MANCHESTER
1824
The University of Manchester

# Deep Learning Algorithms Applied to Blockchain-Based Financial Time Series

Third Year Individual Project – Final Report

April 2018

**Thomas Hollis**

9563426

Supervisor: Dr Hujun Yin

School of Electrical and Electronic Engineering

## Abstract

This project aims to exploit recent advances in deep learning to investigate the predictability challenges posed by cryptocurrencies. These emerging blockchain-based financial time series have been shown to behave somewhat differently from traditional currencies. Indeed, few have built deep learning models for forecasting cryptocurrency prices and this subject is still in its infancy. As cryptocurrencies remain indubitably volatile, the forecasting methodology used in this report relied on the use of significant caution. Commonly used linear and non-linear models were first implemented before proceeding to neural network algorithms. Deep neural network investigations were carried out by building on these neural network implementations. This established a baseline performance to judge the success of deep neural networks in their attempt to outperform existing algorithmic alternatives.

The final deep neural network solution developed was tested thoroughly and achieved performances around 58% in theoretical, sandboxed and live trading trials. This performance was further improved through the consideration of confidence intervals during trading. Significant obstacles were overcome such as extensive computation times, difficulties in tuning model parameters, overfitting models, volatility, limited representative past data and others. Nevertheless, this investigation succeeded in highlighting two key points. Firstly, cryptocurrencies are indeed predictable, despite their large volatility which makes them challenging to address using conventional engineering models. Secondly, deep neural networks can significantly outperform shallower neural network competitors. The main implication of this is that further examinations into deep neural networks might one day help stabilise the fierce cryptocurrency market, promoting a global decentralised banking infrastructure for all countries.

## Contents

# 1. Introduction

## 1.1. Motivation

Blockchain-based financial time series (FTS) have become almost synonymous with uncertainty, fluctuation and volatility. This thesis aims to investigate cryptocurrency unpredictability, one of the many challenges surrounding this newest form of decentralised financial engineering. As this subject is still in its infancy, this Bachelor Thesis was granted as a bespoke project to provide an opportunity to engineer a solution to these novel issues (Appendix 1).

Cryptocurrencies are now maturing beyond their stigma of underground criminality from the dark web era. Indeed, non-profit projects such as the Level One Project from the Bill & Melinda Gates Foundation [1] and the United Nation's World Food Programme [2] now recognise the value behind blockchain finance. Both charities are attempting to use blockchains to help the two billion humans in underdeveloped countries who lack bank accounts and stable economic infrastructure.

However, cryptocurrencies' major issue of unpredictability is stifling efforts to bring blockchain finance to the mainstream as price fluctuation remains too high for many practical economic and monetary applications. Indeed, the cryptocurrency volatility index remains around 8% which is significant compared to the Foreign Exchange (Forex) market which experiences volatilities around 1%. This uncertain behaviour persists even as the cryptocurrency market cap currently exceeds $300Bn [3]. In this unstable environment, all too often compared to the subprime mortgage bubble of 2007 [4], algorithmic trading seems to be the brightest light at the end of the tunnel.

While human speculation fuels high trading volumes on exchanges, it cripples the stability of cryptocurrencies. Fortunately, algorithmic trading allows traders to remain less biased and less prone to human error by predicting prices based on predetermined metrics. This is not to say algorithms are infallible as shown by the failure of the Nobel Prize winning Black-Scholes equation of 1997 [5].

Nevertheless, deep learning algorithms are constantly praised for their abilities to bypass the curse of dimensionality, which provides a unique opportunity to investigate such an important economic challenge. Moreover, being able to forecast more accurately cryptocurrency exchange rates is not without significant financial value.

## 1.2. Aims and Objectives

The aim of this thesis is to develop a Deep Neural Network (DNN) capable of modelling cryptocurrencies to offer accurate real-time trading predictions. More specifically, the objectives to achieve this are as follows:

- Research and learn about theory behind time series analysis including learning the mathematics behind common existing models such as the Auto-Regressive Integrated Moving Average (ARIMA) model, the Generalised Auto-Regressive Conditional Heteroscedasticity (GARCH) model, the Self-Organising Map (SOM) and Neural Networks (NN).

- Collect and process relevant financial blockchain data via a Python script and learn how to implement and apply competitive trading algorithms to cryptocurrency data, using the R language and the TensorFlow library

- Exploit Granger Causality analysis to identify the best trading pair

- Engineer a DNN-based trading algorithm which achieves a forecast performance of at least 55% (current level of competitors) and investigate various DNN architectures to tune as best as possible the model developed to further improve forecasting accuracy

- Compare the performance of the final DNN solution developed to the other trading algorithms implemented and demonstrate that DNNs are a strong alternative for cryptocurrency trading algorithms by writing a script that uses the developed DNN model to undertake live trading (with real money on the Poloniex Exchange)

- Evaluate and compare the performance of DNN-based trading to other existing competitors and identify any shortcomings or issues

The scope of this thesis focusses on the theory and implementation in R of machine learning algorithms, specifically within the context of Blockchain-based FTS. Although some comparisons with Forex are occasionally made, the unique trading opportunities within cryptocurrencies are central to this thesis.

# 2. Mathematical Literature Review

## 2.1. Algorithmic Trading and Financial Time Series

FTS modelling and algorithmic trading is a practice that first revolutionised trading in the early 1970s. A notable example can be seen in the New York Stock Exchange (NYSE) when they introduced the Super Designated Order Turnaround (Super DOT) [6]. This automated system was a key turning point as it allowed trades to become increasingly digitised which subsequently opened the door to practices such as High Frequency Trading (HFT) and black box trading. These increased in popularity, especially following the Black Wednesday GBP crash of 1992 [7].

Amongst this trend in digitisation, the analysis of FTS was divided into two categories: fundamental analysis and technical analysis [8]. Fundamental analysis is the study of a currency's price based on macroeconomic factors. These include the overall state of the economy, interest rates, GDP and others. On the other hand, technical analysis, as defined by J. Murphy in [8], is the study of market action to forecast future trends. This is achieved through the analysis of shorter term financial chart data, primarily price and volume. Both fundamental and technical analysis are critiqued by the Efficient Market Hypothesis (EMH). The EMH, highly disputed since its initial publication in 1970, hypothesises that currency prices are ultimately unpredictable [9].

Nonetheless, technical FTS analysis is fundamentally continuous signal analysis at its core [10]. Continuous signals are commonly classified as either deterministic or stochastic i.e. random. By treating signals as stochastic processes, we can approximate the behaviour of FTS by passing additive gaussian white noise (AWGN) through a linear time invariant (LTI) filter.

The linear models and non-linear models, mentioned hereafter, together form the two main sub-categorises of existing solutions in FTS analysis.

## 2.2. Linear Models

The most rudimentary approach to modelling FTS is by assuming that they follow the random walk model. The random walk model can be simply expressed as a sum of a series of independent random variables. Mathematically [11]:

$$Z = \sum_{i=0}^{n} X_i \qquad (1)$$

*where: $X_i$ is the random variable, $n$ is the number of variables*

The model shown in equation (1) can in fact be viewed as a special case of a more general model. Indeed, the random walk model is equivalent to the linear Auto-Regressive (AR) model with a slope of 1. This model, denoted $AR(p)$, can be expressed mathematically as [11]:

$$\hat{x}[t] = \sum_{i=1}^{p} (\alpha_i \, x_{t-i}) + \epsilon_t \qquad (2)$$

*where: $\hat{x}[t]$ is the predicted value of the variable $x$ in terms of time, $p$ is the model order parameter, $\alpha$ is the model coefficient, $x_{t-i}$ is the value of variable $x$ at time $t - i$, $\epsilon$ is AWGN.*

Equation (2) describes how the predicted value is derived from a regression using coefficients applied to past values summed with AWGN. Hence, (2) shows how the response variable of a previous time period becomes a predictor for the next.

A set of equations were developed by U. Yule and G. Walker in [12] to provide quantitative methods for estimating parameters in AR models. This work was subsequently expanded upon by J. P. Burg in [13] who provided an alternative approach albeit with different stability properties.

These AR models are often accompanied by another type of linear model, the Moving Average (MA) model. This model, denoted $MA(q)$, can be written as [11]:

$$\hat{x}[t] = \mu + \sum_{i=1}^{q} (\beta_i \, \epsilon_{t-i}) + \epsilon_t \qquad (3)$$

*where: $\mu$ is the mean, $q$ is the model order parameter and $\beta$ is the model coefficient.*

In (3) as in (2) the predicted value of $x$ is being modelled from past data by effectively using an impulse response filter. However, in (3) unlike in (2), the mean (often assumed to be 0) and coefficients applied to past noise terms are used instead of using past values directly.

These two linear models can be combined into the Autoregressive Moving Average

(ARMA) model. This new model, denoted $ARMA(p,q)$, can be written as [11]:

$$\hat{x}[t] = \sum_{i=1}^{p}(\alpha_i\, x_{t-i}) + \sum_{i=1}^{q}(\beta_i\, \epsilon_{t-i}) + \epsilon_t \tag{4}$$

Equation (4) can be easily derived from the linear sum of (2) and (3). By combining (2) and (3) the ARMA model is able to use both autoregression and moving averages to provide its predicted value.

However, a fundamental limitation of AR, MA and ARMA models is that they all assume the process being modelled is stationary. Stationarity is a property of processes whereby the probability distribution remains constant over time, thus variance also remains constant. Indeed, this assumption is significant as FTS are often non-stationary processes. This can be seen by observing that variance changes from one period to the next, as will be demonstrated in detail in later sections. Therefore, this model's accuracy will suffer, highlighting the need to take this problem of stationarity into consideration.

This is done by generalising the ARMA model into the Autoregressive Integrated Moving Average (ARIMA) model [11]. The ARIMA model solves the issue of non-stationarity by exploiting the concept of returns (or degrees of differencing). Non-stationary series can therefore be made stationary by differencing. Returns ($r$) and consequently the $ARIMA(p,d,q)$ model are given by [11]:

$$r = x_t - x_{t-1} \tag{5}$$

$$\therefore\ \hat{x}[t] = \mu + \sum_{i=0}^{p}(\alpha_p x_{t-p}) - \sum_{i=0}^{q}\left(\beta_q \epsilon_{t-q}\right) \tag{6}$$

*where: $\beta$ is defined to have negative signs, following Box and Jenkins convention*

As shown in (6), $ARIMA(p,d,q)$ differs from the $ARMA(p,q)$ model through its use of an added derivative parameter $d$. This parameter is used to identify how many times the data was been differenced, as shown in (5).

It is worth noting that stationary ARMA processes are, by construction, ergodic. Ergodicity is a property of processes whereby given sufficient time, samples are statistically representative of the whole.

## 2.3. Non-Linear Models

The aforementioned linear models all suffer from the assumption that FTS are homoscedastic processes. This means that these linear models assume that the variance in FTS remains constant over time.

This is indeed often a poor assumption to make, as shown in [14] by R.F. Engle. In [14] Engle states that by using a more sophisticated model such as the Auto-Regressive Conditional Heteroscedasticity (ARCH) model, the homoscedastic assumption can be avoided. This $ARCH(p)$ model describes the variance of a time series as follows [14]:

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \, \epsilon_{t-i}^2 \qquad (7)$$

*where: $\alpha$ is the ARCH model parameter*

Equation (7) shows that ARCH originated from Engle's realisation that there is a high correlation in squared returns. Indeed, here the ARCH model is expressed as a linear function of past squared disturbances. For this reason, ARCH is often likened to a finite impulse response filter (FIR).

This ARCH model was later described by Bollerslev in [15] as a special case of a more generalised model called the Generalised Auto-Regressive Conditional Heteroscedasticity (GARCH) model. This $GARCH(p,q)$ model describes variance as follows [15]:

$$\sigma_t^2 = \omega + \sum_{i=1}^{q}(\alpha_i \, \epsilon_{t-i}^2) \; + \sum_{i=1}^{p}(\beta_i \, \sigma_{t-i}^2) \qquad (8)$$

*where: $\alpha$ and $\beta$ are the GARCH model parameters*

By comparing equations (4) and (8), it can be noted that GARCH is using the ARMA model to express the error variance of the time series. For this reason, GARCH is often liked to an infinite response filter (IIR).

Many more variants of the GARCH model have been published since its original publication in 1986. These include NAGARCH (nonlinear asymmetric GARCH) [16], EGARCH (exponential GARCH) [17], GJR-GARCH (Glosten-Jagannathan-Runkle GARCH) [18] and many others. These GARCH derivatives are often nested under Hentschel's fGARCH (Family GARCH) model [19] but these all lie outside the scope of this report.

In the same time as the ARCH and GARCH models, J. Leontaris and S. A. Billings published an alternative in [20] known as is the Nonlinear Autoregressive Moving Average model with exogenous inputs (NARMAX). This work, building on their own previous work on ARMAX models, demonstrated that NARMAX models can successfully be applied to model complex time series. This model can be expressed mathematically as [20]:

$$y(k) = F\left[y(k-1), \dots y\left(k-n_y\right), u(k-1), \dots u(k-n_u), e(k-1), \dots e(k-n_e)\right] + e_k \quad (9)$$

*where: $y(t)$ is the system output prediction and $u(t)$ is the system input, $F[\cdot]$ is some non-linear function, $e$ is noise and $n_{y/u/e}$ is the maximum lag for system output/input/noise*

From (9) we can see that NARMAX is, at its core, a function of lags of system input, output and noise (modelled explicitly). While equation (9) does not contain all the detail of NARMAX models, it is about as far as one can go in terms of specifying a general finite non-linear system [20]. NARMAX models are often implemented in varying types of networks, but further mathematical detail lies beyond the scope of this report.

## 2.4. Neural Network Models

A large variety of linear and non-linear solutions exist for FTS modelling, as demonstrated in sections 2.2 and 2.3. However, one competitive alternative is gaining significant popularity both within this field as well as many others. This alternative is artificial neural networks (NN). These models are inspired by the biological neural networks in humans which are the brain [21]. In brains, neurons are connected via synapses to each other to form an interconnected network. Conversely in artificial NNs, neurons are called nodes and connecting synapses are called paths.

