University College London

The use and implementation of Artificial Neural Networks in predicting directional movements of short-term cryptocurrency transactions

Cezary Klimczuk

Department of Economics

Abstract

This paper applies various Artificial Neural Network architectures to two-dimensional, BTC/USD and ETH/USD exchange rate data in an effort to predict directional market movements on a 3 minute prediction horizon. The study shows that combining Long-Short Term Memory and Dense layers allows the network to learn complex, non-linear dependencies and give accurate predictions when presented with not yet encountered data. A trading simulation showed no significant relationship between networks' accuracy and the size of a price change. Higher prediction accuracy correlated with lower prediction entropy suggest that developing a profitable trading strategy is possible, if further changes to the methodology are implemented.

Contents

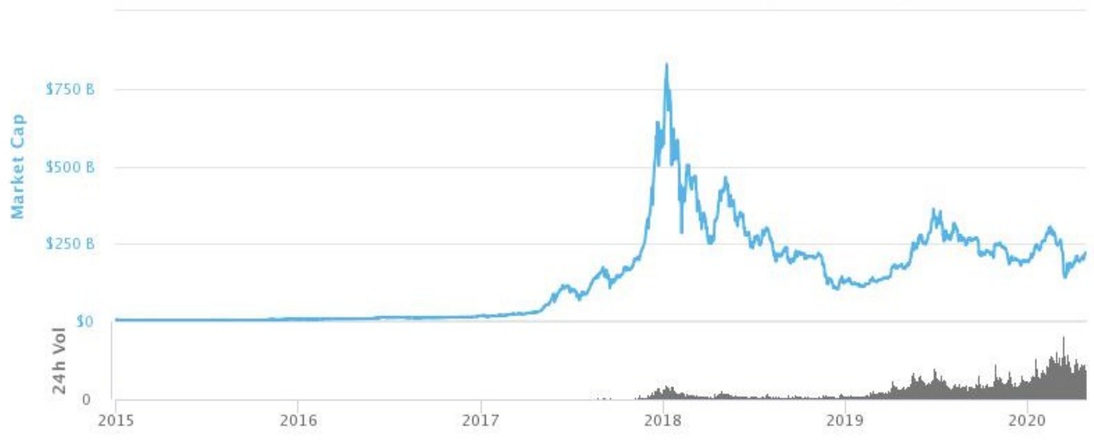
1 Introduction

1.1 Background

1.1.1 The Cryptocurrency Market

Since the invention of the biggest cryptocurrency - Bitcoin (BTC) - in 2008, there has been a rising interest in the cryptocurrency market, as well as blockchain technology. Cryptocurrencies are decentralized digital assets, serving as a medium of exchange, securing financial transactions and verifying the transfer of assets between individuals. Their decentralized nature comes from the distributed ledger technology based on strong cryptography and a proof-of-work technique (Nakamoto 2009). By the end of April 2020, the cryptocurrency market capitalization amounts to $220bn (CoinMakretCap 2020). This figure is higher than the nominal GDP of countries like New Zealand, Qatar or Ukraine; and is predicted to raise in the near time horizon (Research and Markets 2019). Back in 2017, when the cryptocurrency bubble occurred, the market capitalization increased to a point, that it overtook prominent stock market exchanges by the likes of Singapore Exchange or Moscow Exchange.

Figure 1.1: Cryptocurrency market capitalization



For many, this technology is seen as a solution to the intermediary problem, others perceive cryptocurrencies as extremely volatile assets capable of bringing over-the-market-rate margins. (Conrad et al. 2018) have shown that cryptocurrency markets - the market for bitcoin in particular - demonstrate very high levels of volatility in comparison to commodities, stocks or traditional currency markets. It is also very weakly correlated with other markets, hence making it more difficult to predict future movements. Investing in such a market is undoubtedly associated with big risks, but at the same time can bring big returns. In order to exploit the price volatility, a significant amount of research has been done exploring various models. Constructing an accurate model able to predict market movements would allow financial institutions to make more accurate decisions and increase margins.

1.1.2 A Traditional Time-Series Approach

Such modelling was considered impossible over the 1960s and 1970s, when the world of finance was dominated by the Efficient Market Hypothesis (EMH) theory. It assumes that markets instantaneously reflect all available information in the price of assets, hence leaving no room for any successful future prediction. A direct implication of this, is that asset prices follow a random walk, making it impossible to reasonably model them. This hypothesis has its roots in the work of (Samuelson 1965), which was then further extended by (Fama 1970), who outlined three forms of the hypothesis - weak, semi-strong and strong. An *efficient market* in his work is defined as "a market with great number of rational, profit-maximizers actively competing, with each trying to predict future market values of individual securities, and where current important information is almost freely available to all participants". Since then, numerous researchers have criticized the underlying assumptions and pointed towards the behavioural nature of the market participants (Thaler and Bondt 1985), (Lakonishok, Shleifer and Vishny 1994) invalidating the EMH. Nonetheless, the hypothesis remained popular among many economists and was defended as recently as the early 21st century (Malkiel 2003).

Despite the ongoing debate around the validity of EMH, numerous efforts have been made to explore possibilities of modelling asset prices. The most natural choice for time-series data, autoregression models firstly introduced in 1970s by (Box and Jenkins 1970) have been investigated by (Adebiyi, Adewumi and Ayo 2014) and particularly in the context of cryptocurrency market by (Abu Bakar and Rosbi 2017). Both papers revealed limitations associated with the autoregression model highlighting high volatility of the cryptocurrency market and the linear nature of the underlying modelling technique. Although, the ARIMA model can be used to a certain extent to predict future behaviour, it is argued that asset prices can possess somewhat more complex properties and exhibit non-linear nature. Therefore, other predictive modelling techniques were suggested such as the Artificial Neural Networks.

1.2 Problem Formulation

This research is conducted using minute-by-minute exchange rate transaction data, namely BTC/USD and ETH/USD from 2018 and 2019 downloaded from the Gemini exchange (Gemini 2020). The data consists of open, high, low and close prices for every minute in a given year. To complement the dataset, volumes of all transactions that happened every minute are also included in the dataset.

The objective of this research is to create a reliable model that can efficiently predict the exchange rate movement of future Ethereum (ETH) transactions at short-time interval of 3 minutes. The model's task is to return the following predictions: 1 – if the model predicts the value of ETH to increase, 0 – if the model predicts that the value will not increase (i.e. fall or stay the same). The method used specifically in this research is a variation of the Artificial Neural Network techniques (ANN).

In mathematical terms, the objective function can be formulated as the following:

$$\hat{y}_{t+\Delta t} = f(\mathbf{x}_t) \tag{1.1}$$

where $\hat{y}_{t+\Delta t}$ is the prediction of the direction of the Ethereum price movement expressed in US dollars in the next t minutes, based solely on the information embodied in the date at time t.

$$\hat{y}_{t+\Delta t} = \begin{cases} 1 & \text{if } \hat{p}^{ETH}_{t+\Delta t} > p^{ETH}_t \\ 0 & \text{if } \hat{p}^{ETH}_{t+\Delta t} \leq p^{ETH}_t \end{cases} \tag{1.2}$$

The information $\mathbf{x}$ is a set of relevant predictive features such as prices, price dynamics and volumes. Finally, the classifier is evaluated using different accuracy measures as well as putting it in a trading simulation.

1.3 Dissertation Outline

This dissertation is structured in the following way. In chapter 2 the basic setup and the theory behind the concept of Artificial Neural Networks is covered, explaining their structure, learning process and specific architectures that seem particularly promising in our task. Chapter 3 explains how ANNs are implemented in the context of cryptocurrency prediction and how the evaluation process is set up. In chapter 4 the results are shown covering different types of performance assessments. Chapters 5 is left out for discussion around further work and potential implementations of this model. Finally, chapter 6 summarizes conclusions stemming from this paper.

2 Artificial Neural Networks

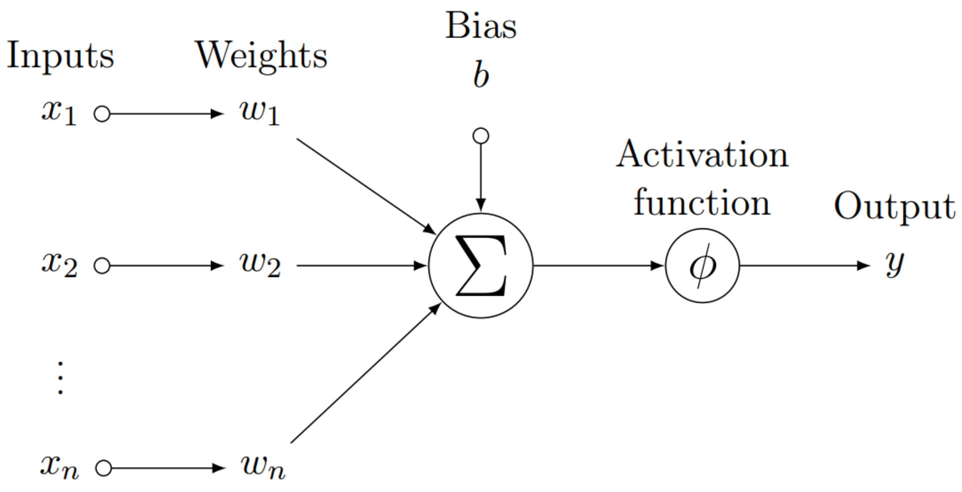
2.1 Perceptron

A perceptron is a building block of every Artificial Neural Network. ANN in itself is a collection of neurons put together into multiple layers that transmit information via links. The information stored in a neuron is received via inputs x_i, weights w_i and bias b. The input vector is multiplied by the weight vector and the bias is added. That signal is then transmitted through the activation function $\phi(.)$ to generate an output y, that is later passed onto the next layer.

$$y = \phi(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) \tag{2.1}$$

The illustration of this process is shown in Figure 2.1:

Figure 2.1: Perceptron architecture



The main reason for the introduction of activation functions is to make sure that values transmitted between neurons stay within certain bounds. Also, this measure introduces non-linearity into the model. This resembles the behaviour of human neurons, which transmit electrical impulses in a non-linear fashion.

A popular choice for the activation functions were the hyperbolic tangent:

$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

and the logistic function:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Both of these functions ensure that the final signal stored within a neuron stays within finite bounds. Recently however, another activation function started to emerge as an industry go-to (Xu et al. 2015), namely Rectified Linear Unit Function (ReLU).

$$\phi(x) = \max(x, 0) \tag{2.4}$$

Its structure is inspired biologically, where the human transmits signals, which are proportional to the amount of stimulation a neuron receives from previous layers.

