e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

BITCOIN FORECASTER WITH NEURAL NETWORKS

Neev Mehta*1

*1Witty International School, Mumbai, India.

DOI : https://www.doi.org/10.56726/IRJMETS58707

ABSTRACT

An artificial neural network is a model from the field of machine learning that was inspired by the structure and function of the brain. Whilst deep learning involves neural networks with multiple layers between the input and output layers. This paper shows how neural networks work, and describes various features. Backpropagation is explained in detail, and the common issues that one can face whilst training a neural network are described. Methods of mitigating these issues are also given. Finally, a neural network is employed to build a daily Bitcoin trading system. The results were inferior to multiple linear regression. The neural network likely suffered from overfitting, but regularisation using dropout helped to mitigate this.

Keywords: Neural networks, deep learning, machine learning, backpropagation

I. INTRODUCTION

Warren McCulloch and Walter Pitts developed the first models of neural networks in 1943. In order to describe how neurons in the brain might work, they modelled a simple neural network using electrical circuits.[1] In 1958 Frank Rosenblatt introduced a layered network of perceptrons, consisting of an input layer, a hidden layer with randomised weights that did not learn, and an output layer with learning connections.[2] The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Valentin Lapa in 1967.[3] Whilst modern backpropagation was first published by Seppo Linnainmaa in 1970 as 'reverse mode of automatic differentiation' for discrete connected networks of nested differentiable functions.[4]

Other authors have employed neural networks to forecast the price of Bitcoin with mixed results. Using data from August 2013 to July 2016, McNally, Roche and Caton sought to forecast the daily closing price of BTC/USD using an autoregressive integrated moving average (ARIMA) model, and two deep learning methods: a Bayesian optimised recurrent neural network (RNN) and a long short-term memory (LSTM) neural network.[5] The inputs used were the daily open, high, low and close price data from CoinDesk, plus the difficulty and hash rate taken from the blockchain. The LSTM achieved the highest accuracy whilst the RNN achieved the lowest root-mean-square error (RMSE). The ARIMA model performed poorly in terms of accuracy and RMSE. Using daily Bitcoin price data from January 2018 to July 2018, Wu et al. proposed a LSTM with AR(2) model, which outperformed a conventional LSTM model.[6] Mudassir et al. used an artificial neural network (ANN), a stacked artificial neural network (SANN), support vector machines (SVM) and LSTM for classification and regression, using data from April 2013 to December 2019, to forecast the Bitcoin price.[7] The LSTM showed the best overall performance. Uras et al. forecast the Bitcoin closing price using data from November 2015 to December 2020 using linear regression, multiple linear regression, univariate LSTM and multivariate LSTM.[8] Multiple linear regression performed best, linear regression second best, univariate LSTM third best and the multivariate LSTM the worst. Ibrahim, Kashef and Corrigan predicted the market movement direction of Bitcoin using ARIMA, Prophet (by Facebook), random forest, random forest lagged-auto-regression and multi-layer perceptron (MLP) neural networks. The MLP achieved the highest accuracy of 54%.[9] Liu et al used a backpropagation neural network (BPNN), principal component analysis-based support vector regression (PCA-SVR), support vector regression (SVR) and stacked denoising autoencoders (SDAE) with data from July 2013 to December 2019 to forecast the price of Bitcoin.[10] The SDAE performed best, the SVR second best, the PCA-SVR third best and the BPNN performed the worst.

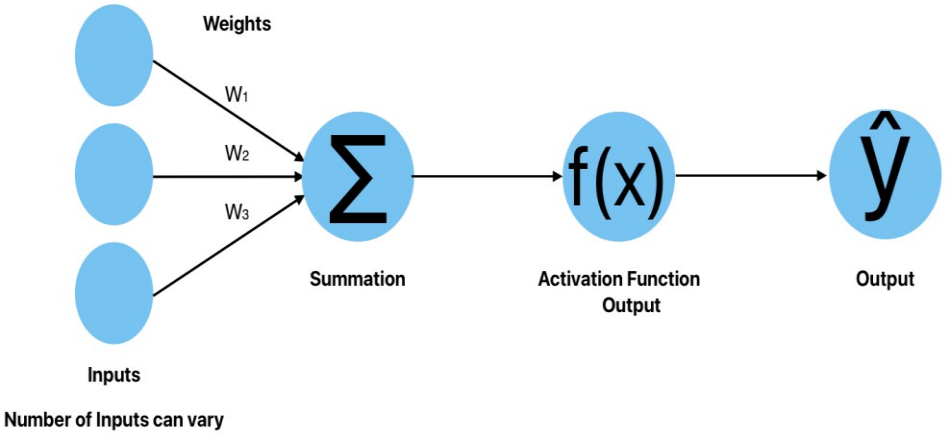
The following section explains how neural networks work, identifies issues and explains how to mitigate them. Then the rest of the paper details a Bitcoin trading system using an ANN, and compares the results to multiple linear regression.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