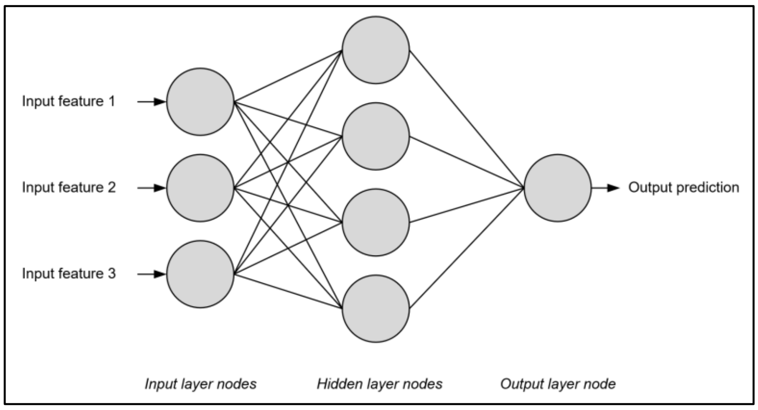


*Figure 2.1 - Neural Network model*

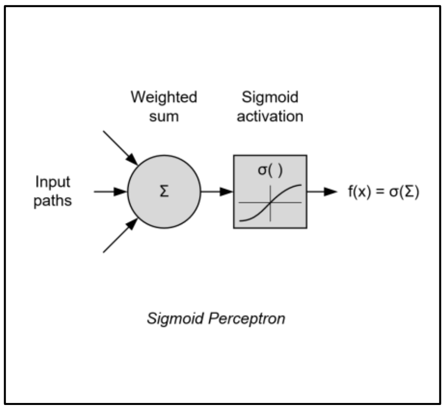


*Figure 2.2 - Perceptron*

As shown in the nodes of figures 2.1 and 2.2, the fundamental building blocks of NNs are sigmoid perceptrons [22]. The mathematical operation of a single sigmoid perceptron can be described by the following linear function [22]:

$$f(x) = \begin{cases} 1 & \text{if } \sigma(\Sigma_{i=1}^{m} w_i x_i + b) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

*where: $f(x)$ is the perceptron output function, $\sigma()$ is the sigmoid activation function, $x_i$ are the inputs to the perceptron, $w_i$ are the weights of each input and $b$ is the bias.*

The perceptrons shown in (10) are binary classifiers that are first trained on a set of training data (supervised learning). These perceptrons can then guess, randomly at first, what the correct classification is. Depending on the result, the weight and bias of each perceptron is subsequently adjusted, and the process is repeated for the entire training dataset. Once the training phase is over, the perceptrons should be suitably tuned for predicting unseen data (also known as test data).

By combining multiple perceptrons into layers, it is possible to construct a simple Multi-Layer Perceptron (MLP) NN model for FTS forecasting. MLPs are a basic class of neural networks which are constructed with at least three layers: the input layer, the hidden layer and the output layer. The inputs nodes of the MLP model are a series of past prices (at times $t-1, t-2, t-3$ ...) and the output node of the MLP is the prediction of the next price. This single hidden layer MLP can make decisions by adjusting its perceptron weights and biases according to a cost function, often using the squared difference error. The output represents the best guess of the network and is henceforth taken as the prediction for the next price. This is an application of multivariate calculus and more specifically an application of using gradient descent to minimise a cost function. This is best visualised by imagining each hidden node creating a linear separation boundary allowing the output node to be a combination of linear separations. Effectively this creates a piecewise separation boundary which is how a network of linear perceptrons can model non-linear processes so effectively.

This separation boundary visualisation is best observed in Self-Organizing-Map (SOM) models. SOMs are a type of NN model which are trained using unsupervised learning rather than supervised learning. Unsupervised learning is distinct from supervised learning as no “right” answer is provided during the training phase. In this case the data is said to be “unlabelled”. Hence in SOMs, neurons update via [24]:

$$W_v\,(s+1) = W_v\,(s) + \alpha(s) \cdot \theta(u, v, s) \cdot \big(D(t) - W_v(s)\big) \tag{11}$$

*where: $W$ is the weight vector, $\theta$ is the neighbourhood function, $\alpha$ is the learning rate, $s$ is steps, v is neuron number, t is training index and $D$ is the vector of input neurons*

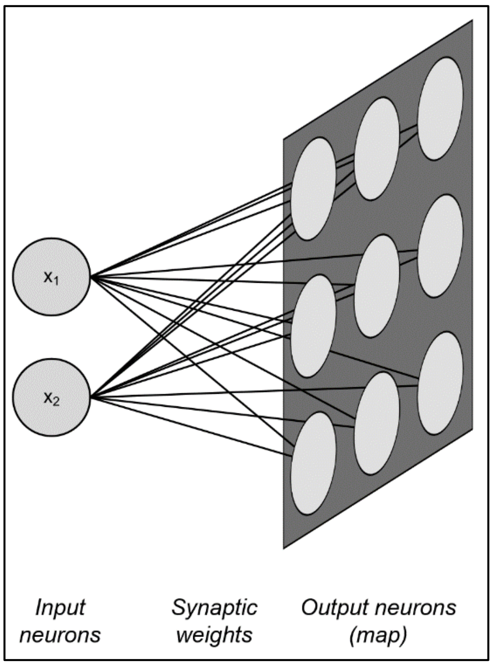


*Figure 2.3 - SOM model*

As shown in equation (11), the updating of neurons is based on the neuron of the previous step summed with a product of the neighbourhood function, learning rate and difference of inputs with neurons of previous steps. Equation (11) is therefore a form of unsupervised clustering. The neuronal basis of this unsupervised learning is known as Hebbian learning. This SOM model can be adapted for FTS forecasting (see section 3). Figure 2.3 aside shows a topological representation of the SOM model. Indeed, input neurons connect to the output map via weighed synaptic paths.

## 2.5. Deep Learning Models

Despite all the success of NN models, this is not without shortcomings. In [25], Bishop noticed that when applying neural networks to complex tasks, such as image recognition, a significant flaw occurred known as the curse of dimensionality. This effect, also known in ML as the Hughes Phenomenon, arises due to an increasingly large feature set (i.e. resolution for images) which cannot be accounted for [26].

To remedy the curse of dimensionality, Deep Neural Networks (DNN) were henceforth developed by allowing each hidden layer to take as inputs the output of the previous layer. LeCun pioneered much of the work done in applying backpropagation to such networks in [27] and many other publications. Indeed, by exploiting partial connections rather than using fully connected networks, this allows for much deeper DNN experimentation (up to 100s of hidden layers). These DNNs allowed significantly improved performances in a wide range of applications including language processing, computer vision and bioinformatics [27]. Mathematically:

$$E = \frac{1}{2n} \sum_x \left| |y(x) - y'(x)| \right|^2 \qquad (12)$$

*where: $E$ is the error function, $n$ is the training example number, $y(x)$ is the output*

Equation (12) shows that backpropagation works by calculating the error function using the square of the Euclidian distance between predicted and actual outputs.

However, using this method for gradient descent optimisation is seldom implemented in DNNs due to its high overhead. Instead, less intensive alternatives such as simulated annealing or stochastic gradient descent are used to approximate this process through an iterative numerical method. In 2014, Kingma and Ba described in [28] one of the most effective implementations of stochastic gradient descent: the Adaptive Moment Estimation (ADAM) optimisation algorithm. This algorithm exploits the concept of momentum and can be described as follows [28]:

$$\Delta w := \alpha \Delta w - \eta \nabla Q_i(w) \tag{13}$$

$$w := w + \Delta w \tag{14}$$

$$\therefore w := w + \alpha \Delta w - \eta \nabla Q_i(w) \tag{15}$$

*where: $\eta$ is learning rate, $Q$ is the cost function, $w$ is the parameter that minimises $Q$*

Equation (15) is reached by substituting (13) into (14). Equations (13) - (15) show how an estimate for the value of $w$ that minimises the cost function can be iterated. Indeed, $w$ is estimated from its past values, hence is said to have "momentum".

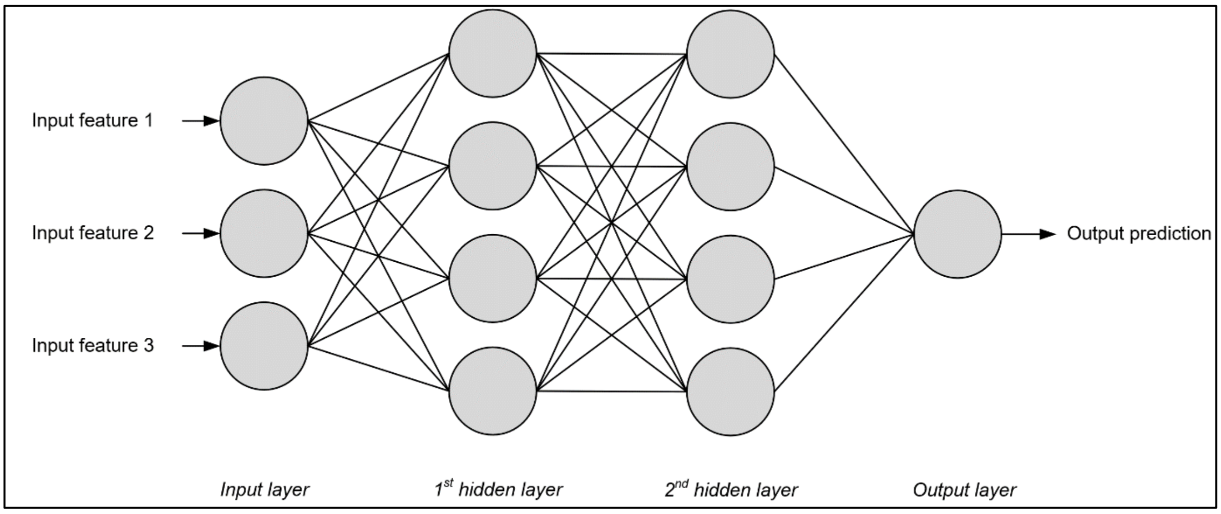


*Figure 2.4 - DNN Model (3-4-4-1 architecture)*

Figure 2.4 shows one such DNN model with two hidden layers. The two key challenges of DNNs are overfitting and extensive computation time. Overfitting arises when DNNs are trained to fit the training data too closely and therefore fail to generalise on testing data, leading to poor performances. Similarly, extensive computation times arise because DNNs are have far more nodes and paths, thus far more weights and biases than simpler NN alternatives. Since, each of these need to be computed, the training period of DNNs is much longer than other alternatives. These challenges were encountered and tackled in sections 3 and 4.

## 2.6. Blockchain-based Finance

The origin of blockchain-based finance can be traced back to an anonymous user under the pseudonym "Satoshi Nakamoto". This alias was listed as the author of a pioneering white paper [29] published in 2008, describing peer-to-peer payments through a blockchain infrastructure. This payment method went on to become Bitcoin (BTC), the first and largest cryptocurrency from which branched hundreds more, including Digital Cash (DASH) [30], the cryptocurrency of focus of this thesis. Blockchain technology is founded on the principle of collecting all transactional data into blocks of a public ledger. These are hashed and attached to previous blocks using a Merkle root, thus forming an immutable chain. This is illustrated as follows:

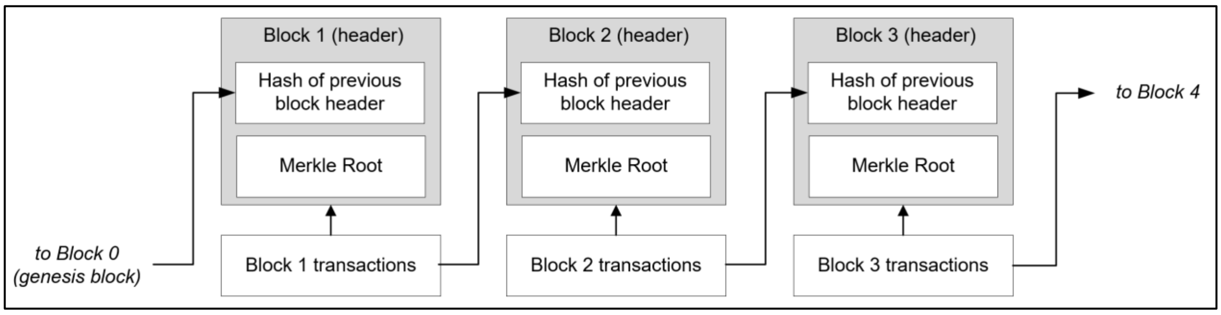


*Figure 2.5 - Simplified blockchain model*

Users of such cryptocurrencies broadcast their transactions which get approved by neighbouring nodes that compete to verify their integrity. This competitive verification process is known as mining and rewards the miners with their share of the transaction fee (paid by the user). It is for this reason that finance on blockchains is known as a decentralised process and has remarkable self-management capabilities. This decentralisation is made possible by the competitive verification of peer-to-peer transactions. The cryptographic process underpinning this technology is known as Proof-of-Work (PoW) [31]. The PoW algorithm dictates how much reward each node obtains, as a function of the node's contributed hashing power. Mathematically [31]:

$$P(B=1)=\frac{R^2}{H}\left(\frac{H-1}{H}\right)^{R-1} \tag{16}$$

*where: $P(B=1)$ is the probability of a node of hashing 1 block, $R$ is the hash-rate of the node and $H$ is the network hash rate (competitors)*

Equation (16) shows that the larger the hashing power of the user, the higher the probability of being the node who finds the correct hash, confirms the transaction and hence obtains the reward. Blocks found per minute is indeed binomially distributed. An alternative cryptographic approach to achieve a distributed consensus is Proof-of-Stake (PoS) [32]. The PoS algorithm is fundamentally reliant on the leader selection process, as expressed below [32]:

$$p_i=\frac{s_i}{\sum_{k=1}^{n} s_k} \tag{17}$$

*where: $p_i$ is the probability of leadership of node $i$, $s_i$ is the stake held by stakeholder $U_i$ and $s_k$ is the stake held by the other stakeholders*

Equation (17) simply shows that the larger the stake (i.e. the more cryptocurrency is owned by a node) the larger the probability that node successfully confirms5 the next block. This is equivalent to having a larger hashing power in PoW, as in (16).