Finally, in the case where a probability distribution over a discrete variable with n possible values is required, a softmax activation function is widely used. Given the signal z_i received by a neuron, and other signals z_j received by neurons in the same layer, the output is given by:

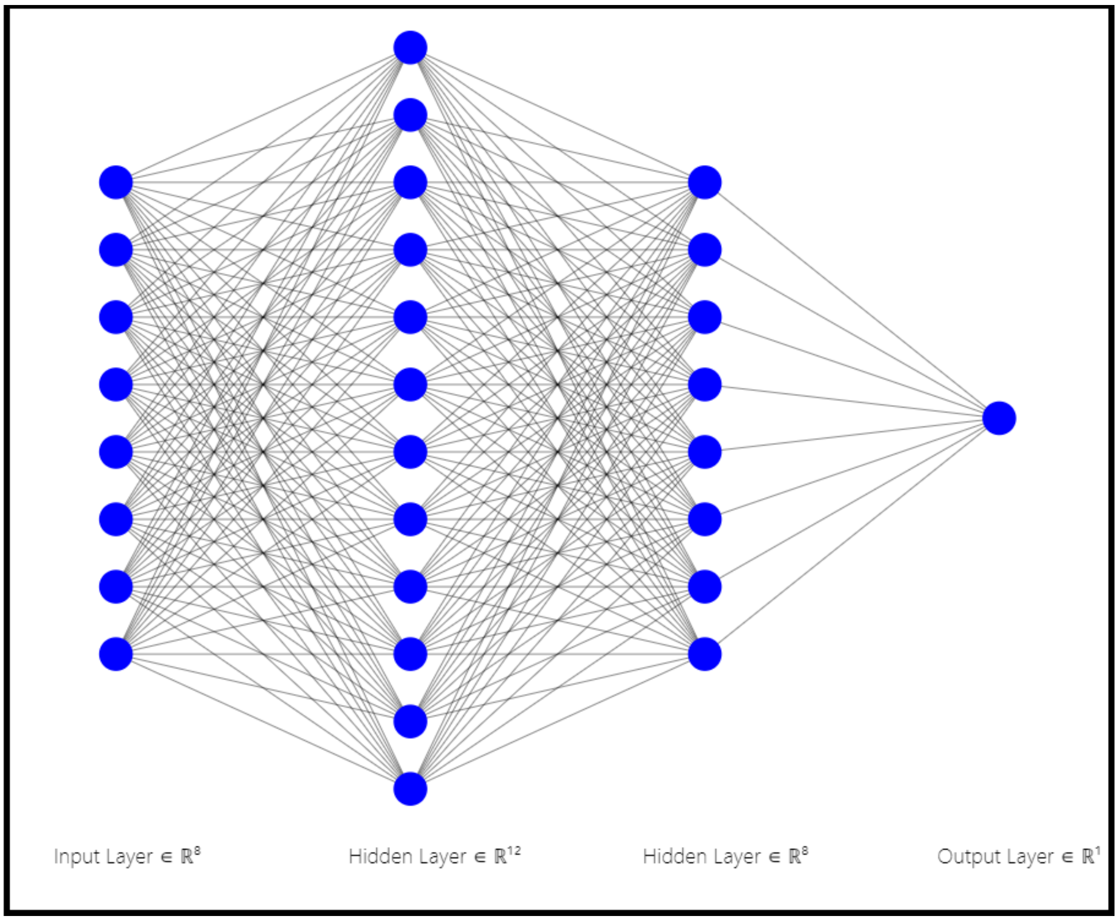
$$\phi(\mathbf{z}) = \frac{\exp z_i}{\sum_j \exp z_j} \tag{2.5}$$

2.2 Basic Dense Architecture

As mentioned previously, the ANN is based on a collection of interconnected layers of nodes. In the most basic dense neural network, a signal from each node within a layer is passed via specially dedicated links to every node in the following layer, as shown in Figure 2.2. It is a feedforward network, where the signals are only flowing towards the output layer. In section 2.5 slightly more complex architecture is introduced.

The input to the neural network consist of a set of signals $\mathbf{x} = x_1, x_2, \ldots, x_n$, where $\mathbf{x} \in \mathbf{R}^n$. The input to the first hidden layer $\mathbf{h}^{(1)}$, consisting of m nodes is

Figure 2.2: Dense Neural Network



calculated by:

$$\mathbf{h}^{(1)} = \begin{cases} h_1^{(1)} = \phi^{(1)}(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + \cdots + w_{1n}^{(1)}x_n + b_1^{(1)}) \\ h_2^{(1)} = \phi^{(1)}(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + \cdots + w_{2n}^{(1)}x_n + b_2^{(1)}) \\ \vdots \\ h_m^{(1)} = \phi^{(1)}(w_{m1}^{(1)}x_1 + w_{m2}^{(1)}x_2 + \cdots + w_{mn}^{(1)}x_n + b_m^{(1)}) \end{cases} \tag{2.6}$$

The next hidden layer $\mathbf{h}^{(2)}$ is calculated in a similar fashion with $\mathbf{h}^{(1)}$ being treated as an input.

$$\mathbf{h}^{(2)} = \begin{cases} h_1^{(2)} = \phi^{(2)}(w_{11}^{(2)}h_1^{(1)} + w_{12}^{(2)}h_2^{(1)} + \cdots + w_{1m}^{(2)}h_m^{(1)} + b_1^{(2)}) \\ h_2^{(2)} = \phi^{(2)}(w_{21}^{(2)}h_1^{(1)} + w_{22}^{(2)}h_2^{(1)} + \cdots + w_{2m}^{(2)}h_m^{(1)} + b_2^{(2)}) \\ \vdots \end{cases} \tag{2.7}$$

A similar pattern follows for the remainder of hidden layers until the output layer is reached.

Since this is a set of purely algebraic operations, we can write this mechanism down in linear algebra terms with:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \mathbf{h}^{(i)} = \begin{bmatrix} h_1^{(i)} \\ h_2^{(i)} \\ \vdots \\ h_n^{(i)} \end{bmatrix}, \mathbf{W}^{(i)} = \begin{bmatrix} w_{11}^{(i)} & \dots & w_{1n}^{(i)} \\ \vdots & \ddots & \vdots \\ w_{m1}^{(i)} & \dots & w_{mn}^{(i)} \end{bmatrix} \tag{2.8}$$

This notation allows to generalize the operations performed by the neural network at each step. The j^{th} layer is calculated as follows:

$$\mathbf{h}^{(j)} = \phi^{(j)}(\mathbf{W}^{(j)}\mathbf{h}^{(j-1)} + \mathbf{b}^{(j)}) \tag{2.9}$$

Hence given an initial output of $\mathbf{x}$, a dense neural network consisting of k layers produces an output $\hat{y}$ given by:

$$\hat{y} = \phi^{(k)}(\mathbf{W}^{(k)}\phi^{(k-1)}(\mathbf{W}^{(k-1)}\phi^{(k-2)}(\dots) + \mathbf{b}^{(k-1)}) + \mathbf{b}^{(k)}) \tag{2.10}$$

The process represented by the equation above is called *forward propagation.* In order to arrive at the values of weights $\mathbf{W}^{(i)}$ and biases $\mathbf{b}^{(i)}$, the network needs to be trained by assessing training examples and updating links through the *back-propagation algorithm.*

2.3 Backpropagation Algorithm and Training Process

The goal of the neural network is to return a prediction $\hat{y}$ that is as close as possible to the true value of y given the information $\mathbf{x}$. To quantify how far off the neural network is from the true value, a certain specific cost function needs to be defined. It indicates how well the network is doing, hence the training process aims at minimizing the error value raised by that function.

In the case of regression problems, where the network's objective is to predict a certain numerical value, a quadratic function is widely used to evaluate its performance. It is given by:

$$J_S(\boldsymbol{\theta}, \mathbf{x}, y) = \frac{1}{2n} \sum_{i \in S} (f(\mathbf{x}_i) - y_i)^2 \tag{2.11}$$

Where $\boldsymbol{\theta}$ represents the current set of weights and biases ($\mathbf{W}$, $\mathbf{b}$), S is a set of cases over which the accuracy is evaluated, and n is the number of cases in set S.

When the network is faced with a classification problem, whereby it returns a probability distribution that an information $\mathbf{x_i}$ comes from a class k, the cross-entropy cost function is used:

$$J_S(\boldsymbol{\theta}, \mathbf{x}, y) = -\frac{1}{n} \sum_{i \in S} \sum_{k \in K} y_i^k \log(\hat{y}_i^k) \tag{2.12}$$

Where K is the set of classes that can be predicted by a model, $\hat{\mathbf{y}}_i$ is the predicted probability distribution by $(f(\mathbf{x}_i))$, and $\mathbf{y}_i$ is the true probability distribution. Notably, for multi-class classification problems, the true distribution is given by a vector of zeros and a one. For instance, where $\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ is the true probability distribution, meaning that the information comes from the class $k = 4$, the predicted one could be $\begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.5 & 0.1 \end{bmatrix}$, yielding a cross-entropy loss of $\log(0.5) \approx 0.6931$.

The network improves its prediction by updating weights and biases within itself. In order to do so, after each iteration (each feedforward procedure), the loss is calculated, as well as the derivative of loss with respect to each weight and bias within the network. Mathematically, this can be formulated as:

$$w_{jk}^{(i)*} = w_{jk}^{(i)} - \epsilon \frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial w_{jk}^{(i)}} \tag{2.13}$$

$$b_j^{(i)*} = b_j^{(i)} - \epsilon \frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial b_j^{(i)}} \tag{2.14}$$

Where $w_{jk}^{(i)*}$ and $b_j^{(i)*}$ are the updated weights and biases, and ϵ is the learning rate, by which they change. Learning rate is somewhat central to the performance of a training process, and will be covered in more detail in section 2.4. The partial derivatives for both $\frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial w_{jk}^{(i)}}$ and $\frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial b_j^{(i)}}$ are obtained via the chain rule, iteratively investigating how the value of loss changes with respect to neighbouring nodes. As an example, the value of a weight in the final hidden layer n would be:

$$\frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial w_{jk}^{(n)}} = \frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(n)}} \frac{\partial z^{(n)}}{\partial w_{jk}^{(n)}} \tag{2.15}$$

Where $z^{(n)}$ is the output before applying the final activation function, i.e. $\hat{y} = \phi^{(n)}(z^{(n)})$. This reasoning can be generalized for both weights and biases from any hidden layer. Summing derivatives across all paths $p \in P$ from $w_{jk}^{(i)}$ that lead to J gives a total derivative. This can be formulated as:

$$\frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial w_{jk}^{(i)}} = \sum_{p \in P} \frac{\partial J_S(\boldsymbol{\theta}, \mathbf{x}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(n)}} \underbrace{\cdots\cdots}_{\text{path } p \in P} \frac{\partial h_j^{(i)}}{\partial z_j^{(i)}} \frac{\partial z_j^{(i)}}{\partial w_{jk}^{(n)}} \tag{2.16}$$

This procedure is the core of the backpropagation algorithm, which is a gradient-based method. It drew a significant amount of interest after being published by (Rumelhart, Geoffrey Hinton and Williams 1986). This is only one of the various training algorithms. Other examples would include genetic-based approaches (Such et al. 2017).