II. METHODOLOGY

Introduction to Neural Networks - Neural networks are used for building machine learning software. A specific subset of machine learning, called deep learning, uses multi-layered neural networks to mimic the complex decision-making power of a human brain. Neural networks consist of many artificial neurons, which function in a manner similar to the real neurons of our brains. Humans learn by experience, which can be said to be a 'complex' form of deep learning. Neural networks do the exact same thing.

Artificial Neuron



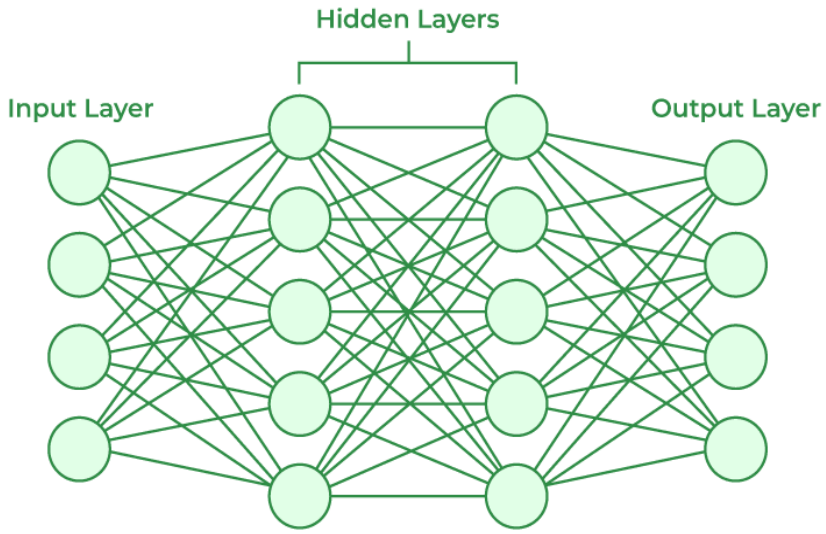
A neuron, also called a perceptron, consists of inputs, weights, biases and activation functions, which all come together to generate an output. Let us look at a simple illustration of a single perceptron.

Each of the inputs have certain weights connecting them to the next node. The summation of the product of each weight with its input is calculated, to which a bias is added, and then it is used as an input to an activation function. The activation function can be any nonlinear function. For example, the sigmoid function, a rectified linear unit (ReLU), etc. The output produced is the output of a particular perceptron.

$\hat{y} = g(\sum_{n=1}^{m} x_n w_n + b)$

Weights are set to random values at first, and then during the training of the neural network, the values of the weights are changed appropriately by the optimiser so that it can accurately predict data of the same type.

How a Neural Network Works- When data is passed to a neural network, it is passed through forward propagation. This means that data is passed from left to right. However, instead of a single perceptron shown in the diagram above, an actual neural network is made of multiple layers, each layer consisting of multiple perceptrons. The output of the activation function in a node is the input for the next layer, which continues until the final output layer. A diagram of a neural network is given below.



e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

A neural network is trained using a training data set. This data set consists of data to be input in the network and the expected output from the neural network, called labels. For each output node in an output layer, a certain value is output. This value can be seen as a probability that the network thinks which output is correct. Hence, the output node with the greatest probability is the output that the neural network thinks will work. Whilst training, the neural network tries to increase the probability of a correct output and decrease the probability of an incorrect output.

Each neural network also has a loss function, which is the value of how wrong the neural network is with its output. Hence, the neural network tries to decrease the loss each time it is run.

