0.005 | 0.10025 | 0.1710 | 306 |
| | 15 | [5,5,5,5] | Tanh | 0.02 | 0.0147 | 0.0593 | 23 |
| | 15 | [10,10] | Tanh | 0.02 | 0.1921 | 0.5809 | 14 |
| | 15 | [30,30] | Tanh | 0.02 | 0.0293 | 0.0421 | 820 |
| | 15 | [5,5,5,5] | Tanh | 0.05 | 0.0016 | 0.0026 | 7631 |
| NN2 | 15 | [10,10] | Tanh | 0.05 | 0.0012 | 0.0020 | 8065 |
| | 15 | [30,30] | Tanh | 0.05 | 0.0088 | 0.0130 | 256 |

## 5.2 Results on test data set

After finding the most optimal hyperparameters using the validation set, predictions are performed using the test data set. For the predictions, 100.000 epochs are used. To have an efficient trading strategy, the buy or sell strategy of the cryptocurrencies is based on the average price of the cryptocurrencies. The cryptocurrency will be sold (bought) if at least two of the below situations is giving a selling (buying) opportunity.

- If the predicted price is higher (lower) than the average over the last six months, the cryptocurrency will be potentially sold (bought).[1]
- If the predicted price is higher (lower) than the average over the last three months, the cryptocurrency will be potentially sold (bought).[2]
- If the predicted price is higher (lower) than the average over the last year, the cryptocurrency will be potentially sold (bought).[3]

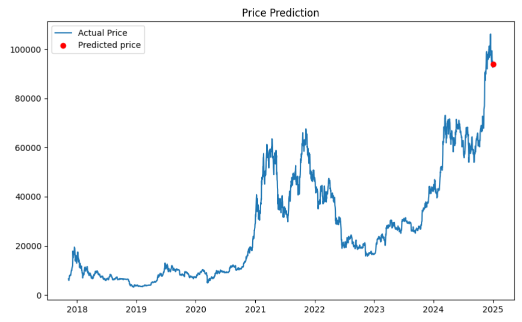
Table 4 shows the predicted prices for the NNs selected with the lowest validation losses. The Table shows that the test loss is low for most instances, except from NN2 and NN3 for Ethereum. In most cases, the predicted prices are higher than the average over last year, last half year and the last three months, meaning that this almost always results in a buying strategy. It results in a selling strategy for NN2 for Ethereum, but since the test loss is also very high, this result should be taken with caution and is not accurate.

Table 4: Predicted cryptocurrency prices on test data set.

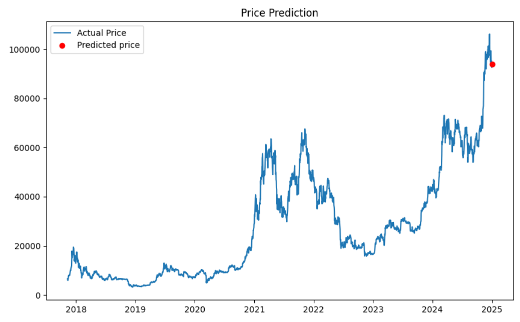
| | | Test prediction | Test true | Test loss | Trading strategy |
|---|---|---|---|---|---|
| NN1 | BTC | \$93.846 | \$93.429 | 0.0017 | Buy (3 of 3 True) |
| NN2 | BTC | \$93.966 | \$93.429 | 0.0029 | Buy (3 of 3 True) |
| NN1 | ETH | \$3597 | \$3333 | 0.0047 | Buy (3 of 3 True) |
| NN2 | ETH | \$1540 | \$3333 | 0.2402 | Sell (3 of 3 False) |
| NN3 | ETH | \$3461 | \$3333 | 0.0115 | Buy (3 of 3 True) |
| NN1 | XRP | \$2.16 | \$2.08 | 0.0026 | Buy (3 of 3 True) |
| NN2 | XRP | \$1.96 | \$2.08 | 0.0019 | Buy (3 of 3 True) |

Figures 2, 3 and 4 show the actual cryptocurrency prices and the predicted price.

Figure 2: Prediction Bitcoin price in dollars.



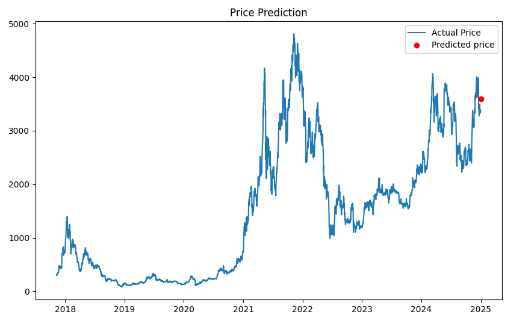
(a) BTC NN1



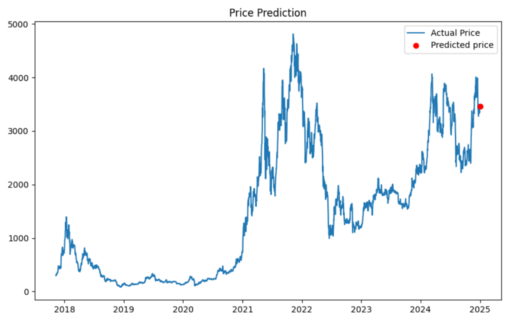
(b) BTC NN2

[1]The half year average for BTC, ETH and XRP are equal to \$71.343, \$3032 and \$0.86, respectively.
[2]The half year average for BTC, ETH and XRP are equal to \$83.426, \$3092 and \$1.29, respectively.
[3]The half year average for BTC, ETH and XRP are equal to \$65.964, \$3045 and \$0.74, respectively.