The entire training process consists of feeding the neural network with examples $(\mathbf{x}_i, y_i)$ form a *training set*, calculating the loss in each instance and updating weights and biases accordingly. One pass through the training set is called an *epoch*. Once an epoch of training has passed, the accuracy of the neural network is evaluated over a *validation set*. The net can be trained through multiple epochs until the validation set accuracy reaches a plateau (or starts decreasing).

2.4 Regularization methods

It is important to note that an increasing accuracy of the training set (i.e. the network's ability to correctly predict outputs of the training set examples seen in previous epochs) does not necessarily imply an increased accuracy of the validation set. Conversely, if a training process takes place over too many epochs, the neural net will cease looking for complex patterns within the information set and will start "memorizing" training examples, given it has enough capacity. In the context of ANNs this is called *overfitting*. It occurs when the network's structure is overly complex and can accommodate all training examples into its structure. To address this problem, many regularization methods exist in the deep learning literature.

2.4.1 Random Initialization

Random initialization of neural network weights and biases ensures robustness of the initial model. Modern approaches used these days are heuristic and simple. Since the network optimization is not yet well understood by the academic community, there is no consensus on how the neural network weights should be initialized. The most popular choices for initialization are random Gaussian of uniform distributions (Mishkin and Matas 2016). The choice of a prior distribution does not seems to matter as much as the scale of initial weights. Most commonly used heuristic is:

$$W_{i,j} \sim U\Big(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}} \Big) \tag{2.17}$$

Where m is the number of inputs into the fully connected layer. Other suggestions have been made (Glorot and Bengio 2010) using normalized initialization

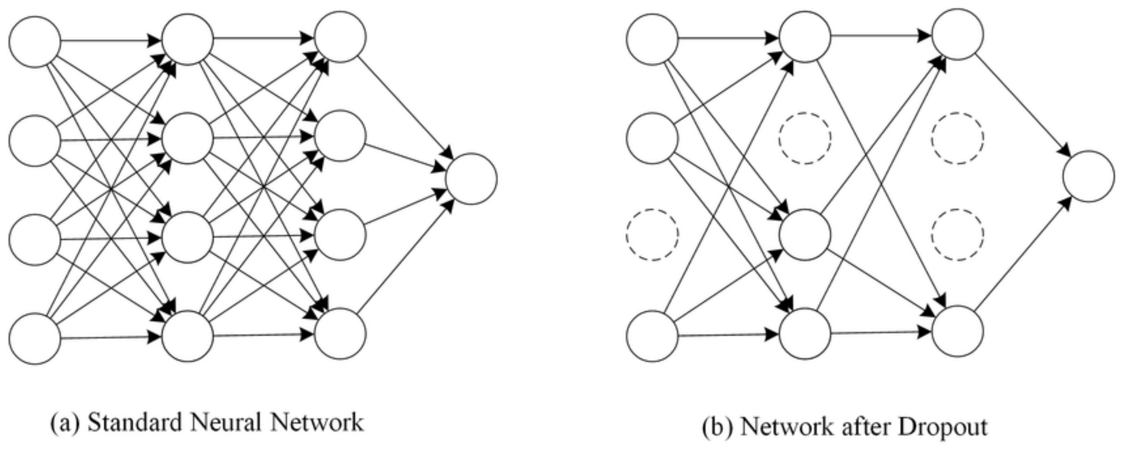
$$W_{i,j} \sim U\Big(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}} \Big) \tag{2.18}$$

Because of the many trade-offs that need to be made while choosing the right initialization technique, it sometimes is a good idea to treat the initial scale of weights and biases as a *hyperparameter* - a variable based on which different models are proposed. However, due to the limits of computational resources, very often such manoeuvre is unfeasible.

2.4.2 Dropout

First introduced by (Srivastava et al. 2014), dropout is a method of preventing neural networks from overfitting. It introduces a computationally inexpensive concept of 'dropping' certain randomly chosen connections between neurons during the training process. In each iteration, every neuron in the network is dropped with a certain probability p. Then, the learning operation is performed without updating weights and biases associated with that neuron as shown in the Figure 2.3 below:

Figure 2.3: Learning process with dropout

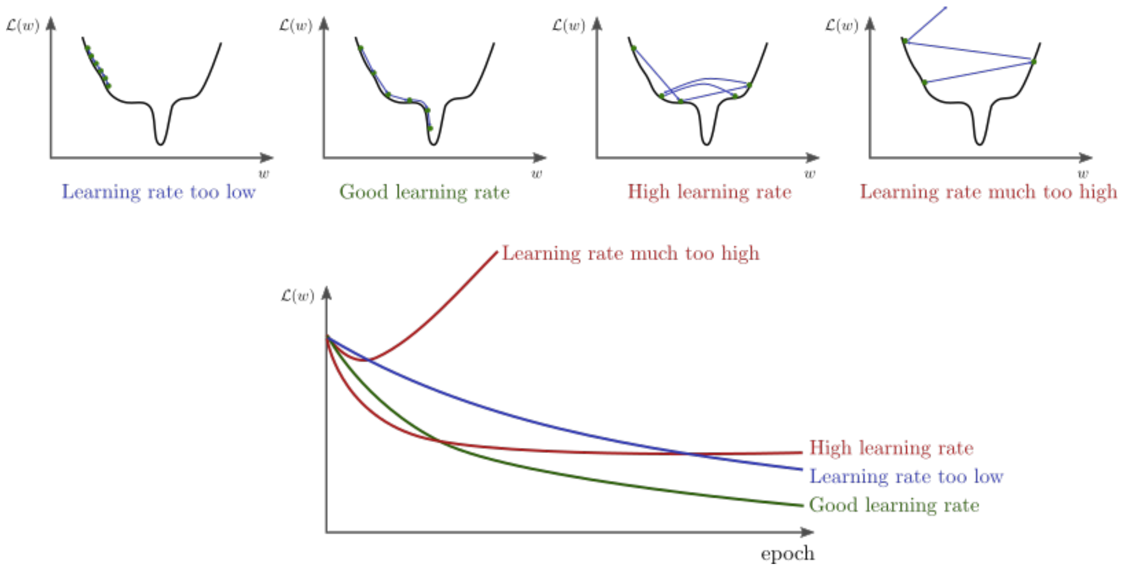


This particular approach makes it harder for neurons to simply 'memorize' training examples, and forces it to learn a feature in the data, so that it contributes towards providing the right answer. This technique is further argued by (ibid.) to not only reduce the extent of overfitting, but also to speed up the learning process. Many test have shown that it is one of the most efficient regularization methods. Nowadays, there is a wide range of literature concerning the most optimal dropout rate p, with recommendations of p being a training hyperparameter on its own. (Srivastava et al. 2014) suggest starting with an initial dropout of $p = 0.5$.

2.4.3 Momentum Learning Adjustments

Recall the equations 2.13 and 2.14 describing the learning process for weights and biases. For both these procedures, a learning rate parameter ϵ must be set. It sets a 'step-size' with which weights and biases converge towards the direction of steepest descent of the loss function. Figure 2.4 illustrates well the importance of choosing a correct learning rate.

Figure 2.4: An illustration of learning rates performances



Oftentimes, the network needs to take bigger steps initially, to avoid being stuck in a local minimum, but then requires smaller steps as it approaches the global minimum. It is a commonly used technique to decay the learning rate in a linear fashion through all k iterations starting from an initial rate ϵ_0 up until ϵ_T.

$$\epsilon_k = (1 - \frac{k}{T})\epsilon_0 + \frac{k}{T}\epsilon_T \tag{2.19}$$

Again, there is no scientific consensus confirming what learning rate is the most optimal, making it more kind of an art rather than a science. Because the choice of initial parameters ϵ_0, ϵ_T and T is somewhat arbitrary and still allows for unstable oscillations for high values of ϵ_0, a momentum-based techniques have been developed.

(Polyak 1964) has designed a method that accelerates learning in settings where there's high curvature and noisy gradients. It is based on exponentially decaying moving average (EMA) of previous gradients. Generalizing equations 2.13 and 2.14, the learning process can be represented using the set of weights and biases $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}^* = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} \tag{2.20}$$

Where $\nabla_{\boldsymbol{\theta}}$ is a gradient obtained in the training process. The momentum method formally adds a $\boldsymbol{v}$ parameter, which in physical terms would represent velocity. In the context of gradient learning, it represents an EMA of previous gradients applied to the network. In this instance, the update is given by

$$\boldsymbol{\theta}^* = \boldsymbol{\theta} - \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \tag{2.21}$$

Where the hyperparameter $\alpha \in [0, 1]$ determines the decay rate of previous gradients. The larger the α, the slower the decay of previous direction and the less volatile changes to the direction of descent. Similar to other methods, there is no free lunch when it comes to the choice of α. It must be tuned for a specific problem.

Since the introduction of this technique, many improvements to the method have been suggested. One of them is Nesterov momentum (I. Sutskever et al. 2013), which follows the same idea, with slight changes as to where the gradient is evaluated. More novel adaptive learning algorithms largely build up on the same idea as presented by (Polyak 1964). The AdaGrad (standing for Adaptive Gradient) algorithm introduced by Duchi, Hazan and Singer 2011 adapts the learning rate with respect to the square root of all previous squared values of the gradient. This results in greater learning rates for parameters with bigger partial derivatives. RMSProp (G. Hinton 2012) is a variation of AdaGrad with exponentially decaying previous values of the gradient. It was tested (G. Hinton 2012) to perform better in a non-convex setting. Finally, the most prominent learning algorithm, Adam (standing for Adaptive Moments), is a yet another variation of RMSProp and momentum method, which includes first-order moments (Kingma and Ba 2014). It is considered a more robust choice, since RMSProp is prone to a high bias early in the training process.