Learning Rate- A neural network also has a 'learning rate' set up. It is the rate at which the model changes its values depending on the error. If a smaller value is adopted as the learning rate, the neural network works more accurately, but slowly. If a larger value is adopted as the learning rate, the neural network works more quickly, but less accurately. It is like taking steps in one direction, if the network takes smaller steps (a lower learning rate), it can accurately find the spot (optimum values of the weights). However, if larger steps are taken (a higher learning rate), it can miss the optimum values of weights, by stepping over the spot (optimum value of weights) to another nearby spot (values of weights here cause less accurate outputs).

Backward propagation- A neural network uses optimisers to change the value of weights so that it can run and predict the output accurately. Stochastic gradient descent (SGD) is an example of an optimiser. It uses the derivative of the loss relative to the derivative of the weight, to can then change the value of the weights accordingly.

An input from a node is actually the output calculated from the previous layers. To change the output, the optimiser can do two things, change the weights or change the inputs (of course, the inputs of the input layer cannot be changed, here input denotes the input from one hidden layer to another or hidden layer to output layer). To change the input, the weight of the previous layer has to be changed. To change the input of the previous layer, the weights of the layer before the previous layer need to be changed.

Hence, with each epoch (a single pass of all the training data in forward propagation), the optimiser will change the weights of the previous layers accordingly such that the neural network becomes more accurate. This is called backward propagation (since the optimising goes from right to left).

More formally, backpropagation works as follows.

1. Forward Pass

First, we perform a forward pass to compute the outputs of the network. For a neural network with L layers, where the input to the l-th layer is $\mathbf{a}^{[l-1]}$ and the output of the l-th layer is $\mathbf{a}^{[l]}$:

- Input to the l-th layer:
- $\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$
- where $\mathbf{W}^{[l]}$ is the weight matrix and $\mathbf{b}^{[l]}$ is the bias vector of the l-th layer.
- Output of the l-th layer (after activation):
- $\mathbf{a}^{[l]} = g(\mathbf{z}^{[l]})$

where $g(\cdot)$ is the activation function.

2. Compute Loss

Calculate the loss L between the predicted output $\mathbf{a}^{[L]}$ and the true output $\mathbf{y}$. For example, using the mean squared error (MSE) loss for regression:

$$L = \frac{1}{2}\|\mathbf{a}^{[L]} - \mathbf{y}\|^2$$

or using the cross-entropy loss for classification:

$$L = -\sum_{i}^{n} y_i \log (a_i^{[L]})$$

3. Backward Pass

In the backward pass, we compute the gradients of the loss with respect to each weight and bias in the network to update them.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

- Error at the output layer:

$\delta^{[L]} = \nabla_a L \odot g'(\mathbf{z}^{[L]})$

where $\odot$ denotes element-wise multiplication and $g'(\cdot)$ is the derivative of the activation function. For MSE, $\nabla_a L = \mathbf{a}^{[L]} - \mathbf{y}$.

- Error at layer l:

$\delta^{[l]} = (\mathbf{W}^{[l+1]})^T \delta^{[l+1]} \odot g'(\mathbf{z}^{[l]})$

This propagates the error backward through the network.

- Gradient of the loss with respect to weights and biases:

$\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]} (\mathbf{a}^{[l-1]})^T$

$\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}$

4. Update Weights and Biases

Using gradient descent, update the weights and biases. For a learning rate η:

Update weights:

$\mathbf{W}^{[l]} \leftarrow \mathbf{W}^{[l]} - \eta \frac{\partial L}{\partial W^{[l]}}$

- Update biases:

$\mathbf{b}^{[l]} \leftarrow \mathbf{b}^{[l]} - \eta \frac{\partial L}{\partial b^{[l]}}$

Overfitting and Underfitting- Overfitting is a process where the neural network becomes very good at solving the training data set, but cannot accurately solve out-of-sample (previously-unseen) data. Overfitting has always been a huge obstacle in deep learning. There are multiple methods that can reduce overfitting in a neural network:

1) Use more data to train the neural network. This provides more varying input to the neural network, which helps the network to be more accurate. Data augmentation can be used to increase the amount of data.
2) Reduce the complexity of the model. The number of layers or number of neurons per layer can be decreased. This may help the model to generalise better and work with unseen data.
3) Dropouts can be used by the network as well. It will automatically ignore some of the nodes in the layer during an epoch so that the network generalises better and works with unseen data.