DASH uses a mix of both PoW and PoS for its decentralized operation and is known as a PoW/PoS hybrid [30]. DASH was originally created in 2014 under the name Darkcoin, as a faster and privacy-focussed alternative to Bitcoin [30]. The reason behind choosing DASH for this thesis' final algorithm is discussed in section 3.6.

A major advantage of cryptocurrencies is that blockchains contain a huge amount of publicly downloadable data for use in trading algorithms. In addition, unlike traditional exchange markets with open and close times, cryptocurrency exchanges do not close. This means the data collected has the advantage of being truly continuous. One of the largest of these exchanges by trading volume is Poloniex [33]. Large trading volumes are preferable for FTS modelling as they allow constantly changing prices even at very high frequency which enables the deployment of high frequency algorithms. This significant advantage, combined with their excellent API capabilities, constituted the main motivation behind choosing Poloniex for this thesis.

Despite what misleading websites such as NeuroBot claim, the application of deep learning algorithms for cryptocurrency forecasting is thus far very limited. Indeed, there is currently no direct competitor which has attempted prediction of DASH using deep learning and most other competitors focus on BTC forecasting. Amongst these BTC competitors, most attempts are not published in academic journals [34] - [36], with only two competitors found to be peer-reviewed and published in academia [37], [38]. Of the unpublished competitors cited, [34] achieves the best performance which ranges from 50% to 55%. All three attempts conclude that their findings warrant further investigation into applications of deep learning to blockchain FTS. Amongst the published competitors, Greaves and Au in [37] use a 2-hidden-layer DNN and achieve a 55% prediction accuracy without heavy tuning. Similarly, Jang and Lee in [38] use a Bayesian Neural Network (BNN) and achieve accuracies between 55% to 60%. Both [37] and [38] conclude that significant potential was uncovered and further research into DNN modelling for cryptocurrencies is required.

These encouraging conclusions from BTC competitors is mirrored by research into Deep Learning within classical Forex markets [39], [40] as well as on the CATS benchmarking FTS [41]. Indeed, this suggests that a prediction performance above 55% is theoretically achievable by leveraging deep learning against blockchain FTS.

# 3. Theoretical and Practical Development

## 3.1. Project Organisation

This thesis is founded on a strong organisational structure to optimise work output. All 24 weeks of work were rigorously planned and organised as shown in the project's final Gantt Chart (Appendix 2). In addition, all documentation was done via a GitHub repository [42] which tracks code commits, progress and issues (Appendix 3). Each significant piece of work undertaken is also organised and classified in a Work Log (Appendix 1). An initial progress report was also written and submitted for review along with a technical, health and safety assessments to help identify any potential shortcomings of the project (Appendix 1).

## 3.2. Data Processing

One of the most important elements of any deep learning algorithm is the quality of its input data. In addition, a major advantage of blockchain FTS is the public nature of blockchain infrastructures. Indeed, all cryptocurrency transactional data is publicly downloadable, searchable and analysable (as shown in section 2.6).

For this reason, a heavy emphasis was placed on gathering large amounts of high quality data for use throughout the project lifecycle. Various data processing techniques were used such as API calls, HTTP GET requests, web scraping and data cleansing.

The first type of data gathered was the historical close price of past data on an hourly scale. This was useful for examining and comparing linear and non-linear models (ARIMA, GARCH, SOM & NARMAX) as applied to Forex and cryptocurrency time series. This data was downloaded directly from FTP servers of various platforms such as HistData [43] and CoinMarketCap [3]. Less pre-processing effort was required for this stage as the data was mostly complete and directly usable by financial models. This data, spread across 22 files, was eventually combined into a large .csv dataset of 1.6GiB.

The next type of data gathered was historical live data from the Poloniex cryptocurrency exchange at a high frequency timescale. This data is only accessible live and was used to create the training sandbox used by the machine learning algorithms in their training and backtesting phases. The only way to collect this data is by live API queries repeated non-stop for extended periods of time. In order to gather such a large volume of data, a python script was written (see Appendix 1), in collaboration with N. Frisiani [44], to automatically query the Poloniex API at each time interval where new data was available. This script gathered 3200 features that could potentially have an impact on trading price into a .csv file. These features include Open, Close, Low, High (OCLH) data, volume data, order books, tickers, loan orders and others. The script was run for various periods of time. The longest of which was four months consecutively and the shortest was two weeks. This was done using cloud computing, here a Google Compute Engine (GCE) [45]. The GCE was running Debian Linux (Jessie) on an octo-core Intel Haswell processor with 8GiB of RAM and 100GiB of SSD storage. This was more than sufficient for the 2GiB of data collected in total. This data required significant cleansing as the Poloniex API had periods of downtime, errors and dirty API responses.

Further data was gathered and analysed in real time during live trading trials (see section 4.2). This was done via the use of an R wrapper to the Poloniex API and was focussed on OCLH features only. This allowed for quasi-instantaneous dataset updates which enabled the final algorithm to run and make trade predictions at frequencies of up to 30min.

The final type of data collected was some extended non-financial data. This includes full cryptocurrency blockchains and metadata, GitHub commits to core blockchain code and others. This data was collected to analyse the potential predictability of some less obvious non-financial features. This was not used in the final algorithm but constitutes an encouraging baseline for future investigations (discussed further in section 5.2).

## 3.3. AR, ARMA, ARIMA Models

### 3.3.1. Implementation

Before delving into the implementation of deep neural networks, it is important to first familiarise ourselves with more rudimentary financial time series models. In this section, the linear AR, ARMA and ARIMA models are implemented in an R script (see source code in Appendix 1). This script runs on 400 hours (or roughly 16 days) of historical BTC data and produces a graph showing the original training data and the model fitting to the data. At the graph tails, AR, ARMA and ARIMA coefficients for different model orders were used to plot a model function. While interesting, this extra function is only showing the different model orders used, not the performance.

### 3.3.2. Performance and Analysis

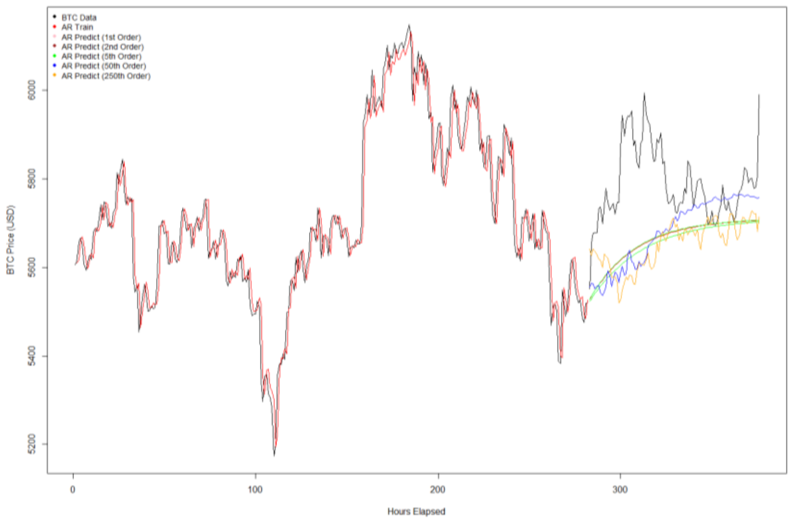


*Figure 3.1 - AR Yule-Walker models (BTC data)*

Figure 3.1 aside shows the performance of the AR Yule-Walker models (red) applied to the BTC time series (black). As expected from the theory discussed in section 2, a time-delayed fit can be observed. The error seems low but the average direction prediction performance of these models over this period is roughly 49%, which is weak but acceptable.

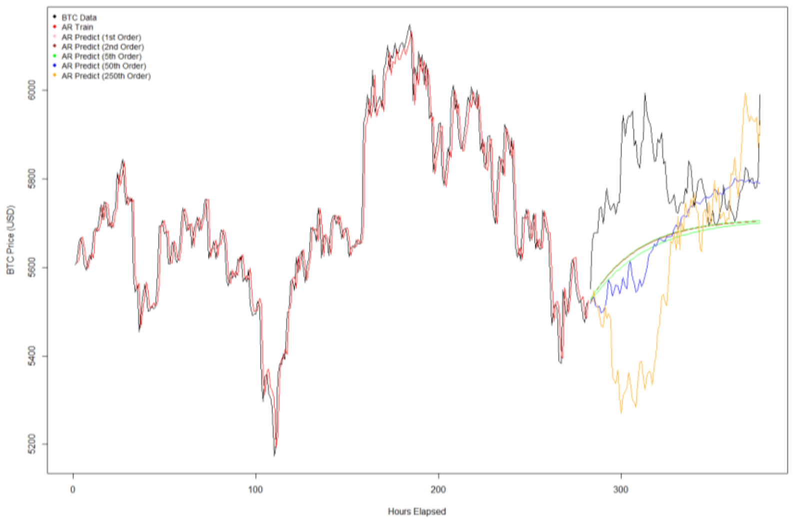


*Figure 3.2 - AR Burg models (BTC data)*

Figure 3.2 aside shows that when applying the AR Burg models to the same time series, a similar fit is observed with again an average direction performance of roughly 49%.

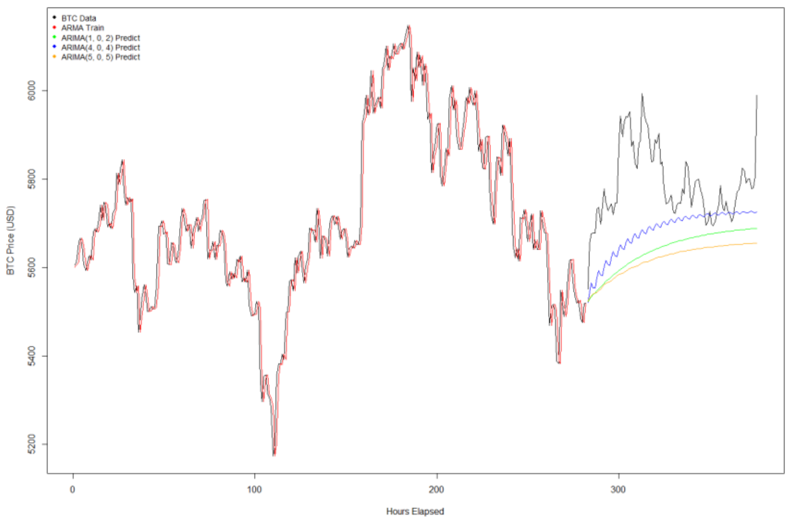


*Figure 3.3 - ARMA models (BTC data)*

Figure 3.3 aside shows that the ARMA models are slightly closer fits to the original BTC data. Indeed, a smaller NMSE arises but the performance remains modest, around 50%.

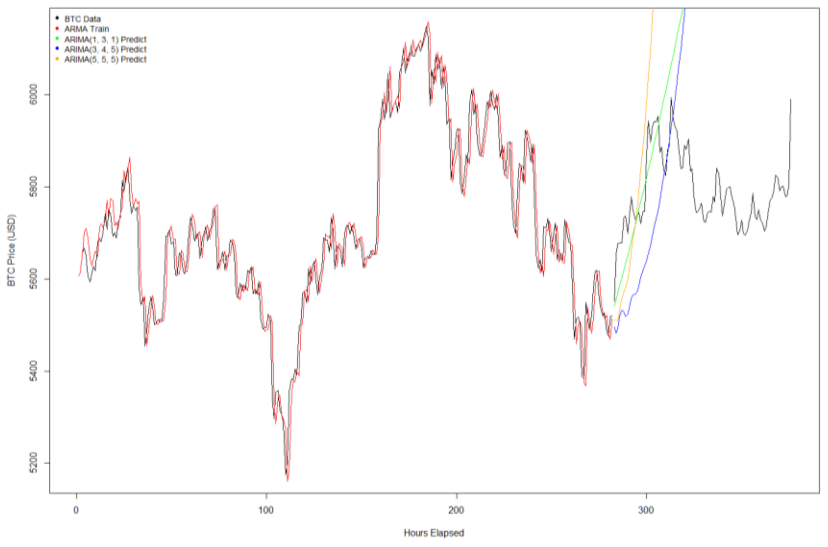


*Figure 3.4 - ARIMA models (BTC data)*

Figure 3.4 aside shows that the ARIMA models seem to be a similar fit and NMSE than the previous ARMA models. However, this may be an artefact of using this specific BTC timeframe. Here, an improved performance of 52% is nonetheless observed.

From the figures 3.1 to 3.4, the basic linear models implemented seem to perform as expected from equations (2) to (6), with performances ranging from 49% to 52%. Predictability rates of anything from 45% to 55% are indeed common amongst linear models, as observed in the literature review.

As for fitting metrics, the NMSE for these models ranges from 0.0077 to 0.0093. This is also in line with theoretical expectations. The AR Yule-Walker NMSE was 0.0093. This decreased to 0.0077 for AR Burg. On the other hand, the NMSE was 0.0085 and 0.0081 for ARMA and ARIMA respectively. When comparing to the NMSE of the same models but applied to Forex currencies, BTC indeed has higher NMSEs for all models. This could be due to the volatile nature of cryptocurrencies. However, further discussion of these linear models and their results can be found in the Proposal Report in appendix 1 as this is not the focus of this report.

While some of these linear models seem to have somewhat underperformed in this particular BTC timeframe compared to the expected values, this discrepancy is expected as performance fluctuations are common when using only 400 hours of data points.

Using the above implementations, more representative NMSE and performance accuracy values were collected for the DASH cryptocurrency under larger timeframes. This will later be compared and discussed in the summary table in section 4.1.

## 3.4. ARCH, GARCH Models

### 3.4.1. Implementation

Having observed the performance of the basic linear models, the non-linear ARCH and GARCH models were next to be implemented. This was implemented in an R script using the “fGarch” package [46] (see source code in Appendix 1) and was run, as before, on 400 hours of historical BTC data.

### 3.4.2. Performance and Analysis

A wide range of different ARIMA parameters were used to attempt to model BTC using ARCH or GARCH to model the variance. While the task of jointly determining ARIMA-GARCH orders is challenging, experimenting with a pool of candidate models is commonly used to differentiate between performant and under-performant models.