Figure 3: Prediction Ethereum price in dollars.

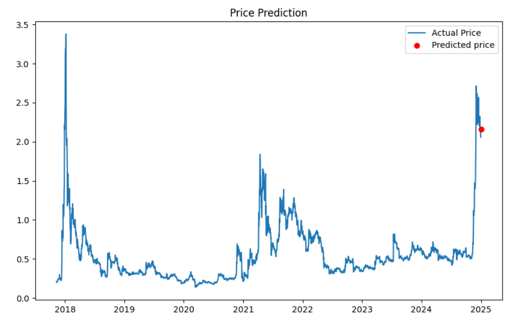


(a) ETH NN1

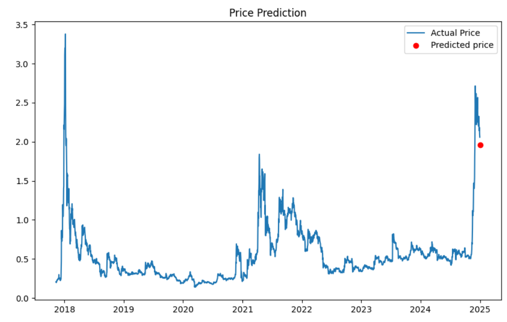


(b) ETH NN3

Figure 4: Prediction XRP price in dollars.



(a) XRP NN1



(b) XRP NN2

## 6 Conclusion & discussion

The predicted Bitcoin price for the NN with the lowest test loss (0.0017) is equal to 93.846 dollars, the predicted Ethereum price for the NN with the lowest test loss (0.0047) is equal to 3597 dollars and the predicted XRP price for the NN with the lowest test loss (0.0019) is equal to 1.96 dollars. This results in a buying strategy for all cryptocurrencies.

A relatively simple structure of the NN is used in this report, which is possible to extend further. Because using a black-box model is already quite difficult to interpret, simple data inputs are used to predict the prices of cryptocurrencies. The input data can be one of the improvements to finetune the training of the NN and prediction of the cryptocurrencies.

Also, in this report the number of hidden layers is restricted to four hidden layers with a maximum of 30 neurons. Deeper networks can lead to overfitting but also can possibly explain the data structures better. This can be taken into consideration for research in the future. Moreover, the most optimal hyperparameter settings are obtained via trail-and-error. A more structured approach by using a grid search can be an improvement of the study as well. Regarding the structure, now only one loss function, namely MSE, is used. Multiple other loss functions such as root mean-squared error and mean absolute error can be investigated further.

Finally, another improvement for this algorithm is using stochastic gradient descent instead of (standard) gradient descent (Nielsen, 2015). For standard gradient descent, the algorithm calculates the derivative of the loss function with respect to the parameters using the entire dataset and then update the parameters. This can be computationally expensive, so to make the algorithm quicker stochastic gradient descent, which updates the parameters of the model only using a random subset of the entire dataset, can be an improvement. However, this also requires in general more iterations.

## References

[1] Aroussi, R. (2020). *yfinance 0.2.51.*
Retrieved December 2024, from: $https://pypi.org/project/yfinance/files$

[2] Foote K. D. (2021). *A Brief History of Neural Networks.* Retrieved February 2022, from: $https://www.dataversity.net/a-brief-history-of-neural-networks/$

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2017). *Deep learning (adaptive computation and machine learning series).* Cambridge Massachusetts, 321-359.

[4] Hastie, T., Tibshirani, R. & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.

[5] Leung, K. (2021). *The Dying ReLU Problem, Clearly Explained.*
Retrieved January 2025, from: $https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24$

[6] Medium (2021). *How to do Time Series Split using Sklearn.*
Retrieved January 2025, from: $https://medium.com/@Stan_DS/timeseries-split-with-sklearn-tips-8162c83612b9$

[7] Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). San Francisco, CA, USA: Determination press.

[8] Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). *Activation functions: Comparison of trends in practice and research for deep learning.* arXiv preprint arXiv:1811.03378.

[9] Penm, J., Chaar, B., Moles, R., & Penm, J. (2013). *Predicting ASX Health Care Stock Index Movements After the Recent Financial Crisis Using Patterned Neural Networks.* In Rethinking Valuation and Pricing Models (pp. 599-610). Academic Press.

[10] Taddy, M. (2019). *The technological elements of artificial intelligence (p. 61).* Chicago, IL: University of Chicago Press.

[11] Yahoo Finance (n.d.). *Bitcoin USD (BTC-USD).*
Retrieved January 2025, from: $https://finance.yahoo.com/quote/BTC-USD/?guccounter=1guce_referrer=aHR0cHM6Ly93d3cuYmluZy5jb20vguce_referrer_sig=AQAAAF4_2X1Of4s_OyNVE49ECq6ZRfgtzZYlpTnd4bAE40lKnBhTBGZuVZiiucQcKfrLNBwun-LFlOA7Ulncg-xCZLcw-hXj-uxP7_oEqw3EV4X8xFMlS_p5ajRMeZoCfNnsUOa08876gWKYfza7gwrtjEglB$