   Underfitting is a process where the neural network is not able to solve the training data with great accuracy itself, let alone being able to solve unseen data. There are multiple methods that can reduce underfitting in a neural network:

1) Use more data to train the neural network. This provides more varying input to the neural network which helps the network to be more accurate. Data augmentation can be used to increase the amount of data.
2) Increase the complexity of the model. The number of layers or number of nodes can be increased. If data is more complex and the model is relatively simple, the network may not be able to predict accurately.
3) Add more features to the data in the input set. This might help the network classify the information better.

Vanishing Gradient Problem- The vanishing gradient problem is a common problem that arises during the training of a neural network. This happens when the value of the weight becomes so small that is it almost negligible, such that for the update of the weights during each epoch, the weight changes by a negligible value.

Due to backpropagation, each value of weight in the earlier layers is dependent on the value of weights in the later layers. Hence, if the value of weights and output is small in the later layers, it causes the weights in the earlier layers to be changed by a negligible value, and hence there is no substantial change in the neural network for it to get properly trained.

Exploding Gradient Problem- The exploding gradient problem is the same as the vanishing gradient problem, but instead of the weights being too small, the weights are too large.

Mitigation of the Unstable Gradient Problem- Since there are multiple weights used to calculate the value of a single node, the variation of output from a node is quite high. Hence, the standard deviation of the possible

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

outputs of the node is also, in turn, quite high. A higher standard deviation causes the desired output from the activation function to be further away from where the variations of nodes are saturated. With each epoch, there is only a small change in the value towards the desired output. Hence, it causes issues during the training.

To mitigate this issue, the variation of output from the node needs to be reduced. If n is the number of weights, the optimum value for this variation is 1/n. To change weights that are randomly generated to fall into this variation, each weight needs to be multiplied by the square root of 1/n. The optimum value used depends on the activation function it refers to. Values here are for the sigmoid function. This method of initialisation is called Xavier initialisation.

Bias in a Neural Network- Biases are values in every node that are added to the summation of the product of weights and inputs. Biases are learnable values, which means that they are also updated like weights. A node in a neural network, depending on the value generated by the activation function, decides whether the value should be forwarded on or not. For example, in the ReLu activation function, if the output from the function is 0, the value is not forwarded on. We can say that there is a certain threshold of inputs which cause the value to be output as 0, which means the value will not be forwarded. Now, to change the thresholds, biases come into play.

$\hat{y} = g(\sum_{n=1}^{m} x_n w_n + b)$

In the above equation, b denotes bias, which changes the summation entirely and hence can be used to control the threshold of the inputs that need to be changed.

III. MODELING AND ANALYSIS

The goal is to build a trading system that trades daily and is either long or short Bitcoin.

Technical analysis is the art/science of forecasting future market prices by means of analysis of historical data generated by the process of trading. If the weak form of the efficient market hypothesis (EMH) holds, then technical analysis has no value.[11] Conversely, for technical analysis to work requires that the weak (and therefore the semi-strong and strong) forms of the EMH are false. This model exclusively uses technical analysis (price and volume), so assumes that all forms of the EMH are false, and is thus a difficult challenge. Bitcoin data from September 2014 to May 2024 was downloaded from Yahoo! Finance. For model inputs, returns are used because the distribution is closer to stationary. Whilst logarithms are used because the distribution is closer to normal. Note that all inputs are orthogonal, which avoids the problems caused by multicollinearity. Let p_t be the closing price of BTC/USD on day t and $high_t$, low_t and $volume_t$ the daily high, low and volume on day t. The six inputs were as follows.

$x_1 = \log(p_0/p_{-1})$

$x_2 = \log(p_{-1}/p_{-3})$

$x_3 = \log(p_{-3}/p_{-7})$

$x_4 = \log(p_{-7}/p_{-15})$

$x_5 = \log(high_0/low_0)$

$x_6 = \log(volume_0)$

The model target was one-day-ahead log returns.