The most successful and notable results of these ARIMA-GARCH models are summarised in the table hereafter.

| Model | | Prediction Accuracy | NMSE |
|---|---|---|---|
| ARIMA - GARCH | **ARIMA(1,1,0) GARCH(1,1)** | 0.513 | 0.03875 |
| | **ARIMA(1,1,1) GARCH(1,1)** | 0.525 | 0.01729 |
| | **ARIMA(0,1,3) GARCH(1,1)** | 0.515 | 0.10155 |
| | **ARIMA(4,1,1) GARCH(1,1)** | 0.520 | 0.04832 |
| | **ARIMA(3,1,5) GARCH(1,1)** | 0.538 | 0.06251 |
| | **ARIMA(5,1,0) GARCH(1,1)** | 0.528 | 0.06357 |
| | **ARIMA(4,1,4) GARCH(1,1)** | 0.533 | 0.07194 |
| | **ARIMA(1,2,2) GARCH(1,1)** | 0.528 | 0.08061 |
| | **ARIMA(0,2,1) GARCH(1,1)** | 0.518 | 0.03904 |
| | **ARIMA(0,2,5) GARCH(1,1)** | 0.535 | 0.10173 |

*Table 3.1 - ARCH and GARCH model performances (BTC data)*

Table 3.1 shows a range of different ARIMA-GARCH candidate models for predicting BTC. Indeed, the first group of models, using degree of differencing $d = 1$, are at the top of the table while those using a degree of differencing $d = 2$ are at the bottom of the table.

The performances of all these models ranges from 51.3% to 53.8%. This is in line with theoretical expectations of performances of 45%-60%. The best model, has parameters ARIMA(3,1,5) GARCH(1,1) and yields a prediction rate of 53.8%. It is nonetheless worth treating this value with caution as 400 hours is not necessarily representative of all time periods of the BTC time series. On the other hand, the NMSE values range from 0.01729 to 0.10173. Hence, the model that seems to fit the curve the best with an NMSE of 0.01729 was the ARIMA(1,1,1) GARCH(1,1).

Further discussion of these ARCH and GARCH models and their results can be found in appendix 1 as this is not the focus of this thesis. In addition, these ARIMA-GARCH implementations were used to obtain more representative NMSE and performance accuracy values during the final model comparison for DASH. This was done over larger timeframes and will later be discussed in section 4.

## 3.5. SOM, NARMAX Models

### 3.5.1. Implementation

Following the ARCH and GARCH models, SOM and NARMAX models were the next models to examine. The SOM algorithm was first written from scratch and used on a practice exercise with animal data (source code in Appendix 4). This SOM algorithm was then adapted and used to model an extract (400 hours) of the BTC FTS (source code in Appendix 4).

This SOM model was then compared to an implementation of the NARMAX model in R using the “tsDyn” package [47]. Again, this model was run on approximately 400 hours of normalised BTC data (source code in Appendix 4).

### 3.5.2. Performance and Analysis

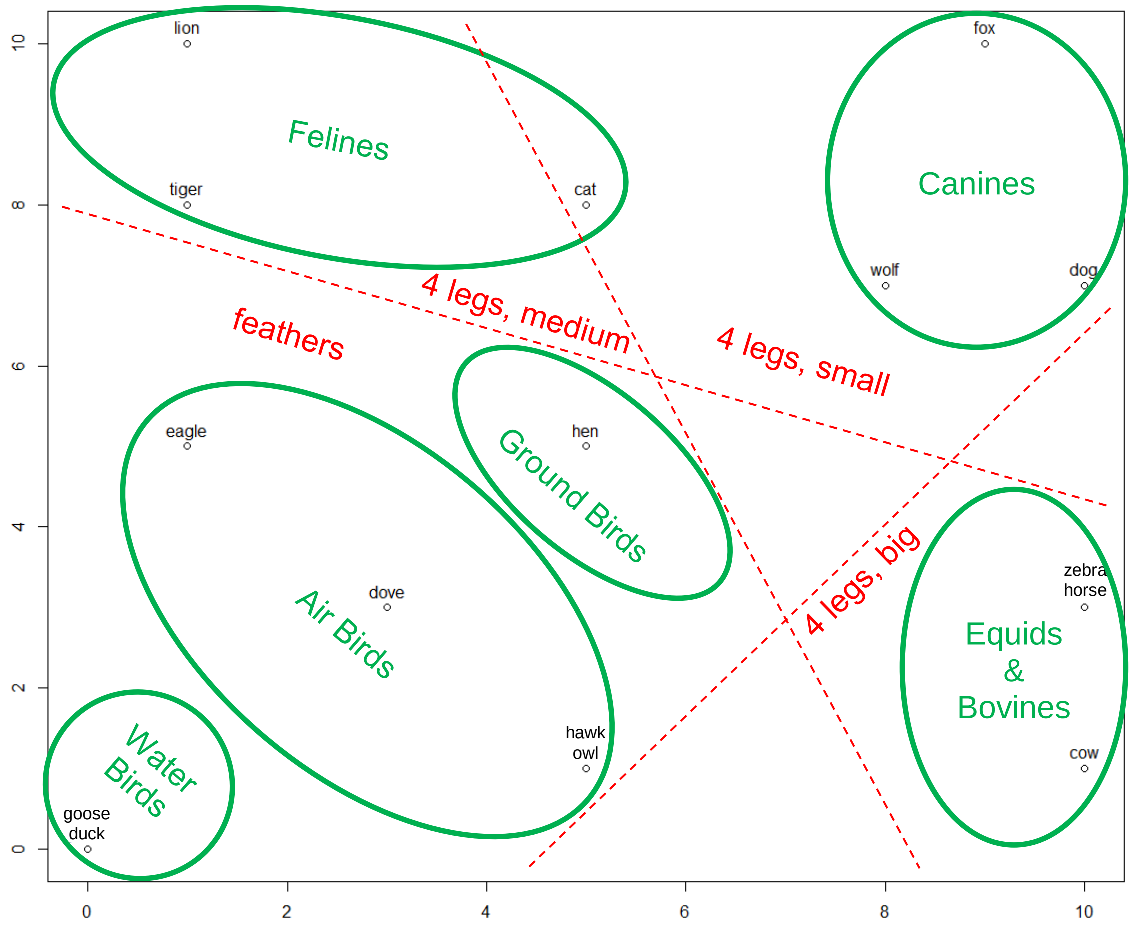


*Figure 3.5 - Output map of SOM applied to animal data (with human analysis)*

Figure 3.5 shows the SOM model successfully clustering groups of animals together, only using their physical traits. These physical traits (main ones highlighted in red) are the SOM's only input data from which it clusters the animals. This is achieved by creating separation boundaries based on this physical input data, as discussed in section 2. It is worth noting that while this clustering changes upon each iteration, similar animals of the same family (shown in green) are usually clustered together on all iterations. This exercise highlights the potential of SOMs to reach classification conclusions from raw input data.

The same SOM clustering principle can be applied to FTS. In this case, the time series are split into vectors of desired length. These vectors are then clustered using the SOM model as seen previously, which results in a map of various gradients grouped by similarity.

One such map clustered by similarity can be obtained by implementing the SOM model and applying it to the BTC time series as shown on the following page.

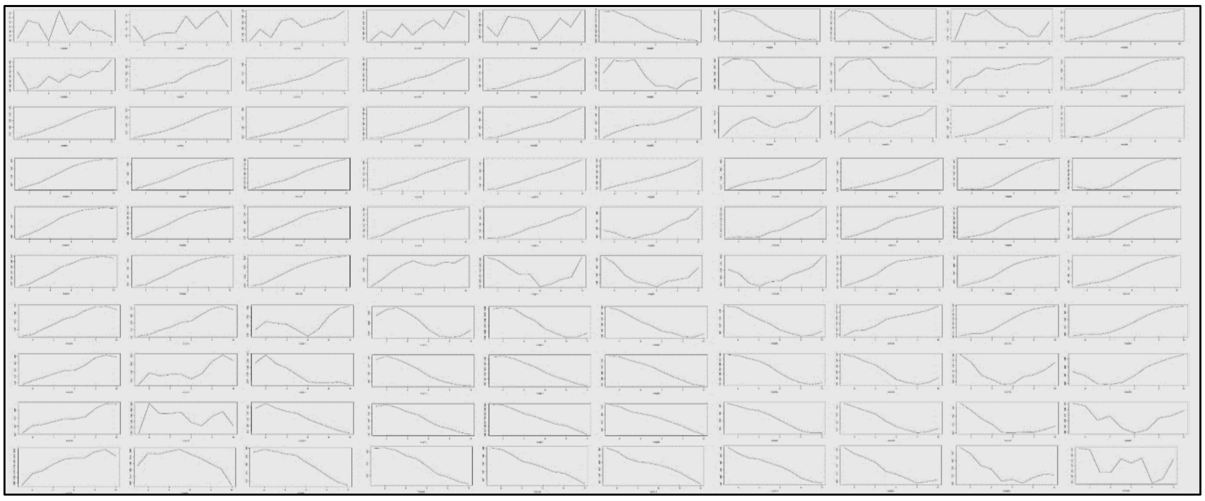


*Figure 3.6 - Output of SOM (applied to BTC data)*

Figure 3.6 shows a similarity clustering of the BTC time series, as expected from the theory discussed in section 2. Indeed, oscillatory gradients can be found in the top left and bottom right corners, while smoother sections are found on the middle left side and middle right side. In addition, ascending gradients dominated the top half of the map while descending gradients occupied the bottom half of the map. In fact, this map can now be used for BTC prediction by matching each incoming data to the winning neuron of the map as shown in figure 3.6.

Tuning this SOM model was challenging as clustering gradients is far less intuitive than clustering animals, as in the practice exercise. The learning rate was varied to cluster with more or less “strength”. The input length and map size were also tweaked, as well as experimenting with normalising input BTC data.

The final SOM model used achieved a direction prediction performance of 54% which is higher than the non-linear models examined thus far and significantly higher than the linear models examined. The NMSE of 2.3906 was however significantly worse than previous models. Further SOM performances are discussed in section 4.

Another approach investigated here was the NARMAX model. For this model, trading volume was used as an exogenous input. The implementation of the NARMAX model on BTC data achieved a reasonable direction prediction performance of 54%. Once again, it is worth noting that this was only implemented on a short time frame and that more extensive analysis is required to truly characterise the NARMAX model as applied to blockchain financial time series.

Indeed, further NARMAX performance analysis is discussed in section 4, when it is applied to the DASH time series.

## 3.6. Granger Causality and Volatility Analysis

### 3.6.1. Implementation

Now that a thorough understanding of common FTS models has been obtained, the next key step to building a strong cryptocurrency forecasting model is the examination of causality. Granger Causality is a statistical test for determining whether changes in a particular time series causes changes in another. While cryptocurrencies are known to have high correlations, investigating if certain cryptocurrency pairs are causally linked can be of great use for the final DNN model. This is because, if two FTS are causally linked, an algorithm can be built to predict the non-leading FTS using data from the leading FTS. Indeed, the FTS leading and impacting the other FTS can be used as an important input to the trading algorithm used to predict the non-leading FTS.

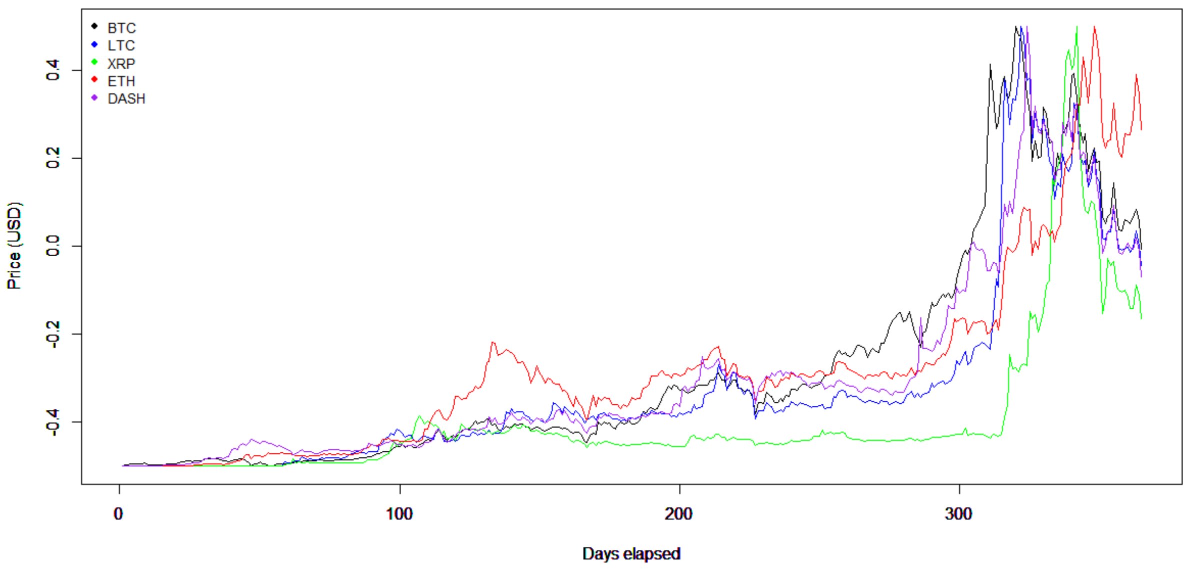


*Figure 3.7 - Long term causality (365 days of normalised cryptocurrency prices)*

Looking at the yearly data shown above in figure 3.7, there is indeed a strong indication that cryptocurrencies are at least correlated if not causally linked. This can be best observed at around the 300th day (in the 2017 dataset), where all currencies go up thereafter, before all falling back down shortly later. Unsurprisingly, the leading cryptocurrency (in black) is Bitcoin which is the first one to rise, followed by others that rise as well shortly after.

However, a formal method is required to quantitatively give a probability or degree of certainty of causation. Here, Granger Causality analysis was implemented in an R script using the “MSBVAR” package [48] and running on 30-minute cryptocurrency data (source code in appendix 4). This 30-min frequency was chosen as it is the shortest time frame where meaningful trades can be manually executed on Poloniex.