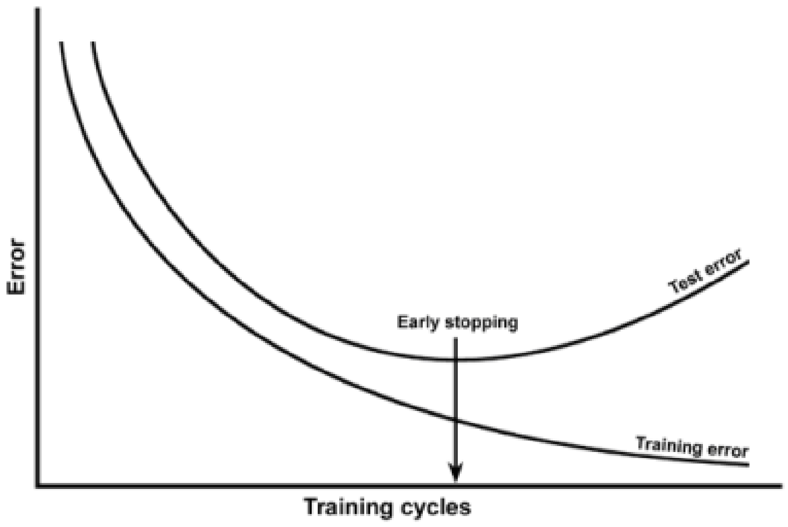
Adaptive learning algorithms are still a dynamically researched area. In (Schaul, Antonoglou and Silver 2014) authors compared different learning algorithms, however no clear choice has emerged from their findings. Therefore, the choice of the learning algorithm is still largely a subject of tuning.

2.4.4 Early Stopping

As mentioned before, training a neural network over the course of multiple epochs can be a desirable choice, since it can learn to pick up more patterns and better understand the complexities of data. The choice of the number of training epochs is a hyperparameter in itself. However, there are always limits to the final accuracy of the model on a validation set. Learning algorithms are designed, such that they always reduce the error on the training set. During the initial phases of training, the reduction in the training error largely goes in line with reducing the error of the test set (test and validation terms are here used interchangeably), hence also increasing the accuracy of prediction on 'unseen' data. The extent to which 'pattern learning' occurs is limited by the train set robustness. Later in the process the network tends to further reduce the training error, with no further (or detrimental) effect to the validation error (Perlich 2011).

Early stopping (Lee et al. 2001) is a technique that allows the training to take place as long as it sees an improvement in the performance on a test set. Once the parameters show no sign of improvement and the test set accuracy plateaus, the training is stopped, and the model with the best recorded validation set accuracy is returned. It is an efficient way of finding the right hyperparameter, and unlike many others, does not require time-consuming guess and check processes.

Figure 2.5: Theoretical behaviour of train and test errors relative to the training time



2.5 LSTM Network

In previous sections the structure and behaviour of the most simplistic Dense Neural Networks was described. Although they perform well in numerous settings, some tasks require different, more complex variations.

Long-Short Term Memory (LSTM) networks are a specific case of a wider subclass of networks called Recurrent Neural Networks (RNN). Introduced by (Hochreiter and Schmidhuber 1997) are an improved version of recurrent nets suffering from a vanishing gradient problem, whereby the errors flowing backwards through time eventually vanish. This problem is solved via the use of a *gated recurrent unit.* They control inputs getting into the memory unit, and prevent them from taking up unnecessary information.

This architecture has been widely successful in areas like handwriting recognition (Voigtlaender, Doetsch and Ney 2016), handwriting generation (Graves 2013), speech recognition (Graves, Mohamed and Geoffrey Hinton 2013), machine translation (Ilya Sutskever, Vinyals and Le 2014) and many more.

2.6 ANN in Financial Time-Series Forecasting

Over the years, various different applications of ANNs have been present in the artificial intelligence industry. Most notably, speech recognition has shown great progress. Variability of different techniques, including Hidden Markov Models, Dynamic Time Wrapping or Vector Quantization, emerged (Balaji and Baskaran 2013). Also, image processing has been a widely researched area, with its advances in emotion recognition, pattern detection and image reproduction. In that field, the convolutional structure of the neural network has proven to be the most efficient one in comparison to others like dense layers or recurrent neural networks (Amberkar et al. 2018). Also the financial industry turned out to be an area, in which neural networks can achieve satisfying results.

Despite the ANN ability to learn non-linear patterns and achieve high levels of accuracy in comparison with other models, some researchers question its ability to accommodate non-linear time-series patterns in its traditional form. In (Tealab, Hefny and Badr 2017), authors have shown that using pure dense layers to capture non-linearities in time series models is ineffective. In this particular instance, neural networks perform poorly creating a need to develop more time-series orientated models using hybrid technologies such as Fuzzy Logic.

(Munim, Shakil and Alon 2019) compared the autoregressive integrated moving average (ARIMA) model with neural network autoregression (NNAR) models. To measure accuracy of both models, they decided to introduce Mean Absolute Scaled Error alongside more popular error measurement techniques, namely Root Mean Squared Error and Mean Absolute Percent Error to diversify measurement results. The study was concluded using training dataset covering 2,000 days of bitcoin trading, and the errors were compared. Despite the neural network complexity, ARIMA model turned out to yield smaller errors for all of the three measurement techniques, regardless of the test sample size. To confirm that ARIMA model indeed performs better, Diebold-Mariano Test has been carried out, in each case yielding the p-value below 0.01.

On the contrary, (McNally, Roche and Caton 2018) arrived at different results by using slightly different architectures, namely Bayesian optimized recurrent neural network (RNN) and a Long Short Term Memory (LSTM) network versus the ARIMA model. Their method involves CRISP mining technology and covers three years' worth of bitcoin price data. The data was standardized for the activation functions to work well. The models output was designed to indicate whether the price of bitcoin will go up or down the next day. The results have shown, that the ARIMA model gives the correct prediction only 50.05% of times. Both RNN and LSTM score better results – 50.25% and 52.78% respectively. However, for longer-term predictions LSTM showed higher accuracy than RNN. This research shows that the use of neural networks in forecasting is absolutely validated and is capable of giving accurate predictions.

A similar study (Radityo, Munajat and Budi 2017) has been performed using a broader range of neural network techniques, including Backpropagation Neural Network (BPNN), Genetic Algorithm Neural Network (GANN), Genetic Algorithm Backpropagation Neural Network (GABPNN) and Neuroevolution of augmenting topologies (NEAT). To evaluate the performance of these models, solely MAPE is used, similarly to the previous study (Munim, Shakil and Alon 2019). The networks were trained for 5,000 epochs, apart from the BPNN, where the limit was 10,000 epochs. GABPNN turned to be the best architecture in this paper with MAPE of 1.883%. BPNN and NEAT performed slightly worse, with a MAPE of 1.998% and 2.175% respectively. The outlier in this study, GANN had a MAPE of 4.461%, making it the worst choice of architecture. Interestingly enough, GANN performed better in the case of stock market (Nayak, Misra and Behera 2012). This shows, that there is a fundamental difference in both markets linked to the volatility of cryptocurrencies, making it impossible for the mutation operations to learn the patterns. It is important to mention, that in this study many hyperparameters of the network have not been optimized, including for instance the number of layers or nodes per layer leaving a significant room for improvement in accuracy.

(Yiying and Yeze 2019) in their study investigated the performance of fully connected ANN and LSTM recurrent Neural Network to analyse the price dynamics of Bitcoin. They used 1030 of trading days data, which was then divided into 80:20 ratio of train and test data to avoid overfitting. The results were evaluated using the mean squared errors. In their study, both ANN and LSTM yielded similar results despite different internal structures, with LSTM performing better in picking up short-term dynamics. Despite that, ANN can achieve similar benchmarks given enough historical data.

The majority of the work in predicting the future cryptocurrency movements has been done using day-long time horizons. This is an ambitious time span to model, since unexpected news may appear during the day, significantly affecting the future price of a cryptocurrency. This paper focuses on shorter time horizons. Also, most of the research papers tend not to follow a consistent performance measure, focusing on different aspects of model accuracy, hence making it extremely difficult to compare findings. It is also difficult to assess how each of the models mentioned above would perform given a trading task. Performance evaluation techniques used to assess the model constructed in this paper is covered in sections 3.4 and 3.5.

3 Implementation

This section outlines how the theory covered in the previous section is applied to the real-world dataset. The choice of neural net architecture, learning process and other features are described, and evaluation metrics are introduced to compare the performance of different models tested in this research.

3.1 Data

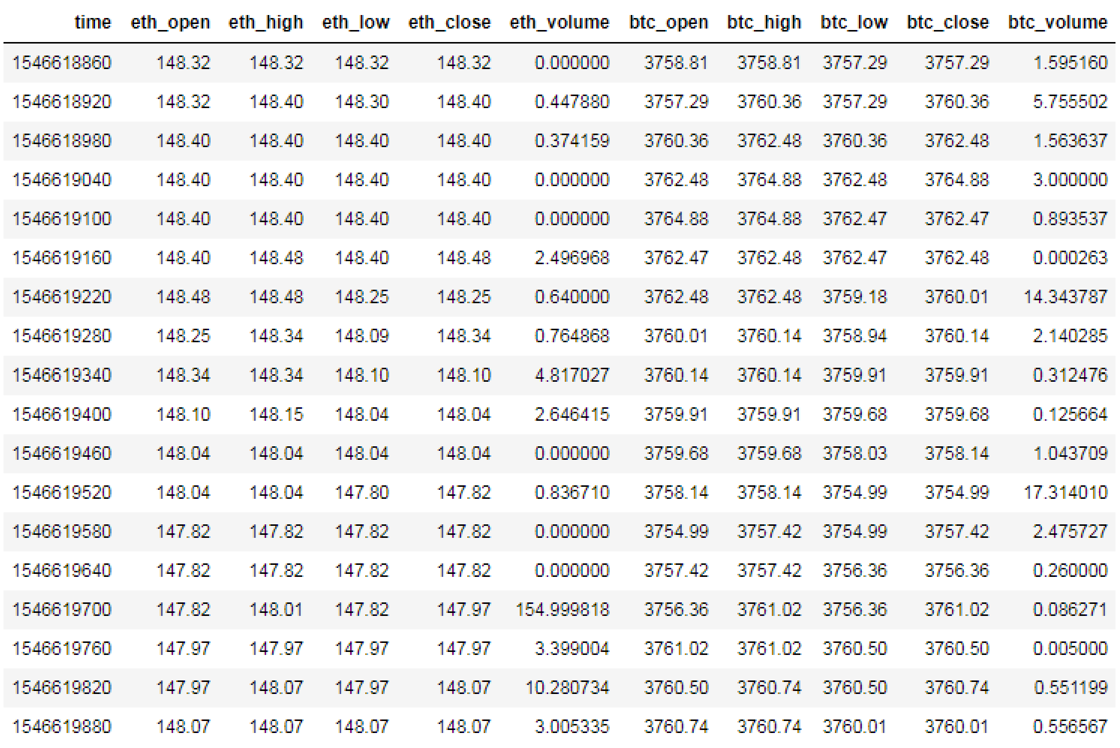
The raw data used for this paper consists of minute-by-minute prices of different cryptocurrencies expressed in US dollars. It covers the time period of 2018 and 2019. Each minute-long time window dataset consists of the following fields:

- Timestamp - the beginning of the time window expressed in unix time
- Open price - the last price, at which a trade took place before the time window began.
- High price - the highest price, at which the trade took place during the time window.
- Low price - the lowest price, at which the trade took place during the time window.
- Close price - the last price, at which a trade took place before the end of the time window.
- Volume - the total amount of cryptocurrency that was traded within the time window.