$y = \log(p_1/p_0)$

The data was split into 50% training set, 25% validation set and 25% test set, keeping the time series in order. Then the independent variables were scaled to fit the training set. The system is trained to maximise profit. No transaction costs are included. Feedforward multilayer perceptron networks trained using the backpropagation method were implemented using the Sequential model in Python. The neural network was run with 2, 3, 4 and 5 layers, and also with no regularisation, dropout and early stopping. The validation set was used to optimise the number of units in each layer, the activation function, the dropout rate and the patience (the number of epochs with no improvement after which training will be stopped). For comparison, multiple linear regression was trained on the training+validation set, and tested on the test set.

IV. RESULTS

The results of the neural network trading systems are given in Table 1. For comparison, the multiple linear regression model produced average daily log returns of 0.00106.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:06/June-2024 Impact Factor- 7.868 www.irjmets.com

Table 1. Average daily log returns generated by the ANNs on the test set

		Number of layers				
		2	3	4	5	Average
Regularisation	None	-0.00212	0.00032	-0.00060	-0.00041	-0.00070
	Dropout	0.00055	0.00106	-0.00184	0.00056	0.00008
	Early stopping	-0.00079	-0.00037	0.00028	-0.00074	-0.00040
	Average	-0.00079	0.00034	-0.00072	-0.00020	-0.00034

V. CONCLUSION

Neural networks are affected by many issues like overfitting, underfitting and unstable gradients. The key to an efficient neural network is to train and retrain the network with lots of data, whilst trying different parameters (different activation functions and trying out different complexities of network) such that the neural network can deliver its best potential. With the Bitcoin trading system, on average, the neural networks with three layers performed best. Whilst regularisation helped, particularly dropout. However, none of the neural networks outperformed linear regression. This could be because there were no significant nonlinearities present (which would be surprising in a financial market), or, more likely, the neural networks were overfitting the training data because it was relatively sparse (daily) and the dynamics of an immature market changed through time.

ACKNOWLEDGEMENTS

I wish to thank Dr Martin Sewell, BSc (Hons), MSc, PhD for his assistance with this paper.

VI. REFERENCES

[1] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” Bulletin of Mathematical Biophysics, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[2] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.” Psychological Review, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[3] G. Ivakhnenko and V. G. Lapa, Cybernetics and Forecasting Techniques. New York: American Elsevier Publishing Company, Jan. 1967.

[4] S. Linnainmaa, “Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden Taylor-kehitelmänä,” Master’s thesis, University of Helsinki, Helsinki, 1970, the representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.

[5] S. McNally, J. Roche, and S. Caton, “Predicting the price of Bitcoin using machine learning,” in 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2018), I. Merelli, P. Liò, and I. Kotenko, Eds. Los Alamitos: IEEE Computer Society, Mar. 2018, pp. 339–343.

[6] C.-H. Wu, C.-C. Lu, Y.-F. Ma, and R.-S. Lu, “A new forecasting framework for bitcoin price with LSTM,” in 18th IEEE International Conference on Data Mining Workshops, H. Tong, Z. J. Li, F. Zhu, and J. Yu, Eds. Los Alamitos: IEEE, Nov. 2018, pp. 168–175.

[7] M. Mudassir, S. Bennbaia, D. Unal, and M. Hammoudeh, “Time-series forecasting of Bitcoin prices using high-dimensional features: A machine learning approach,” Neural Computing and Applications, Jul. 2020.

[8] N. Uras , L. Marchesi , M. Marchesi, and R. Tonelli, “Forecasting Bitcoin closing price series using linear regression and neural networks models,” PeerJ Computer Science, vol. 6, p. e279, Jul. 2020.

[9] Ibrahim, R. Kashef, and L. Corrigan, “Predicting market movement direction for bitcoin: A comparison of time series modeling methods,” Computers & Electrical Engineering, vol. 89, p. 106905, Jan. 2021.

[10] M. Liu, G. Li, J. Li, X. Zhu, and Y. Yao, “Forecasting the price of Bitcoin using deep learning,” Finance Research Letters, vol. 40, p. 101755, May 2021.

[11] E. F. Fama, “Efficient capital markets: A review of theory and empirical work,” The Journal of Finance, vol. 25, no. 2, pp. 383–417, May 1970.