This Granger Causality analysis was repeated for periods of generally rising prices and periods of drops in prices. This was to ensure observations are not dependant on the dynamics of the market. If not controlled for, it is possible that causal links are highly different depending on the general direction of the cryptocurrency markets. Although the technique of Granger Causality has its limitations, more advanced techniques such as Transfer Entropy are beyond the scope of this report.

In addition to Granger Causality analysis, typical volatilities in cryptocurrency FTS were also quantified for analysis and contextual purposes.

### 3.6.2. Performance and Analysis

The following Granger Causality results, shown in tables 3.2 and 3.3, were obtained by comparing 5 different cryptocurrency time series to the BTC time series. Each table shows a series of p-values. Each p-value describes the statistical probability of the data, given that the null hypothesis is true. In this case the null hypothesis is that the two observed FTS are not causally linked. Hence the lower the p-value, the higher the likelihood that the FTS are causally linked. Hereafter, the threshold of $p \leq 0.01$ will be used to determine if a pair of FTS are causally linked.

| **Lag order** | **BTC-**BCH | **BCH-**BTC | **BTC-**ETH | **ETH-**BTC | **BTC-**ETC | **ETC-**BTC | **BTC-**LTC | **LTC-**BTC | **BTC-**DASH | **DASH**-BTC |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | **0.0031** | 0.0445 | **0.0038** | 0.1555 | 0.2005 | 0.2233 | 0.2581 | 0.4961 | **0.0022** | 0.3929 |
| **2** | **0.0033** | 0.1388 | **0.0035** | 0.0373 | 0.1628 | 0.3293 | 0.1729 | 0.0513 | 0.0378 | 0.4539 |
| **3** | 0.0196 | 0.0145 | **0.0061** | 0.0425 | 0.2624 | 0.4417 | 0.2638 | 0.2319 | 0.0602 | 0.3706 |
| **4** | 0.0380 | 0.1746 | 0.0151 | 0.4483 | 0.0720 | 0.2654 | 0.0461 | 0.0638 | 0.0538 | 0.0582 |
| **5** | 0.0202 | 0.0698 | 0.0120 | 0.0827 | 0.2792 | 0.0951 | 0.2805 | 0.4891 | 0.1356 | 0.2619 |
| **6** | 0.0141 | 0.0991 | 0.1913 | 0.4742 | 0.1189 | 0.4784 | 0.2469 | 0.2761 | 0.0966 | 0.1983 |
| **7** | 0.0437 | 0.7220 | 0.1662 | 0.1926 | 0.2580 | 0.2847 | 0.2959 | 0.0668 | 0.1975 | 0.0800 |
| **8** | 0.0204 | 0.5392 | 0.1147 | 0.3814 | 0.0741 | 0.4527 | 0.1393 | 0.2350 | 0.0641 | 0.4266 |
| **9** | 0.0313 | 0.3079 | 0.1019 | 0.2015 | 0.0944 | 0.4084 | 0.1190 | 0.3489 | 0.1290 | 0.3573 |
| **10** | 0.0440 | 0.4545 | 0.1321 | 0.2265 | 0.2548 | 0.4877 | 0.2025 | 0.2391 | 0.2834 | 0.2016 |

*Table 3.2 - 30-Minute Frequency Granger Causality for different lags (rising data)*

In table 3.2, the leading cryptocurrency for each pair is shown in bold on the top row and all pairs are tested in both directions (i.e. both when BTC is leading or when the other cryptocurrency is leading). P-values suggesting causality are shown in bold. Before extracting any conclusions, this analysis must be repeated for a falling period.

| **Lag order** | **BTC-** BCH | **BCH-** BTC | **BTC-** ETH | **ETH-** BTC | **BTC-** ETC | **ETC-** BTC | **BTC-** LTC | **LTC-** BTC | **BTC-** DASH | **DASH** -BTC |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | **0.0049** | 0.0377 | **0.0043** | 0.4041 | 0.2769 | 0.1289 | 0.2564 | 0.2799 | **0.0035** | 0.1516 |
| **2** | **0.0041** | 0.0942 | **0.0056** | 0.0810 | 0.1086 | 0.2133 | 0.1650 | 0.4811 | **0.0091** | 0.4638 |
| **3** | 0.0266 | 0.0155 | **0.0075** | 0.2005 | 0.2584 | 0.4180 | 0.2587 | 0.2841 | 0.1919 | 0.0998 |
| **4** | 0.0299 | 0.1357 | 0.0112 | 0.2627 | 0.1606 | 0.1398 | 0.1799 | 0.3843 | 0.0887 | 0.4091 |
| **5** | **0.0097** | 0.2059 | 0.1037 | 0.2030 | 0.0572 | 0.2540 | 0.2904 | 0.4285 | 0.0924 | 0.4675 |
| **6** | 0.0376 | 0.1594 | 0.0714 | 0.3066 | 0.1632 | 0.0770 | 0.2096 | 0.4323 | 0.1179 | 0.3988 |
| **7** | 0.0139 | 0.5169 | 0.1150 | 0.2769 | 0.2100 | 0.0497 | 0.0503 | 0.0362 | 0.0583 | 0.3632 |
| **8** | 0.0231 | 0.5201 | 0.0567 | 0.3808 | 0.1318 | 0.4836 | 0.1083 | 0.1394 | 0.1608 | 0.4629 |
| **9** | 0.0243 | 0.2629 | 0.1173 | 0.3046 | 0.2415 | 0.4297 | 0.2766 | 0.2319 | 0.1532 | 0.4886 |
| **10** | 0.0367 | 0.1960 | 0.0732 | 0.2308 | 0.0618 | 0.4180 | 0.1673 | 0.4659 | 0.1742 | 0.4485 |

*Table 3.3 - 30-Minute Frequency Granger Causality for different lags (falling data)*

When comparing tables 3.2 and 3.3, a few key observations can be made.

Firstly, all pairs have significantly lower p-values when BTC is leading but significantly higher p-values when BTC is being led by the other cryptocurrency. This means BTC has a high probability of causally leading the smaller cryptocurrencies while these smaller cryptocurrencies show no sign of leading BTC.

This supports the original hypothesis which states that changes in the BTC price have a direct causal impact on the prices of other cryptocurrencies. Indeed, BTC is seen by many people as representative of the blockchain market and if BTC is doing poorly, cryptocurrencies overall are usually also following suit.

High p-values for BTC being led by others is also expected as this suggests BTC is rarely lead by other cryptocurrencies. This suggests smaller, lower-volume cryptocurrencies have no causal impact on larger ones which makes sense as they are seldom popular enough to have the same knock-on effect as BTC has.

Secondly, it seems that Ethereum (ETH), Bitcoin Cash (BCH) and Digital Cash (DASH) meet the causality threshold of 0.01 for certain lags. It can therefore be concluded that all three ETH, BCH and DASH are causally linked to Bitcoin (BTC).

This can be directly seen as ETH has p-values below 0.01 for lags up to 1.5 hours, BCH has p-values below 0.01 for lags up to 1 hour and DASH has p-values below 0.01 for lags up to 30 minutes. In fact, DASH has p-values below 0.01 for lags up to 1 hour but this can only be seen in the falling data so is omitted as it is likely to be an odd point. These p-values are very encouraging as they suggest that the prediction of either ETH, BCH or DASH can be drastically improved by using exogenous BTC input data.

BCH being causally linked to BTC is expected as BCH is a fork of BTC. A blockchain fork is an event where a blockchain splits into two similar but entirely separate blockchains. This is usually done when a desired change in the blockchain protocol has mixed support amongst blockchain miners, as shown in equations (16) and (17). Indeed, ETH is a fork of ETC the same way as BCH is a fork of BTC. Two forks of a certain blockchain usually crash at similar times since vulnerabilities or shortcomings often affect both forks. For this reason, it would not make much sense to leverage BTC data for BCH prediction as the relationship between BTC and BCH is already common knowledge amongst traders hence the opportunities are lower. This relationship also explains why BCH-BTC values are lower than for other pairs.

On the other hand, ETH has remained for most of history the second largest cryptocurrency, with many people believing it is destined to overthrow BTC’s domination. Hence if ever BTC crashes and is overtaken by ETH in the future, it is unlikely that the BTC-leading causality will continue to hold. This makes ETH an unattractive choice to predict as any exogenous BTC inputs risk being made redundant in the future.

In addition to the above reasons, DASH seems to be the cryptocurrency with the lowest p-values of all three of the BTC-linked cryptocurrencies. Indeed, DASH has p-values as low as 0.0022 while ETH and BCH only reach p-values as low as 0.0033. This suggests exogenous BTC inputs would be of most significant use for predicting future DASH prices as they are most likely to be causally linked.

Therefore DASH, which is a smaller cryptocurrency, seems to be not only the most causal but also the best choice to attempt to predict using exogenous BTC inputs. As seen in tables 3.2 and 3.3, the optimal number of delayed BTC inputs for DASH prediction is 1 (a.k.a. 30min lags). Therefore, when future NN models are implemented it is recommended that they use a group of delayed DASH inputs with only 1 exogenous BTC input of 30min lag.

As for examining the high DASH volatility, this can clearly be observed in the following chart.

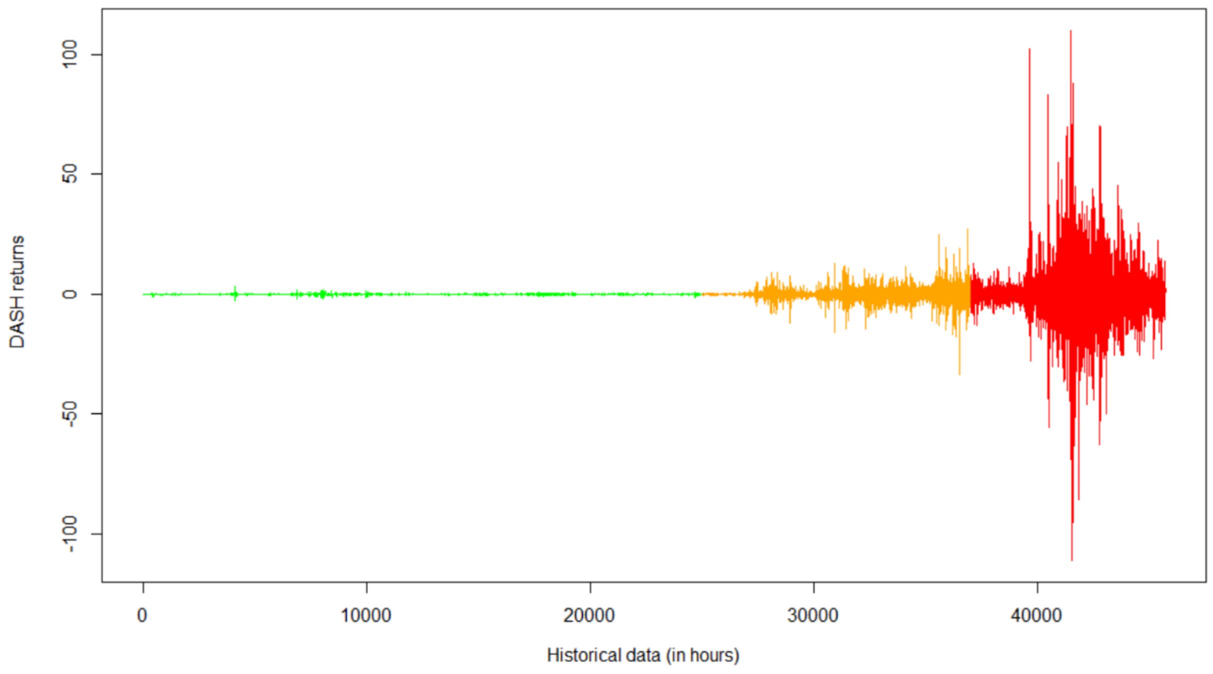


*Figure 3.8 - DASH volatility (DASH returns from 2015 to 2018)*

Figure 3.8 shows that the volatility of DASH has been getting significantly worse in recent years. This can be quantified by taking the standard deviation of this plot. Indeed, the section in green has a relatively low volatility, getting worse in the orange section and reaching its worst level in the red section during recent months.

While gold's 30-day volatility averages around 1.2% and fiat currencies around 0.5% to 1%, DASH 30-volatility averages around 25% in 2018. For comparison, BTC's 30-day volatility is usually between 10% to 15%. Indeed, smaller currencies such as DASH experience even larger volatilities than BTC, a cryptocurrency known for its high volatility.

This volatility is also seen across many other cryptocurrencies and is the biggest challenge behind blockchain price forecasting. However, this also constitutes DASH's large potential for profit, especially if this volatility can be controlled.

## 3.7. Neural Network Models

### 3.7.1. Implementation

Now that DASH has been chosen as the target cryptocurrency for prediction, simple neural networks can be built to prepare for the final DNN model. The most fundamental building block of neural networks is the single layer perceptron (SLP). A custom SLP was implemented in R from scratch for a digit recognition exercise. This SLP was then run on the MNIST dataset [49] following recommendations from LeCun (source code in Appendix 1). Writing this SLP from scratch helped immensely to gain an intuitive understanding of NNs, which were subsequently used as the building block for implementing the more sophisticated DNN models.

The next NN model to be implemented was a single hidden layer NN for forecasting FTS. This custom NN was implemented with a variable number of hidden nodes, input nodes as well as various other parameters. This was used to explore two key models.

The first single hidden layer NN model explored was one using various delayed DASH data as inputs. This was done to obtain an estimate of the order of the DASH time series, in the goal of discovering the optimal number of delayed DASH inputs for use in future models.

The second single hidden layer NN model explored was one using the optimal number of delayed DASH inputs, with the optimal number of delayed exogenous BTC inputs, over different numbers of hidden neurons and timeframes. This was done to characterise the best NN model so far and examine its performance across extended historical data.

Both NN models above were implemented in R using the “neuralnet” package [50] (source code in appendix 4).

### 3.7.2. Performance and Analysis

The practice SLP model used for digit recognition performed well with a classification performance of 94% for binary digits and a classification performance of 74% for multi-class prediction. While this is somewhat low, this is expected for a single perceptron model. Further discussions for this model can be found in Appendix 1.

This model nonetheless encouraged further investigations into more complex NN architectures for FTS modelling, such as those described hereafter.