One of the positive aspects of this dataset is the fact, that it represents a time-series with equidistant time points, which reduces the amount of work that needs to be done in terms of preprocessing - see Figure 3.1.

In terms of negatives, some of the data points are missing form the dataset, whereas this is rarely the case for the BTC/USD and ETH/USD pairs, the LTC/USD (LTC - Litecoin) pair is relatively sparsely populated, making it difficult to efficiently implement. For that reason, only BTC/USD and ETH/USD pairs are used as an

Figure 3.1: A snapshot of the price data



time	eth_open	eth_high	eth_low	eth_close	eth_volume	btc_open	btc_high	btc_low	btc_close	btc_volume
1546618860	148.32	148.32	148.32	148.32	0.000000	3758.81	3758.81	3757.29	3757.29	1.595160
1546618920	148.32	148.40	148.30	148.40	0.447880	3757.29	3760.36	3757.29	3760.36	5.755502
1546618980	148.40	148.40	148.40	148.40	0.374159	3760.36	3762.48	3760.36	3762.48	1.563637
1546619040	148.40	148.40	148.40	148.40	0.000000	3762.48	3764.88	3762.48	3764.88	3.000000
1546619100	148.40	148.40	148.40	148.40	0.000000	3764.88	3764.88	3762.47	3762.47	0.893537
1546619160	148.40	148.48	148.40	148.48	2.496968	3762.47	3762.48	3762.47	3762.48	0.000263
1546619220	148.48	148.48	148.25	148.25	0.640000	3762.48	3762.48	3759.18	3760.01	14.343787
1546619280	148.25	148.34	148.09	148.34	0.764868	3760.01	3760.14	3758.94	3760.14	2.140285
1546619340	148.34	148.34	148.10	148.10	4.817027	3760.14	3760.14	3759.91	3759.91	0.312476
1546619400	148.10	148.15	148.04	148.04	2.646415	3759.91	3759.91	3759.68	3759.68	0.125664
1546619460	148.04	148.04	148.04	148.04	0.000000	3759.68	3759.68	3758.03	3758.14	1.043709
1546619520	148.04	148.04	147.80	147.82	0.836710	3758.14	3758.14	3754.99	3754.99	17.314010
1546619580	147.82	147.82	147.82	147.82	0.000000	3754.99	3757.42	3754.99	3757.42	2.475727
1546619640	147.82	147.82	147.82	147.82	0.000000	3757.42	3757.42	3756.36	3756.36	0.260000
1546619700	147.82	148.01	147.82	147.97	154.999818	3756.36	3761.02	3756.36	3761.02	0.086271
1546619760	147.97	147.97	147.97	147.97	3.399004	3761.02	3761.02	3760.50	3760.50	0.005000
1546619820	147.97	148.07	147.97	148.07	10.280734	3760.50	3760.74	3760.50	3760.74	0.551199
1546619880	148.07	148.07	148.07	148.07	3.005335	3760.74	3760.74	3760.01	3760.01	0.556567

input to our model.

3.2 Data preprocessing

3.2.1 Data labelling

Before training of the network can happen, the objective and data input need to be explicitly specified. Recalling eq. 1.1, the goal of this model is to predict the direction of movement of the price of cryptocurrency within the time Δt.

$$\hat{y}_{t+\Delta t} = f(\mathbf{x}_t) \tag{3.1}$$

To perform empirical tests, we need to specify the time horizon over which we try to predict the movement. For the purposes of this paper, 1-minute, 3-minute, and 10-minute horizons were picked. Based on that, the closing price within time

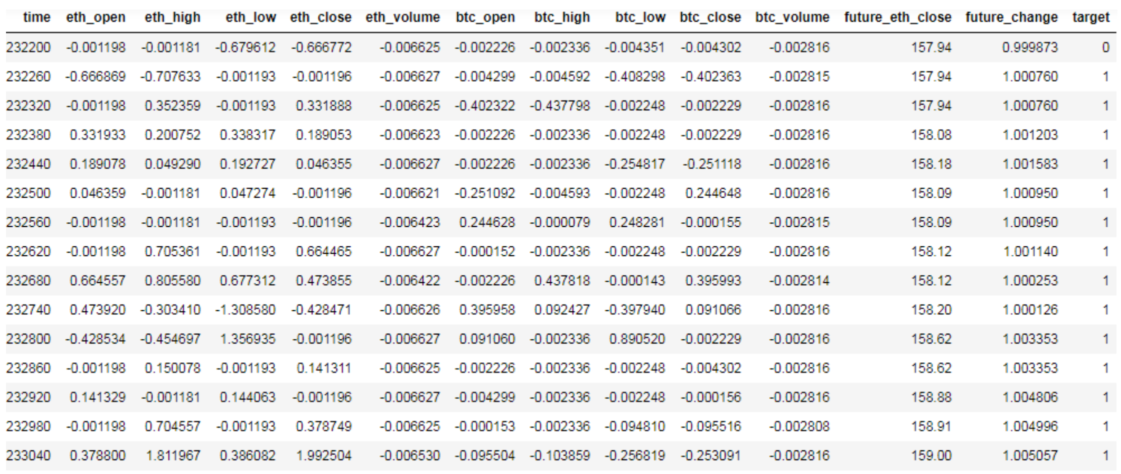
t is divided by the opening price at time t to create a new feature called *future change*. Based on this feature, a *target* feature is created taking a value of 1 if the *future change* is greater than 1 and 0 otherwise, as follows from eq. 1.2. Notably, the information $\mathbf{x}_t$ also needs defining, since the information available at time t consists of all trades that happened in the past. In this paper, the model we take only information up to τ minutes back, which for empirical purposes is chosen to be 45 minutes.

3.2.2 Feature tuning

In order for the neural network to start learning efficiently, features passed as an input need to be normalized to ensure an optimal learning trajectory for weights and biases (Bhanja and Das 2018).

In this paper, nominal prices and volumes of cryptocurrencies are replaced by a minute-by-minute percentage changes, so that any feature (excluding time, and target features) $x_{i,t}$ is replaced by a new feature $x^*_{i,t} = \frac{x_{i,t}}{x_{i,t-1}}$. Furthermore, because minute-by-minute price swings tend not to be very volatile, and almost all feature values were centered around 1, normalization was done using the scikit-learn library, which Gaussian-normalized values as in Figure 3.2.

Figure 3.2: A snapshot of the preprocessed data



time	eth_open	eth_high	eth_low	eth_close	eth_volume	btc_open	btc_high	btc_low	btc_close	btc_volume	future_eth_close	future_change	target
232200	-0.001198	-0.001181	-0.679612	-0.666772	-0.006625	-0.002226	-0.002336	-0.004351	-0.004302	-0.002816	157.94	0.999873	0
232260	-0.666869	-0.707633	-0.001193	-0.001196	-0.006627	-0.004299	-0.004592	-0.408298	-0.402363	-0.002815	157.94	1.000760	1
232320	-0.001198	0.352359	-0.001193	0.331888	-0.006625	-0.402322	-0.437798	-0.002248	-0.002229	-0.002816	157.94	1.000760	1
232380	0.331933	0.200752	0.338317	0.189053	-0.006623	-0.002226	-0.002336	-0.002248	-0.002229	-0.002816	158.08	1.001203	1
232440	0.189078	0.049290	0.192727	0.046355	-0.006627	-0.002226	-0.002336	-0.254817	-0.251118	-0.002816	158.18	1.001583	1
232500	0.046359	-0.001181	0.047274	-0.001196	-0.006621	-0.251092	-0.004593	-0.002248	0.244648	-0.002816	158.09	1.000950	1
232560	-0.001198	-0.001181	-0.001193	-0.001196	-0.006423	0.244628	-0.000079	0.248281	-0.000155	-0.002815	158.09	1.000950	1
232620	-0.001198	0.705361	-0.001193	0.664465	-0.006627	-0.000152	-0.002336	-0.002248	-0.002229	-0.002816	158.12	1.001140	1
232680	0.664557	0.805580	0.677312	0.473855	-0.006422	-0.002226	0.437818	-0.000143	0.395993	-0.002814	158.12	1.000253	1
232740	0.473920	-0.303410	-1.308580	-0.428471	-0.006626	0.395958	0.092427	-0.397940	0.091066	-0.002816	158.20	1.000126	1
232800	-0.428534	-0.454697	1.356935	-0.001196	-0.006627	0.091060	-0.002336	0.890520	-0.002229	-0.002816	158.62	1.003353	1
232860	-0.001198	0.150078	-0.001193	0.141311	-0.006625	-0.002226	-0.002336	-0.002248	-0.004302	-0.002816	158.62	1.003353	1
232920	0.141329	-0.001181	0.144063	-0.001196	-0.006627	-0.004299	-0.002336	-0.002248	-0.000156	-0.002816	158.88	1.004806	1
232980	-0.001198	0.704557	-0.001193	0.378749	-0.006625	-0.000153	-0.002336	-0.094810	-0.095516	-0.002808	158.91	1.004996	1
233040	0.378800	1.811967	0.386082	1.992504	-0.006530	-0.095504	-0.103859	-0.256819	-0.253091	-0.002816	159.00	1.005057	1

The goal of the neural network is to predict the target value based on the last τ

minutes of the 10 non-target feature values related to price and volume. The data from 2018 is split into train and test sets. The 2019 data is left out completely as another evaluation sample to see whether validation accuracy represents similar levels irrespective of the year. The target variable y comes from the *target* column. Each target value y_t is associated with an information set put into a $\tau \times 10$ matrix $\mathbf{X}_t$. This information is then fed through the neural network to obtain a probability distribution $\hat{\mathbf{y}}_{t+\Delta t} = (\hat{y}^{k=0}_{t+\Delta t}, \hat{y}^{k=1}_{t+\Delta t})$.