The first single hidden layer NN explored was a DASH prediction model used to estimate the order (i.e. the optimal number of DASH inputs) of the DASH time series. This model operates at a frequency of 30 minutes. Hence, this NN takes as inputs a varying number of delayed DASH prices in steps of 30min, and outputs its prediction for the closing price every 30min, as shown below.

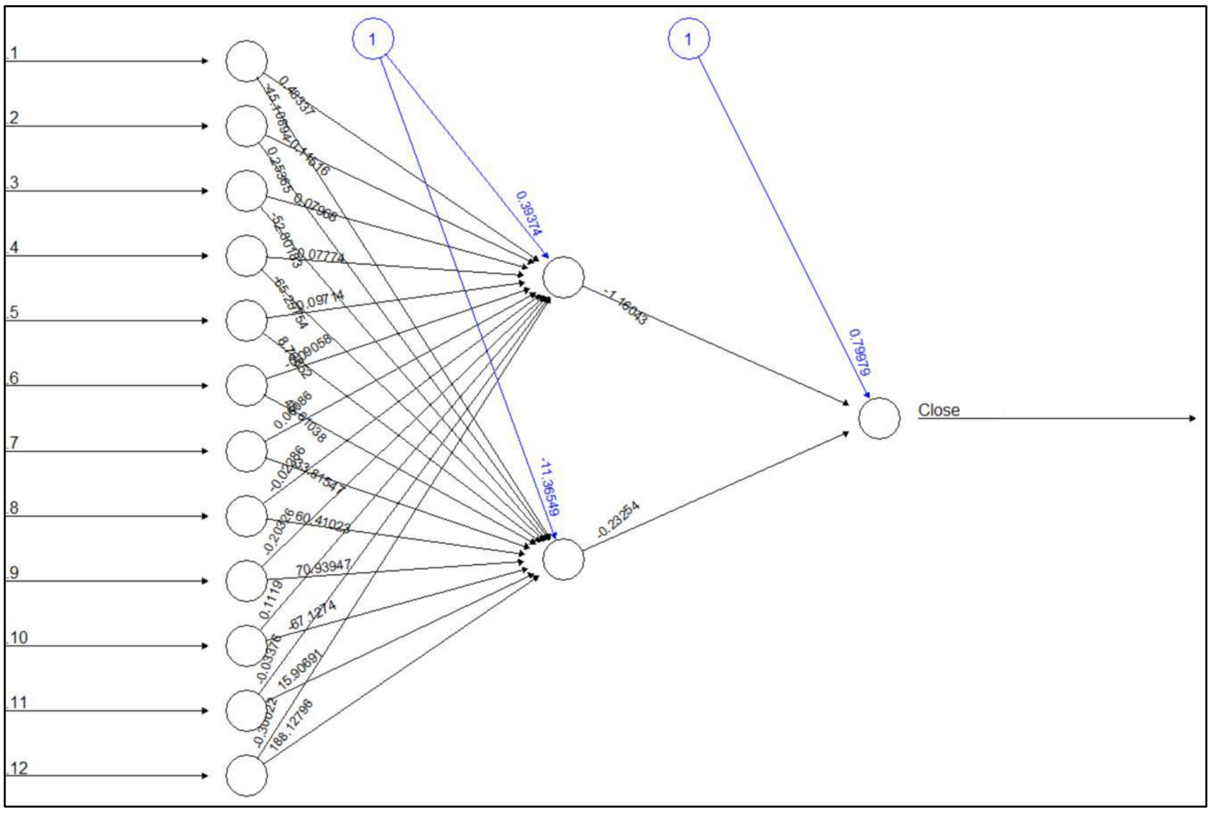


*Figure 3.9 - DASH NN (varying number of DASH inputs & hidden nodes)*

Figure 3.9 shows one of the instances of the BTC prediction NN. This particular instance has two hidden nodes and uses DASH close prices delayed from $t-1$ to $t-12$. These 12 delays of 30min each correspond to a total of 6 hours of delays. The output neuron is the prediction for the DASH close price (a.k.a. price at $t-0$), as discussed in section 2. Figure 3.9 is therefore a 12-2-1 NN architecture.

Various other architectures from the model shown in figure 3.9 were built, in order to best determine the order of the DASH time series. One model was created and tested for each varying number of DASH inputs and each varying number of hidden nodes. The results are summarised in the following table.

| DASH inputs | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hidden Neurons | 1 | - | 0.49 | 0.52 | 0.51 | 0.52 | 0.53 | 0.53 | 0.53 | 0.52 | 0.53 | 0.53 | 0.52 |
| | 2 | - | 0.51 | 0.50 | 0.52 | 0.53 | 0.52 | 0.53 | 0.54 | 0.54 | 0.53 | 0.53 | 0.53 |
| | 3 | - | 0.50 | 0.50 | 0.52 | 0.52 | 0.53 | 0.53 | 0.54 | 0.53 | 0.54 | 0.53 | 0.53 |

*Table 3.4 - DASH NN performances (varying inputs & hidden nodes)*

Table 3.4 only shows performances for networks with up to 3 hidden neurons because at 4 hidden neurons, non-convergence issues arise. These issues are not critical as a full examination of larger and more complicated architectures is performed in section 3.8 with DNN models. In addition, no NNs were created using a single input since a 1-X-1 architecture makes little theoretical sense and will undoubtedly lead to poor results.

The first trend that can be seen from table 3.4 is that as the number of delayed inputs increases, performance also increases. This is expected as the more information is given to the network, the easier it will be able to make an educated decision. The low performances of 50% to 51%, with only two DASH inputs, are not at all unexpected and are in line with theoretical expectations.

The above data also strongly suggests that 8 delayed DASH inputs is the optimal number of delayed inputs to characterise and predict the DASH time series. Indeed, for DASH inputs from 9 to 12 the performance stops improving. In these cases (from 9 to 12 inputs), the weights of the 9th, 10th, 11th and 12th input neurons were consistently close to 0. This suggests that the neural network was not really considering the data from those inputs in its output decision. It is therefore safe to suggest that the order of the DASH time series does not exceed the 8th delayed input. This corresponds to 4 hours of delayed data.

Another interesting trend is that, overall, more hidden neurons in the hidden layer seems to improve the average performance of this set of NNs. Already this single hidden layer NN is performing as strongly as the best model so far which was the SOM with a performance of 54%. This suggests further optimisation of the number of hidden neurons should therefore be investigated for subsequent NN models.

The second NN explored was a DASH prediction network with exogenous BTC inputs. This model was used to determine the optimal NN architecture across various timeframes.

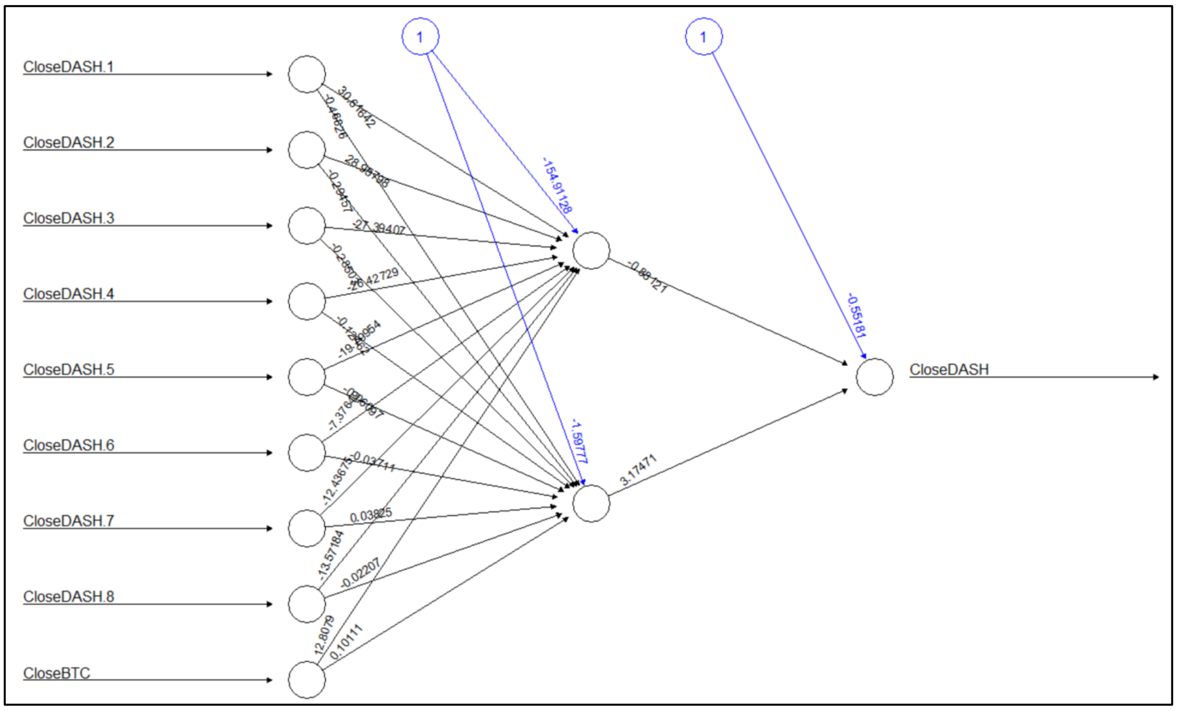


*Figure 3.10 - DASH NN with exogenous BTC (varying hidden neurons)*

Figure 3.10 shows the optimal input configuration for the DASH NN model with BTC inputs. This configuration is with 8 lagged DASH inputs and 1 lagged BTC input. This configuration was chosen because, as seen in the previous page in tables 3.4, an optimal performance is reached with 8 delays of DASH inputs. In addition, the reason why only 1 delay of BTC data was chosen is because of the Granger Causality analysis presented in section 3.6.2. Indeed, the Granger probability tables showed that causality was only beyond the threshold of 0.01 for delays up to 30min. Since the input data of this NN is also at a 30min scale, only one exogenous BTC input is used to improve performance through causality.

However, although the input data has been sufficiently optimised, it is worth determining the optimal number of hidden nodes over a wide range of timeframes. Therefore, various NN architectures were explored to determine the best number of neurons in the hidden layer. In addition, each NN architecture was trained on 12 different segments of training data to determine the optimal training, development and testing data split. The table summarising the performances of all these networks is shown below.

| Batch no. | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Hidden Neurons** | **1** | 0.48 | 0.49 | 0.50 | 0.51 | 0.49 | 0.51 | 0.50 | 0.51 | 0.52 | 0.54 | 0.55 | 0.54 |
| | **2** | 0.49 | 0.51 | 0.51 | 0.50 | 0.50 | 0.50 | 0.51 | 0.52 | 0.51 | 0.53 | 0.55 | 0.53 |
| | **3** | 0.47 | 0.51 | 0.50 | 0.51 | 0.50 | 0.49 | 0.51 | 0.52 | 0.54 | 0.54 | 0.56 | 0.54 |

*Table 3.5 - Performance of DASH NN with exogenous BTC (batching data)*

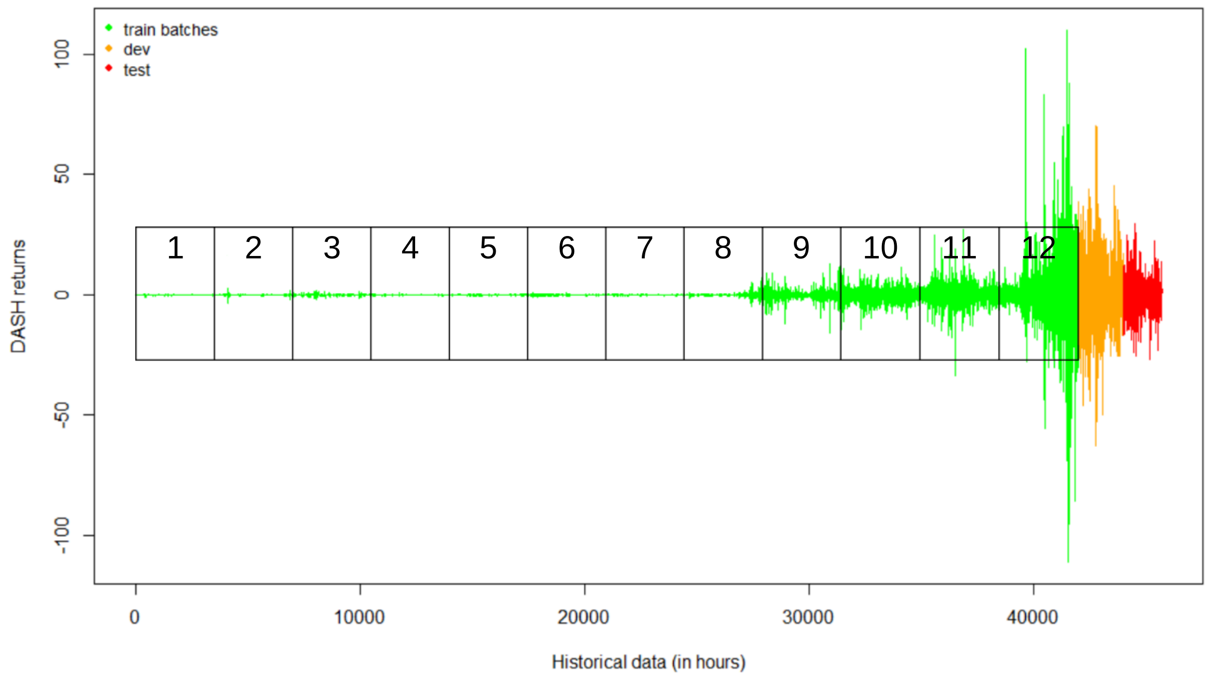


*Figure 3.11 - Batched data used for training*

Table 3.5 and figure 3.11 show the performance of various NN architectures over the full range of the DASH time series (split in 12 equal sized batches of training data). The training batches are shown in green and are used to train all the NNs created. The development data utilised is shown in orange and is used for all preliminary tests, while the test data is shown in red and is only used at the end for evaluating final NN performance. Indeed all 12 training batches were used to attempt to predict modern data (in orange and red). This is critical in examining the relevance of past data to predict current activity. This is because in DASH, unlike in many other FTS, the behaviour of the FTS has changed significantly from its launch in 2015 to its current state in 2018. From this data, significant conclusions can be drawn.

Indeed, in the very first batch, a significantly low performance is observed. This unpredictability is hypothesized to be caused by a change in DASH blockchain protocol (there was a large mining issue that was resolved at that time in 2015).

In batches 2 to 7, performance stays relatively low. This is hypothesised to be because the data being used for training the NN is not very representative of modern behaviour. In batches 8 to 12, this hypothesis is supported as performance is drastically improved. Indeed, in these periods the volatility seen in the returns data also seems to be more representative of modern DASH volatility (as shown in red in the test data). Across all batches, slightly higher performances were reached with a larger number of hidden neurons. While it is difficult to determine the best number of hidden neurons, there seems to be no sign of overfitting yet. This encourages further investigations in deep neural network architectures.

After further experimentation, the following split of data was found to yield the best results in counteracting the volatility difficulties.

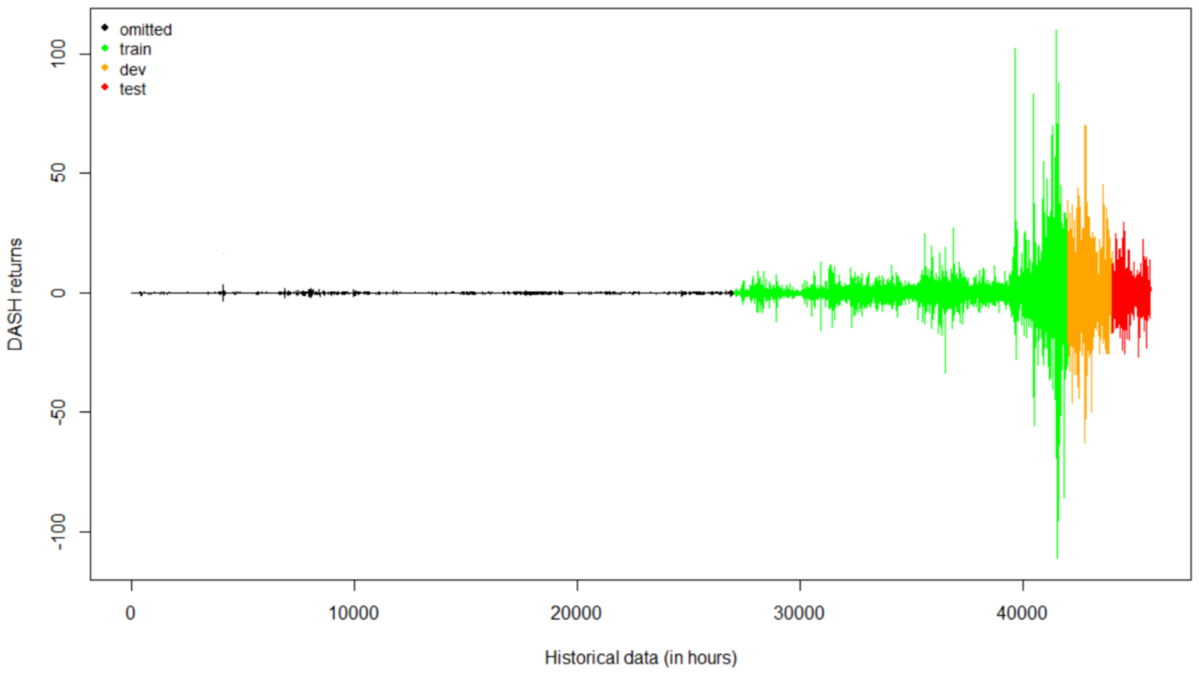


*Figure 3.12 - Optimal data split for training, development and testing*

As shown by figure 3.12, optimal performance is achieved by using as much relevant data as possible for training (green) without entering the low volatility period. This period of unrepresentative low volatilities (black) should be omitted. In addition, the training data should include some of the most volatile periods, such as December 2017. The development split (orange) should use roughly half of the remaining data while the test split (red) should contain the most recent, representative and valid data. Using this suggested split, DASH NNs with exogenous BTC inputs reached performances of up to 56.4% in the testing data.

In this section, single hidden layer neural network models with exogenous BTC inputs have consistently achieved performances of up to 56%, which is higher than all models investigated thus far. Indeed, from the performances observed in this section and in the ones that preceded it, there is strong reason to believe that deep neural networks may hold the key to better grasping and handling the higher complexities, challenges and volatility of cryptocurrencies.

## 3.8. Deep Learning Models

### 3.8.1. Implementation

Before beginning to apply deep learning models to cryptocurrencies, some preparatory work was done by practicing modelling concrete, cancer cells and more using DNNs. This preparation can be found in a separate GitHub repository [51].

Subsequently, for the implementation of DNNs to cryptocurrencies, computing power considerations became top priority. The main arithmetic computation kernel cannot be entirely written in a high-level language like R but should rather rely on a lower level language to benefit from hardware optimisations. In addition, the issues of long training times identified in section 2 can be overcome by transitioning from CPU-based NN training to GPU-based NN training.

Indeed, TensorFlow [52] is a remarkable ML library and was essential in this implementation of deep learning models. Keras [53] is also a particularly versatile library, capable of running on top of TensorFlow for fast experimentation with DNNs. The final implementation is built using an R API wrapper to Keras, running on top of TensorFlow. The particular installation of Tensorflow used was the GPU-based CUDA-compatible version. In this setup, NVIDIA’s CuDNN [54] is used as the main GPU-accelerated computation kernel and many calculations are made using a 2GiB GTX 960M graphics card. In this fully-connected DNN implementation, an R script was written to interface via the Keras API to TensorFlow libraries (source code in appendix 4). Due to time constraints, the depth was limited to two hidden layers.

A few trials of various architectures were first experimented with. Tuning parameters were tinkered with and performances were evaluated using training data. A script was then written to individually build and test 400 different DNNs (source code in appendix 4). Although this script was challenging as it took four days to run to completion, it produced excellent and insightful results.

Once the optimal architecture had been found amongst the 400 DNN contenders, the winning DNN was put through to the next stage: live trials in a sandbox. In order to run these live trials, an R script had to be written to fetch new data every 30 minutes, update the winning DNN, and output the trading recommendation in reasonable time to execute the trade. This script achieves the ultimate aim of this report and its performance is further compared to others in section 4 (source code in appendix 4).

### 3.8.2. Performance and Analysis

A typical DNN training phase is shown in the figure below.

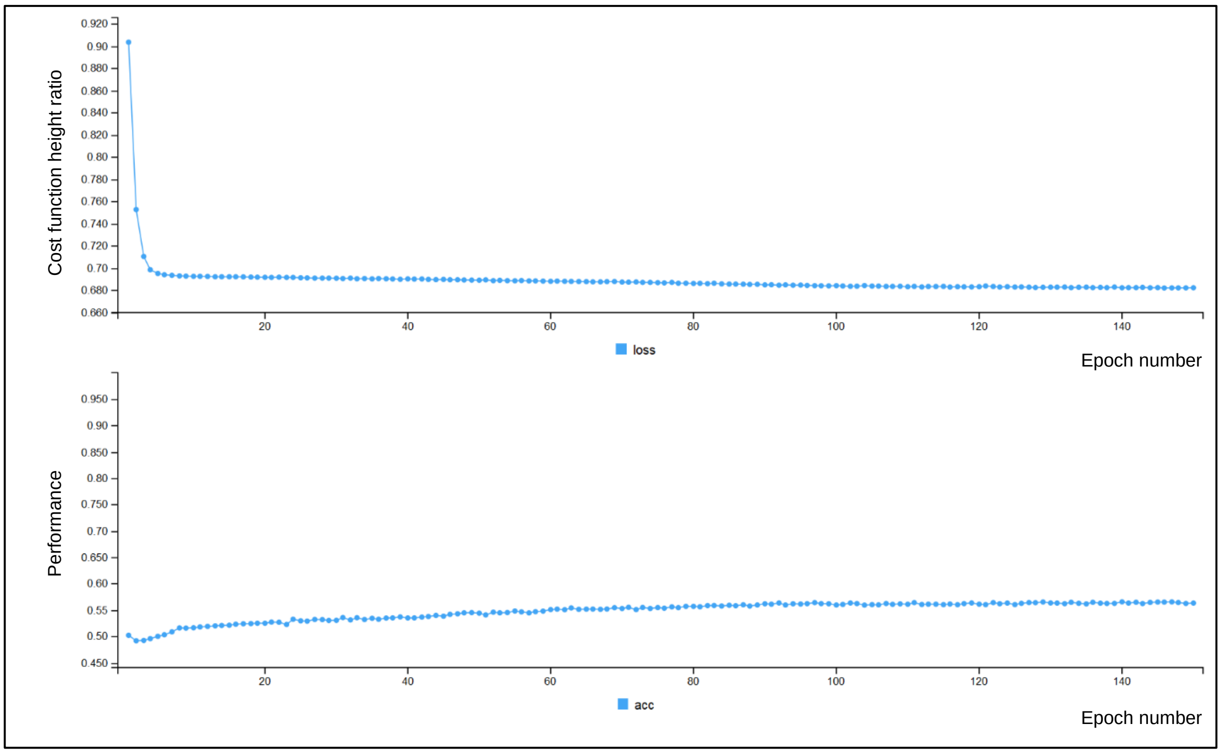
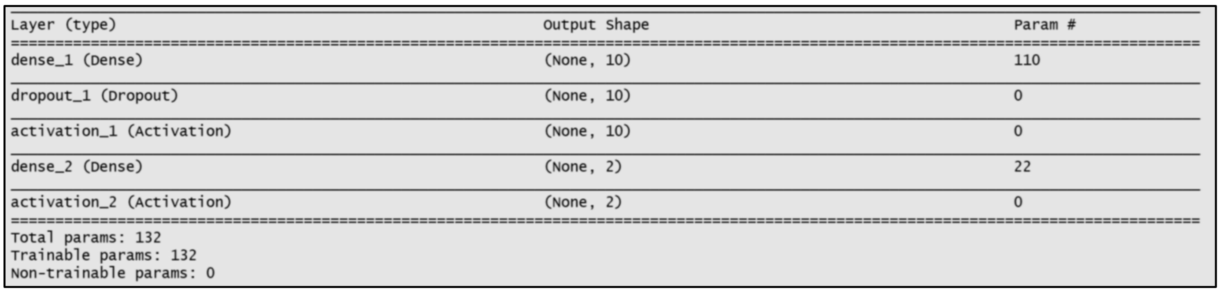


*Figure 3.13 - DNN training phase (cost function & accuracy)*

As shown in figure 3.13, the current height position on the multi-dimensional cost function (or loss) is shown on the top graph while the model accuracy is shown on the bottom one. As expected from equations (13) - (15), as the cost function is minimised, the performance of the predictions increases. At the start of this specific run, the accuracy is seen to increase from slightly below 50% to around 56% at the 150th epoch (training iteration).

This particular training session was undertaken on the following DNN architecture, as shown on the following page.

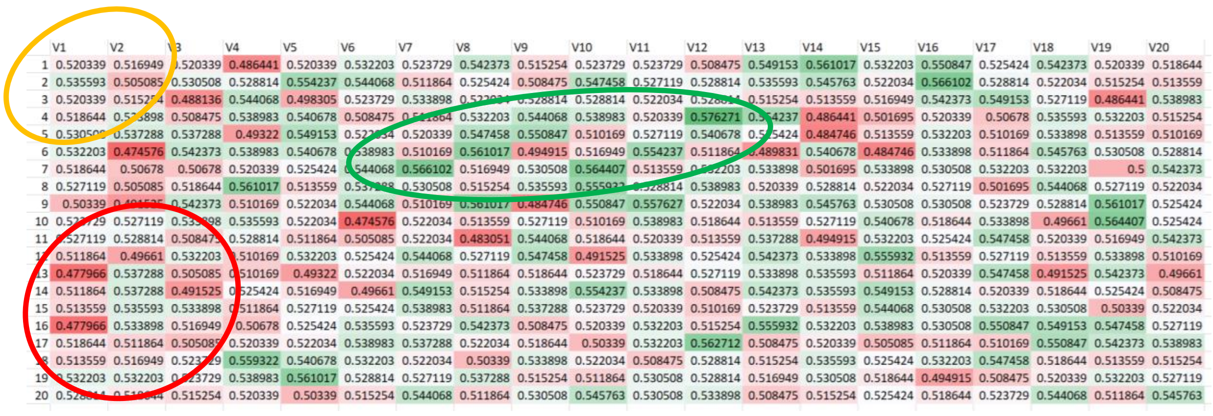


```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dropout_1 (Dropout)          (None, 10)                0
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 22
_________________________________________________________________
activation_2 (Activation)    (None, 2)                 0
=================================================================
Total params: 132
Trainable params: 132
Non-trainable params: 0
```

*Figure 3.14 - Typical DNN architecture used for tuning parameters*

Although figure 3.14 shows a typical DNN architecture, many others were experimented with, as shown later in this section. Various other parameters were also experimented with such as dropout (when networks were overfitting), batching (to improve training speed) and 10-fold cross-validation (for performance estimates). The ADAM optimiser, as discussed in equations (13) - (15), was used as it is one of the most efficient for training DNNs.