One very important aspect of training a deep learning model, is ensuring the training set balance. An imbalanced dataset can possibly lead to unwanted, sub-optimal solutions that predict only one majority class no matter the initial input (Wang et al. 2016). To account for this scenario, the training set has been equalized so that it consists of exactly the same number of instances of price going up or down. This measure is not needed in the test set, since it does not affect the structure of the model. Another dimension of the same problem is the order of the training set. If training examples are passed on a minute-after-minute basis, the model can start picking up a short-term trend, which might be only specific to a certain month, or a week, and then start switching to another trend later on, eventually not accommodating universal signals. A common way to address this issue is training set reshuffling (Meng et al. 2017). This operation forces the neural network to learn by observing training examples from random periods over the training set. That way, the network does not pick up on the short-term trends appearing in the data, but rather looks for universal signals from the information set that might suggest an upward or downward movement.

Such balanced and reshuffled training set is then ready to be used in the training process.

3.3 Neural Net Structure

The neural net used for this classification task is a mixture of LSTM and dense layers. The initial hidden LSTM layers read off the dynamics of feature changes for the span of the last t minutes. The input data is processed in a layer-wise fashion, where each layer provides the representation of the market state from any given minute that is used for the next layer. Instead of just applying LSTM layers with a softmax activation function at the end, two additional dense layers were added before the final 2-node output layer. Rectified Linear Unit activations where used for the final dense hidden layers in each model. Different combinations of layer structures have been tried to find the optimal structure. To train the model, the Adam optimizer has been used, and dropout was applied to hidden layers with varying rates, to allow for robustness of the training process. The training examples are passed in batches to reduce the training variance.

3.4 Performance Measurement of Models

There is a wide range of methods used to evaluate the performance of a deep learning model. One of the most prominent ways to judge how the classification model is doing, is to look at the overall validation set accuracy, which tells how many times the model predicted a correct class from the test set. However, this may cause potential problems if the predictions are imbalanced. The model might predict one class most of the times and achieve high accuracy benchmarks solely due to the fact that the class it predicts constitutes a majority of the test set. Therefore, in contrast to many papers covering similar problems, accuracy in this paper is used as a performance guideline, but not as the main measure.

A method that accounts for prediction imbalances is a *confusion matrix*, also known as a *error matrix*, see Table 3.1. It accounts for the accuracy shortcomings by breaking down the *predicted classes* and *actual classes*. In the context of a problem presented in this paper True Positives are cases, where an upward movement was classified correctly. True Negatives are instances, where the downward movement was correctly identified. False Negative occurs when the model predicts downward movement, when upward actually happens, and False Positive is when downward movement was classified as upward.

class	**predicted 0**	**predicted 1**
actual 0	True Negative (TN)	False Positive (FP)
actual 1	False Negative (FN)	True Positive (TP)

Table 3.1: Confusion Matrix

Two very useful measures can be derived from the above matrix. One of them is *Recall*, which measures the True Positive Rate i.e. the probability of detecting an upward movement if one actually occurs.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

On the other hand, *Precision* indicates what is the Positive Predictive Value, i.e. how many times there actually is an upward movement, given the upward movement is predicted.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.3}$$

In the context of cryptocurrency trading, both measures are important. Recall explains how often the model will notify about a future spike in price if one is about to happen. Low values of recall mean that a trader would be missing out on a lot of occasions. Precision on the other hand, informs how accurate the model is, when it predicts a spike in the price. Low values of precision indicate that the model is very risky and ineffective.

There is also another dimension to measuring models' performance. Recall that the network's output is a probability distribution $\hat{\mathbf{y}}_{t+\Delta t}$. Given the information set, a network assigns a certain probability to the event of price going up and price going down. In principle, if both values are around 0.5, it is an indication that the model does not favour either scenario very much. However, if the ratio changes to 0.75:0.25, the model deems one of the outcomes three times as likely as the other one. While evaluating the performance of a classifier, we can investigate different 'certainty' thresholds and see how the model performs, allowing for different levels of certainty. One could expect that the accuracy of the model should be higher, if asked exclusively for the most certain 20% of instances, rather for the entire sample. This angle of evaluation does not appear in the papers cited above, yet transmits a lot of information about model's capability to assess probability of outcomes. If the accuracy of the model is indeed higher for the most certain 20% of predictions, then that would mean that a higher probability assigned by the model indeed reflects a higher probability of a certain outcome occurring with time Δt.

3.5 Trading simulation

When a trader is hired by a bank or a hedge fund, their performance is evaluated by their ability to create high rates of return, not by how many times were they right or wrong in their decisions. Similarly, it should be the core interest to see how the neural net performs, if it is given a trading task in the actual market. In this paper, each model trained on the subset of 2018 dataset is put in a position of a trader, and its performance is given by the average daily rate of return (ADRR).

Since the dataset used in this paper does not include the order book history associated with this particular exchange, certain simplification must be put in place in order to perform a theoretical trading simulation:

a) The market is infinitely deep with no bid-ask spread associated with the deficit of buyers and sellers. In other words, it is possible to sell or buy an infinite amount of currency at a market price.

b) There are no transaction costs associated with buying or selling the currency.

c) The trader can enter either a long or short position at any time (no restrictions on short selling).

d) The trader buys (short sells) at an open price and sells (buys back) at a close price.

In this theoretical framework, the neural network activity is as follows:

1. Start off with an initial amount of capital K.
2. Analyze the last τ minutes of prices and return a prediction whether the price will go up or down in the next Δt minutes.
3. If the model predicts up - buy $\frac{K}{\Delta t}$ of currency, if down, short sell $\frac{K}{\Delta t}$.
4. After Δt has passed, close the position irrespective of the return.
5. Go onto the next minute and repeat the process for a duration of one day.

Since there are 24×60 minutes in a day, so the model makes 1440 trades a day.

3.6 Performance Benchmarks

To decide whether the model presented in this research has any predictive power, certain performance benchmarks are introduced, which the model is compared against. Since there are only two classes - up or down - the most basic strategy would be random guessing, which by default should give a 50% accuracy. The model's accuracy should also be significantly higher than a pure strategy of picking only ups or downs. Over 2019, which is the validation set, the price went down 50.013% of the time, which is very close to the default 50% benchmark. Another very important aspect of the model's performance is its ability to achieve higher accuracy standards when given higher 'certainty thresholds'. If the accuracy is in no way correlated with the certainty of predictions returned by the network, that would indicate no predictive power of the network's output.

In the case of a trading simulation, the model needs to perform better than a pure long/short position kept over the course of a year. The price of Ethereum started with \$140.82 on the 1^{st} of January 2019 and ended up at \$129.61 on the 1^{st} of January 2020. Holding a short position would bring 8.649% of return, which corresponds to 1.000227 ADRR. Hence the average daily rate of return attained by the model needs to be higher than this to confirm its validity.

4 Results

Different architectures were tried to find the most effective model. Here, three main architectures seemed to suit this problem best. Below their structure is presented in the form 'n-LSTM-m-Dense'. Such naming convention represents a network composed of n LSTM layers and m Dense layers. To specify the number of neurons per layer 'LSTM(X)-...-Dense(Y)' description is given.

1. 3-LSTM-2-Dense - LSTM(128)-LSTM(128)-LSTM(128)-Dense(64)-Dense(32)
2. 3-LSTM-3-Dense - LSTM(128)-LSTM(128)-LSTM(128)-Dense(64)-Dense(32)-Dense(32)
3. 4-LSTM-2-Dense - LSTM(128)-LSTM(128)-LSTM(128)-LSTM(64)-Dense(64)-Dense(32)

Each of those networks was trained over the course of 10 epochs. After each epoch, the model was recorded and its validation accuracy stored to compare against other models.

4.1 Models' Performance

The 2018 validation set accuracy of each network over 10 epochs is shown in Table 4.1. Every entry contains the accuracy of a different model.

epoch	3-LSTM-2-Dense	3-LSTM-3-Dense	4-LSTM-2-Dense
1	0.509	0.506	0.502
2	0.534	0.515	0.527
3	0.521	0.523	0.514
4	0.531	0.513	0.523
5	0.544	0.534	0.499
6	**0.548**	0.534	0.510
7	0.542	**0.549**	0.519
8	0.542	0.532	0.511
9	0.536	0.543	**0.528**
10	0.526	0.539	0.527

Table 4.1: 2018 Validation set accuracy

Similar evaluation is presented with models' accuracy when evaluated over the 2019 dataset.

epoch	3-LSTM-2-Dense	3-LSTM-3-Dense	4-LSTM-2-Dense
1	0.522	0.505	0.520
2	0.528	0.523	0.546
3	0.527	0.508	0.506
4	0.552	0.503	0.544
5	0.540	0.527	0.507
6	0.547	0.549	0.498
7	**0.564**	0.551	0.508
8	0.538	0.534	0.504
9	0.548	**0.560**	0.525
10	0.537	0.540	**0.548**

Table 4.2: 2019 Validation set accuracy

The 3-LSTM-3-Dense network on the 7th epoch of training achieved the highest 2018 validation set accuracy, closely followed by the 6th epoch of the 3-LSTM-2-Dense network. On the 2019 validation set 3-LSTM-3-Dense performed better than 3-LSTM-2-Dense as well. These results suggest a relationship between the complexity of the network, and time needed to achieve its best performance. However, after a certain accuracy threshold is reached, the network starts to overfit by increasing the in-sample accuracy and deteriorating its generalization abilities. This is well illustrated by investigating the confusion matrices of each network at different epochs.

Starting off with the 3-LSTM-2-Dense network, there is a clear difference in terms of prediction balance as shown in Tables 4.3 and 4.4. In the 4th epoch the network predicts downward movement 56.6% of the time. However, this bias increases to 73.4% in the 10th epoch, showing signs of converging towards producing only one prediction, despite the fact that actual movements are split fairly evenly.

class	**predicted 0**	**predicted 1**
actual 0	9285	5767
actual 1	7688	7260

Table 4.3: 3-LSTM-2-Dense epoch 4 - 0.552 accuracy

class	**predicted 0**	**predicted 1**
actual 0	11590	3462
actual 1	10434	4514

Table 4.4: 3-LSTM-2-Dense epoch 10 - 0.537 accuracy

The same scenario takes place for the 3-LSTM-3-Dense network, where the bias towards predicting downward movement is only 58.4% in its best 9^{th} epoch, and suddenly jumps up to 72.5% in the 10^{th} epoch of training as shown in Tables 4.5 and 4.6.

class	**predicted 0**	**predicted 1**
actual 0	9687	5365
actual 1	7827	7121

Table 4.5: 3-LSTM-3-Dense epoch 9 - 0.560 accuracy

class	**predicted 0**	**predicted 1**
actual 0	11498	3554
actual 1	10254	4694

Table 4.6: 3-LSTM-2-Dense epoch 10 - 0.540 accuracy

However, this problem is best represented by the 4-LSTM-2-Dense network, where the bias towards *downward* movement is 96.9% in the 5^{th} and 94.1% towards *upward* movement in the 7^{th} epoch. This underlines the importance of investigating the confusion matrix to make sure the answers given by the network are not skewed towards any particular class.

Figures 4.1 and 4.2 represent models' accuracy with respect to the percentage of the most certain predictions as given by the probability distribution returned. The value of 1.0 on the horizontal axis corresponds to the entire validation sample, whereas the value of 0.4 corresponds to the 40% of the sample predictions having the highest probabilities. All models tend to have an increasing accuracy when considering a narrower subset of validation cases. For instance, the 3^{rd} epoch of the 3-LSTM-2-Dense network has an accuracy of 52.8% when evaluated over the entire sample. This accuracy jumps up to over 63% when evaluated over the subset of 20% of the most certain predictions as shown by the dark green curve in Figure 4.1.

Figure 4.1: 3-LSTM-2-Dense models' accuracy versus x% of the most certain predictions

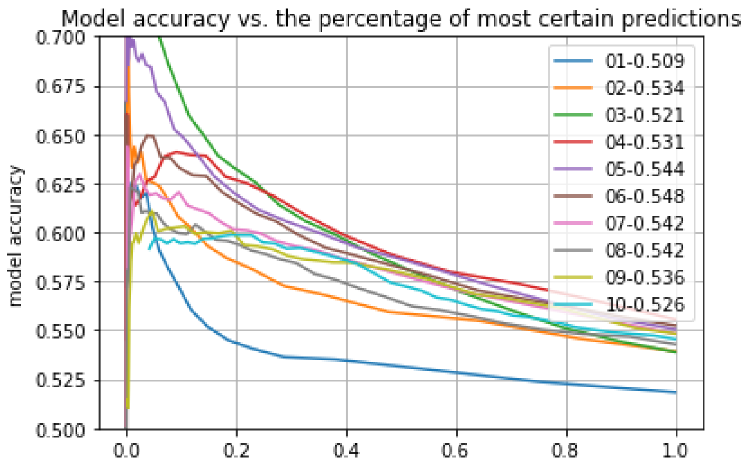
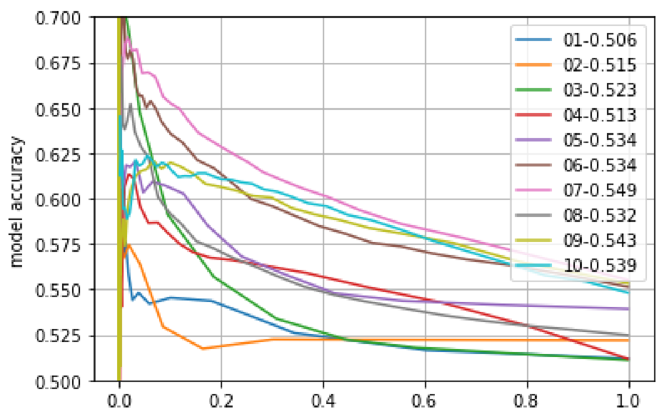


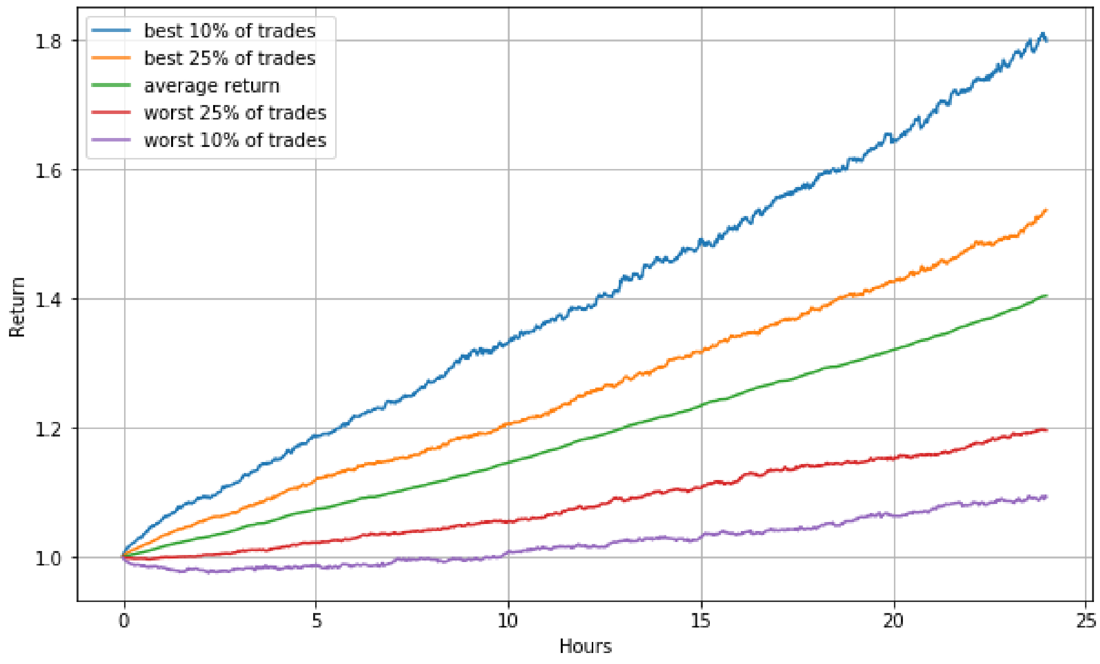
Figure 4.2: 3-LSTM-3-Dense models' accuracy versus x% of the most certain predictions



4.2 Trading Performance

The 9th epoch of the 3-LSTM-3-Dense network was tested against a trading task over the 2019 validation set. This was described in Section 3.5. The averaged results for all trading days of 2019 are presented in Figure 4.3.

Figure 4.3: Trading performance of the 3-LSTM-3-Dense network



On average, this theoretical trading methodology used in the paper was yielding 40.7% of average daily rate of return. Interestingly enough, the average return of the worst performing 10% of days were bringing an ADRR of 9.3%.

5 Discussion

This chapter comments on findings from the previous section, focusing on classification and trading performance. Suggestions for future work and model improvements are made.

5.1 Model Classification Performance

By comparing the accuracy results of different validation samples in Tables 4.1 and 4.2, it becomes apparent that deep learning models significantly outperform random guessing technique. Obtaining similar accuracy levels on both test sets highlights the robustness of the model and suggests that the model did well in capturing complex, non-linear patterns indicating certain price movements regardless of short-term trends. Interestingly, models that tend to do well in one test set, do also relatively well in the second. Spearman's Rank Correlation Coefficient for the 3-LSTM-2-Dense architecture is 0.6091 and for the 3-LSTM-3-Dense - 0.9242.

The evolution of these three networks suggests that overly deep structures fail to learn the patterns efficiently and are prone to big swings in behaviour over the course of training. Judging by the validation accuracy, 10 epochs was an optimal training period, which allowed to find the model that generalized data well. In both 3-LSTM-2-Dense and 3-LSTM-3-Dense cases, later epochs started to show signs of overfitting. However, the 4-LSTM-2-Dense network, exhibited erratic behaviour over the entire training process returning very good and very poor accuracy almost alternately. A longer training process would be needed to assess whether this network architecture reaches its optimum very early in the process, or needs further learning.

The results obtained in Tables 4.3 and 4.5 suggest a healthy balance between prediction choices of the models. The precision and recall in 4.3 are 0.557 and 0.486, which is similar to those in 4.5 with 0.570 and 0.476. This means the networks are right in predicting an upward movement more than 50% of the time. However, they only identify less than 50% of actual upward movements. In Tables 4.4 and 4.6 this balance changes, so that the precision and recall values are 0.566, 0.302

and 0.569, 0.314 accordingly. This shows a significant deterioration in networks' ability to correctly detect an upward movement for no real improvement in the accuracy of correct upward guesses.

Graphs presented in Figures 4.1 and 4.2 reflect a very intuitive, yet interesting relationship. In Section 3.4 it has been suggested that models' accuracy should increase in line with higher probabilities (lower entropy) returned in the output layer. This phenomenon actually takes place in the case of almost all models produced by the two networks. It constitutes a very strong argument in favour of the hypothesis that the probability returned by the network indeed imitates the true probability. This piece of information is an important aspect of the models' nature, yet is not widely used in research papers concerning deep learning time-series models.

5.2 Trading Simulation Performance

It is important to point out that this trading simulation is a purely theoretical process bypassing many obstacles encountered in the real-world exchange markets. First of all, transaction costs are omitted, which in the case a of short-term trading would pose significant costs in comparison to the potential return gained from a 3-minute investment. Secondly, the order book, i.e. limited market depth are ignored assuming that any amount of currency can be both bought *or* sold at a current market price given in the dataset.

Nonetheless, Figure 4.3 gives us a very important information regarding the nature of predictions. The predictions made by the neural network solely focus on the direction of movement. No information is given regarding the size of swing in the price. It could potentially be the case that the network does a very good job in detecting moderate changes to the price, but fails to correctly label big swings in the price. In that instance, even though the accuracy would be better than random chance, the network would perform poorly in the trading simulation. Results obtained deny such hypothesis and show that the prediction accuracy is not in any way correlated with the size of the price swing. In fact, the mean return of a 3-minute investment in ETH is 1 with a standard deviation of 0.00275.

Assuming 55% prediction accuracy of the model, we can estimate the ADRR to be:

$$(1.00275^{0.55} \times 0.99275^{0.45})^{1440} = 1.4478 \approx 44.8\% \tag{5.1}$$

Which is close to the empirical results obtained in this paper.

5.3 Model Improvements and Future Work

The work covered in this research paper can be extended in multiple dimensions.

Only a small subset of network architectures has been investigated mainly focusing on mixtures of LSTM and dense layers. Further work needs to be done in order to find optimal values of hyperparameters such as the number of layers, nodes per layer, dropout rates per layer etc. Further improvements for deeper architectures can possibly be achieved via decreasing the number of nodes per layer, increasing the training sample and training process extension above 10 epochs. In terms of structure - convolutional layers could potentially bring improvements to the accuracy, thanks to their ability to detect local features, like a persistent upward trend or high volatility. Including price data of other cryptocurrencies (or even stock market data!) could bring valuable information that would increase the performance. Furthermore, testing the model accuracy over different time horizons would indicate how far in advance can the neural network accurately predict.

However, for such model to be an applicable tool, changes to the methodology and data inputs are needed. The model shown in this paper solely gives information about what is about to happen in the market within t minutes. It does not assess the viability of a transaction and potential returns. In order for the model to assess whether a transaction can bring returns, instead of open, high, low, close price points, a dataset of the exchange order book is needed. Then, the neural network could make a prediction whether entering a long position at the *ask* price now, would give a positive return by selling at a *bid* price in the future. Transaction costs would also need to be incorporated in such a model. There is a wide range of possibilities of how to construct an output layer - both classification and regression models should be achievable in this context. Having built such model, it would be possible to allow it to trade on a real-world exchange and compare against different benchmark returns.

6 Conclusions

This research paper applies different Artificial Neural Network architectures on two-dimensional, BTC/USD and ETH/USD exchange rate data in an effort to predict directional market movements on a 3 minute time horizon.

The results indicate a possibility of classifying future market movements of ETH significantly better than random chance. The precision of the best performing model achieves satisfying benchmarks over the entire course of learning, however the recall tends to decrease for longer training horizons. Importantly, there is evidence that higher probabilities of predictions made by the network in fact corresponds with higher accuracy of predictions on the validation set. This suggests that the network has the ability to assess how volatile the market is at any given moment. The theoretical trading simulation presented in the paper suggests that there is no significant relationship between the accuracy of predictions and the magnitude of price change.

This paper shows promising capabilities of Artificial Neural Networks in predicting future market swings. Further work is needed to construct more applicable models that could be used in the market exchange environment. Overall, the results show promising signs of developing a profitable trading model that would make decisions based on the limit order book data.

References

Abu Bakar, Nashirah and Sofian Rosbi (Nov. 2017). 'Autoregressive Integrated Moving Average (ARIMA) Model for Forecasting Cryptocurrency Exchange Rate in High Volatility Environment: A New Insight of Bitcoin Transaction'. In: *International Journal of Advanced Engineering Research and Scinece* 4. DOI: 10.22161/ijaers.4.11.20.

Adebiyi, Ayodele, Aderemi Adewumi and Charles Ayo (Mar. 2014). 'Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction'. In: *Journal of Applied Mathematics* 2014, pp. 1–7. DOI: 10.1155/2014/614342.

Amberkar, Aditya et al. (Mar. 2018). 'Speech Recognition using Recurrent Neural Networks'. In: pp. 1–4. DOI: 10.1109/ICCTCT.2018.8551185.

Balaji, S. and K. Baskaran (Mar. 2013). 'Design and Development of Artificial Neural Networking (ANN) System Using Sigmoid Activation Function to Predict Annual Rice Production in Tamilnadu'. In: *International Journal of Computer Science, Engineering and Information Technology* 3. DOI: 10.5121/ijcseit.2013.3102.

Bhanja, Samit and Abhishek Das (Dec. 2018). 'Impact of Data Normalization on Deep Neural Network for Time Series Forecasting'. In:

Box, George and Gwilym Jenkins (May 1970). 'Time series analysis: forecasting and control (third ed'. In: *Time Series Analysis: Forecasting and Control: Fourth Edition.* DOI: 10.1002/9781118619193.

CoinMakretCap (2020). *CoinMakretCap.* URL: https://coinmarketcap.com/ (visited on 27/04/2020).

Conrad, Christian et al. (May 2018). 'Long-and Short-Term Cryptocurrency Volatility Components: A GARCH-MIDAS Analysis'. In: *SSRN Electronic Journal.* DOI: 10.2139/ssrn.3161264.

Duchi, John, Elad Hazan and Yoram Singer (July 2011). 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'. In: *Journal of Machine Learning Research* 12, pp. 2121–2159.

Fama, Eugene (Feb. 1970). 'Efficient Capital Markets: A Review of Theory and Empirical Work'. In: *Journal of Finance* 25, pp. 383–417. DOI: 10.2307/2325486.

Gemini (2020). *Gemini exchange.* URL: https://www.cryptodatadownload.com/data/gemini/ (visited on 27/04/2020).

Glorot, Xavier and Y. Bengio (Jan. 2010). 'Understanding the difficulty of training deep feedforward neural networks'. In: *Journal of Machine Learning Research - Proceedings Track* 9, pp. 249–256.

Graves, Alex (Aug. 2013). 'Generating Sequences With Recurrent Neural Networks'. In:

Graves, Alex, Abdel-rahman Mohamed and Geoffrey Hinton (Mar. 2013). 'Speech Recognition with Deep Recurrent Neural Networks'. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 38. DOI: 10.1109/ICASSP.2013.6638947.

Hinton, G. (2012). 'Neural networks for machine learning'. In: *Coursera, video courses.*

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). 'Long Short-term Memory'. In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Kingma, Diederik and Jimmy Ba (Dec. 2014). 'Adam: A Method for Stochastic Optimization'. In: *International Conference on Learning Representations.*

Lakonishok, Josef, Andrei Shleifer and Robert W. Vishny (1994). 'Contrarian Investment, Extrapolation, and Risk'. In: *Journal of Finance* 49.5. Reprinted in Harold M. Shefrin, ed., Behavioral Finance, Edward Elgar Publishing Company, 2001. Reprinted in Richard Thaler, ed., Advances in Behavioral Finance, Vol. II, Princeton University Press and Russell Sage Foundation, 2005., pp. 1541–1578.

Lee, C. et al. (Mar. 2001). 'Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping'. In: *Advances in Neural Information Processing Systems 13, NIPS 2000.*

Malkiel, Burton G. (Mar. 2003). 'The Efficient Market Hypothesis and Its Critics'. In: *Journal of Economic Perspectives* 17.1, pp. 59–82. DOI: 10.1257/089533003321164958. URL: https://www.aeaweb.org/articles?id=10.1257/089533003321164958.

McNally, Sean, Jason Roche and Simon Caton (Mar. 2018). 'Predicting the Price of Bitcoin Using Machine Learning'. In: pp. 339–343. DOI: 10.1109/PDP2018.2018.00060.

Meng, Qi et al. (Sept. 2017). 'Convergence Analysis of Distributed Stochastic Gradient Descent with Shuffling'. In: *Neurocomputing.* DOI: `10.1016/j.neucom.2019.01.037`.

Mishkin, Dmytro and Jiri Matas (May 2016). 'All you need is a good init'. In:

Munim, Ziaul, Mohammad Shakil and Ilan Alon (June 2019). 'Next-Day Bitcoin Price Forecast'. In: *Journal of Risk and Financial Management* 12. DOI: `10.3390/jrfm12020103`.

Nakamoto, Satoshi (Mar. 2009). 'Bitcoin: A Peer-to-Peer Electronic Cash System'. In: *Cryptography Mailing list at https://metzdowd.com.*

Nayak, Sarat, B. Misra and Dr. H. Behera (Feb. 2012). 'Index prediction with neuro-genetic hybrid network: A comparative analysis of performance'. In: *2012 International Conference on Computing, Communication and Applications, ICCCA 2012.* DOI: `10.1109/ICCCA.2012.6179215`.

Perlich, Claudia (Jan. 2011). 'Learning Curves in Machine Learning'. In: DOI: `10.1007/978-0-387-30164-8_452`.

Polyak, Boris (Dec. 1964). 'Some methods of speeding up the convergence of iteration methods'. In: *Ussr Computational Mathematics and Mathematical Physics* 4, pp. 1–17. DOI: `10.1016/0041-5553(64)90137-5`.

Radityo, Arief, Qorib Munajat and Indra Budi (Oct. 2017). 'Prediction of Bitcoin exchange rate to American dollar using artificial neural network methods'. In: pp. 433–438. DOI: `10.1109/ICACSIS.2017.8355070`.

Research and Markets (2019). *Global Cryptocurrency Market: Analysis By Type (Bitcoin, Ethereum, Ripple, Litecoin, Others), By Constituents (Exchanges, Mining, Wallet and Payment), By Region, By Country (2019 Edition): Opportunities and Forecast (2017-2024).* URL: `https://www.researchandmarkets.com/reports/4832708/global-cryptocurrency-market-analysis-by-type` (visited on 27/04/2020).

Rumelhart, David, Geoffrey Hinton and Ronald Williams (Jan. 1986). 'Learning Internal Representation by Error Propagation'. In: vol. Vol. 1.

Samuelson, Paul (Jan. 1965). 'Rational Theory of Warrant Pricing'. In: *Industrial Management Review* 6. DOI: `10.1007/978-3-319-22237-0_11`.

Schaul, Tom, Ioannis Antonoglou and David Silver (Dec. 2014). ‘Unit Tests for Stochastic Optimization’. In:

Srivastava, Nitish et al. (June 2014). ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’. In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Such, Felipe et al. (Dec. 2017). ‘Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning’. In:

Sutskever, I. et al. (Jan. 2013). ‘On the importance of initialization and momentum in deep learning’. In: *30th International Conference on Machine Learning, ICML 2013*, pp. 1139–1147.

Sutskever, Ilya, Oriol Vinyals and Quoc Le (Sept. 2014). ‘Sequence to Sequence Learning with Neural Networks’. In: *Advances in Neural Information Processing Systems* 4.

Tealab, Ahmed, Hesham Hefny and Amr Badr (June 2017). ‘Forecasting of non-linear time series using ANN’. In: *Future Computing and Informatics Journal* 2. DOI: 10.1016/j.fcij.2017.05.001.

Thaler, Richard and Werner Bondt (July 1985). ‘Does the Stock Market Over-React’. In: *Journal of Finance* 40, pp. 793–805. DOI: 10.1111/j.1540-6261.1985.tb05004.x.

Voigtlaender, Paul, Patrick Doetsch and Hermann Ney (Oct. 2016). ‘Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks’. In: pp. 228–233. DOI: 10.1109/ICFHR.2016.0052.

Wang, Shoujin et al. (July 2016). ‘Training deep neural networks on imbalanced data sets’. In: pp. 4368–4374. DOI: 10.1109/IJCNN.2016.7727770.

Xu, Bing et al. (May 2015). ‘Empirical Evaluation of Rectified Activations in Convolutional Network’. In:

Yiying, Wang and Zang Yeze (Mar. 2019). ‘Cryptocurrency Price Analysis with Artificial Intelligence’. In: pp. 97–101. DOI: 10.1109/INFOMAN.2019.8714700.