While several quantitative methods for tuning deep neural networks exist, we have seen in section 2 that this is often more of an art than a science. To get a good overview of the impact that DNN architecture has on performance, a pool of 400 candidate DNNs models were generated and tested. All these DNNs had two hidden layers with 1 to 20 hidden nodes in each layer. An annotated table of colour gradients of all their respective performances is shown below.



| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | V15 | V16 | V17 | V18 | V19 | V20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.520339 | 0.516949 | 0.520339 | 0.486441 | 0.520339 | 0.532203 | 0.523729 | 0.542373 | 0.515254 | 0.523729 | 0.523729 | 0.508475 | 0.549153 | 0.561017 | 0.532203 | 0.550847 | 0.525424 | 0.542373 | 0.520339 | 0.518644 |
| 2 | 0.535593 | 0.505085 | 0.530508 | 0.528814 | 0.554237 | 0.544068 | 0.511864 | 0.525424 | 0.508475 | 0.547458 | 0.527119 | 0.528814 | 0.535593 | 0.545763 | 0.522034 | 0.566102 | 0.528814 | 0.522034 | 0.515254 | 0.513559 |
| 3 | 0.520339 | 0.515254 | 0.488136 | 0.544068 | 0.498305 | 0.523729 | 0.533898 | 0.522034 | 0.528814 | 0.528814 | 0.522034 | 0.528814 | 0.515254 | 0.513559 | 0.516949 | 0.542373 | 0.549153 | 0.527119 | 0.486441 | 0.538983 |
| 4 | 0.518644 | 0.533898 | 0.508475 | 0.538983 | 0.540678 | 0.508475 | 0.511864 | 0.532203 | 0.544068 | 0.538983 | 0.520339 | 0.576271 | 0.554237 | 0.486441 | 0.501695 | 0.520339 | 0.50678 | 0.535593 | 0.532203 | 0.515254 |
| 5 | 0.530508 | 0.537288 | 0.537288 | 0.49322 | 0.549153 | 0.522034 | 0.520339 | 0.547458 | 0.550847 | 0.510169 | 0.527119 | 0.540678 | 0.525424 | 0.484746 | 0.513559 | 0.532203 | 0.510169 | 0.533898 | 0.513559 | 0.510169 |
| 6 | 0.532203 | 0.474576 | 0.542373 | 0.538983 | 0.540678 | 0.538983 | 0.510169 | 0.561017 | 0.494915 | 0.516949 | 0.554237 | 0.511864 | 0.489831 | 0.540678 | 0.484746 | 0.533898 | 0.511864 | 0.545763 | 0.530508 | 0.528814 |
| 7 | 0.518644 | 0.50678 | 0.50678 | 0.520339 | 0.525424 | 0.544068 | 0.566102 | 0.516949 | 0.530508 | 0.564407 | 0.513559 | 0.532203 | 0.533898 | 0.501695 | 0.533898 | 0.530508 | 0.532203 | 0.532203 | 0.5 | 0.542373 |
| 8 | 0.527119 | 0.505085 | 0.518644 | 0.561017 | 0.513559 | 0.537288 | 0.530508 | 0.515254 | 0.535593 | 0.555932 | 0.528814 | 0.538983 | 0.520339 | 0.528814 | 0.522034 | 0.527119 | 0.501695 | 0.544068 | 0.527119 | 0.522034 |
| 9 | 0.50339 | 0.491525 | 0.542373 | 0.510169 | 0.522034 | 0.544068 | 0.510169 | 0.561017 | 0.484746 | 0.550847 | 0.557627 | 0.522034 | 0.538983 | 0.545763 | 0.530508 | 0.530508 | 0.523729 | 0.528814 | 0.561017 | 0.525424 |
| 10 | 0.523729 | 0.527119 | 0.533898 | 0.535593 | 0.522034 | 0.474576 | 0.522034 | 0.513559 | 0.527119 | 0.510169 | 0.538983 | 0.518644 | 0.513559 | 0.527119 | 0.540678 | 0.518644 | 0.533898 | 0.49661 | 0.564407 | 0.525424 |
| 11 | 0.527119 | 0.528814 | 0.508475 | 0.528814 | 0.511864 | 0.505085 | 0.522034 | 0.483051 | 0.544068 | 0.518644 | 0.520339 | 0.513559 | 0.537288 | 0.494915 | 0.532203 | 0.525424 | 0.547458 | 0.520339 | 0.516949 | 0.542373 |
| 12 | 0.511864 | 0.49661 | 0.532203 | 0.510169 | 0.532203 | 0.525424 | 0.544068 | 0.527119 | 0.547458 | 0.491525 | 0.533898 | 0.525424 | 0.542373 | 0.533898 | 0.555932 | 0.513559 | 0.527119 | 0.513559 | 0.533898 | 0.510169 |
| 13 | 0.477966 | 0.537288 | 0.505085 | 0.510169 | 0.49322 | 0.522034 | 0.516949 | 0.511864 | 0.518644 | 0.523729 | 0.518644 | 0.527119 | 0.533898 | 0.535593 | 0.511864 | 0.520339 | 0.547458 | 0.491525 | 0.542373 | 0.49661 |
| 14 | 0.511864 | 0.537288 | 0.491525 | 0.525424 | 0.516949 | 0.49661 | 0.549153 | 0.515254 | 0.533898 | 0.554237 | 0.533898 | 0.508475 | 0.542373 | 0.535593 | 0.549153 | 0.528814 | 0.520339 | 0.518644 | 0.525424 | 0.508475 |
| 15 | 0.513559 | 0.535593 | 0.533898 | 0.511864 | 0.527119 | 0.525424 | 0.538983 | 0.511864 | 0.537288 | 0.523729 | 0.520339 | 0.510169 | 0.523729 | 0.513559 | 0.544068 | 0.530508 | 0.532203 | 0.530508 | 0.50339 | 0.522034 |
| 16 | 0.477966 | 0.533898 | 0.516949 | 0.50678 | 0.525424 | 0.535593 | 0.523729 | 0.542373 | 0.508475 | 0.520339 | 0.532203 | 0.515254 | 0.555932 | 0.532203 | 0.538983 | 0.530508 | 0.550847 | 0.549153 | 0.547458 | 0.527119 |
| 17 | 0.518644 | 0.511864 | 0.505085 | 0.520339 | 0.522034 | 0.538983 | 0.537288 | 0.522034 | 0.518644 | 0.50339 | 0.532203 | 0.562712 | 0.508475 | 0.520339 | 0.505085 | 0.511864 | 0.510169 | 0.550847 | 0.542373 | 0.538983 |
| 18 | 0.513559 | 0.516949 | 0.523729 | 0.559322 | 0.540678 | 0.532203 | 0.522034 | 0.50339 | 0.533898 | 0.522034 | 0.508475 | 0.528814 | 0.515254 | 0.535593 | 0.525424 | 0.532203 | 0.547458 | 0.518644 | 0.513559 | 0.515254 |
| 19 | 0.532203 | 0.532203 | 0.523729 | 0.538983 | 0.561017 | 0.528814 | 0.527119 | 0.537288 | 0.515254 | 0.511864 | 0.530508 | 0.528814 | 0.516949 | 0.530508 | 0.518644 | 0.494915 | 0.508475 | 0.520339 | 0.532203 | 0.527119 |
| 20 | 0.528814 | 0.518644 | 0.515254 | 0.520339 | 0.50339 | 0.515254 | 0.544068 | 0.511864 | 0.530508 | 0.545763 | 0.530508 | 0.533898 | 0.508475 | 0.515254 | 0.525424 | 0.518644 | 0.523729 | 0.544068 | 0.511864 | 0.545763 |

*Figure 3.15 - Performance of 400 different DNNs on DASH prediction*

Figure 3.15 is interesting as it reveals three key trends regarding DNN architecture when attempting to predict the price of DASH.

Firstly, the models with more hidden nodes in the first hidden layer followed by fewer hidden nodes in the second hidden layer seem to perform the best. Indeed, the strongest DNNs are all in this section (circled in green) and reach very impressive performances of up to 57.6%.

Secondly, the models with a very small number of hidden neurons (circled in orange) behave at the same performance as their more basic linear counterparts such as ARMA. These models indeed do not show any signs of overfitting but are not particularly strong contenders due to their average predictability performance. This is in line with theoretical expectations.

Thirdly, the models with a deeper second layer and a shallower first layer (circled in red) perform much worse on the testing data. This is likely to be caused by the fact that these networks are overfitting the training data, leading to a significantly poorer performance on the test data as shown above. Another visualisation of this data can be achieved by plotting a 3D graph as shown below.

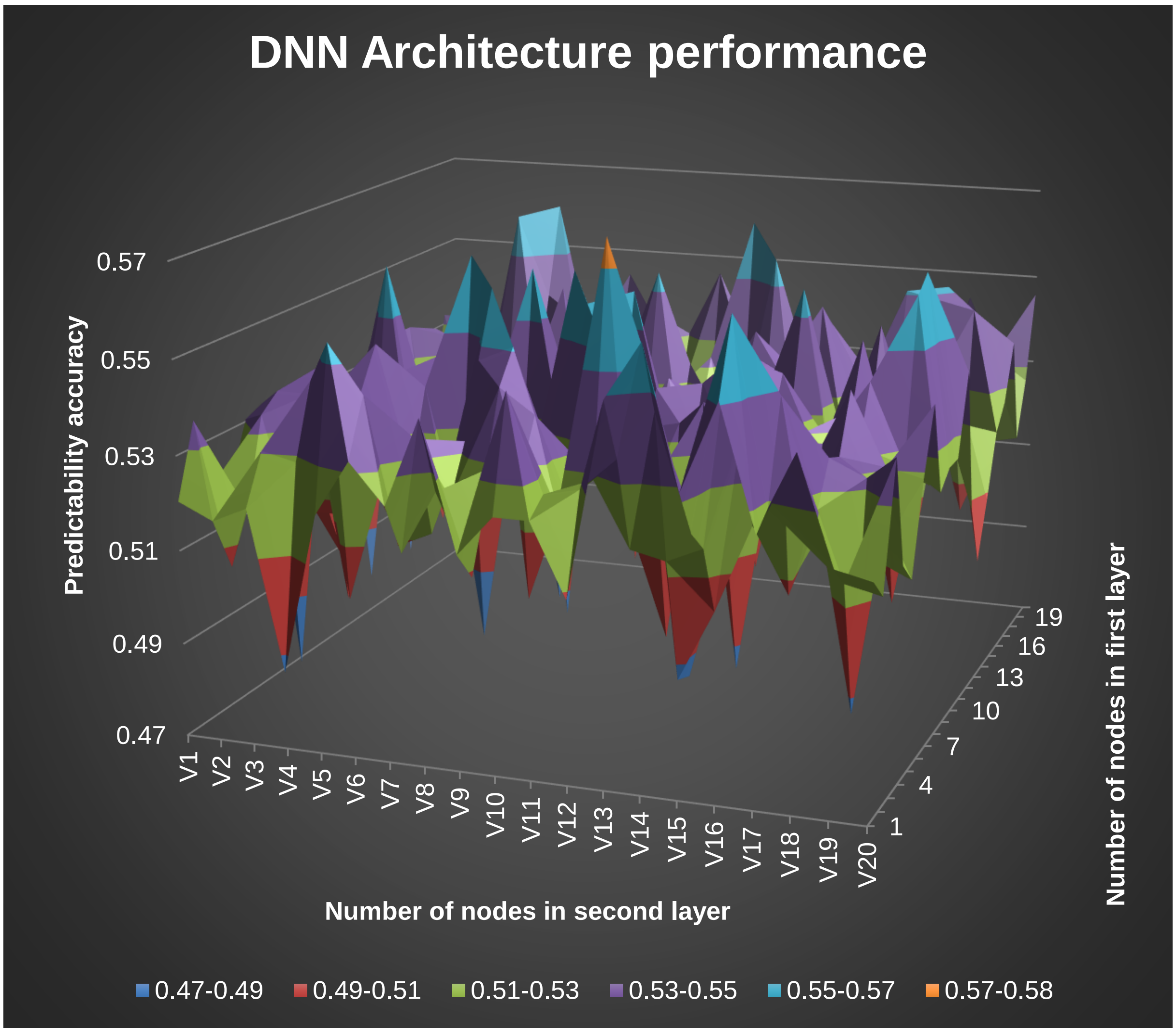


*Figure 3.16 - 3D Graph of DNN performance with changing architecture*

This visualisation is equally interesting as it shows where the different areas of performance are concentrated.

The areas of poor performance due to overfitting, i.e. no better than random, are shown in red. Areas of adequate performance, i.e. no better than linear ARIMA models, are in green while areas of strong performance, i.e. equivalent or better than non-linear models, are in purple. Areas of excellent performance, i.e. better than all previous models are shown in blue. Finally, the best models are shown in the orange spike (at 12, 4) while the worst overfitting models are in dark blue downward spikes. From the above analysis, the DNN with hidden layers of 12 nodes and 4 nodes respectively was selected as the final model and best candidate for live testing with an expected performance of around 58%.

There is however a way of improving this performance of 58%. This can be achieved by considering confidence intervals of the DNN as it outputs each prediction. Maximising performance can be done by only trading on very confident outputs. Fortunately, the Keras library provides an efficient way to extract confidence intervals from TensorFlow-built DNNs. These confidence intervals were exploited as follows.

| **Confidence** | **≥ 50%** | **≥ 52.5%** | **≥ 55%** | **≥ 57.5%** | **≥ 60%** | **≥ 65%** |
|---|---|---|---|---|---|---|
| **Trades executed** | 100.0% | 91.5% | 76.5% | 51.0% | 11.0% | 2.0% |
| **Accuracy** | 0.580 | 0.596 | 0.641 | 0.657 | 0.682 | 0.75* |

*Table 3.6 - Impact of confidence intervals of accuracy and trade frequency*

Table 3.6 shows that the base performance of 58% can be increased as far as the high 60s. However, this comes at the cost of a decreased total number of trades, as shown by the trade execution percentages. Indeed, with confidence intervals of 65% only 4 trades were executed so the value of 75% is somewhat misleading (as identified by the *). The second-best performance is 68.2% and is achieved using confidence intervals of 60%+ on the outputs of the optimal DNN. However, it seems that this prediction performance only increases by around 1.5% from the previous confidence interval, while the percentages of trades executed drops dramatically from 51% to 11%. This may translate to a significant decrease in total revenue if implemented on Poloniex.

This highlights that using confidence intervals is a constant trade-off with trading frequency. Indeed, optimising this trade-off algorithmically is something that could be examined further in the context of future blockchain time series forecasting research.

Using all the knowledge and information obtained, a final trading algorithm can be built to execute trades while running in a sandbox on live Poloniex data. Here, the most optimal DNN with a base performance of roughly 58% is chosen. The confidence interval feature is switched off to execute all trades on all outputs, at any confidence level.

The results of the live sandbox trials are shown in the following table.

| **Time** | **12:30** | **13:00** | **13:30** | **14:00** | **14:30** | **15:00** | **15:30** | **16:00** | **16:30** |
|---|---|---|---|---|---|---|---|---|---|
| **Trade number** | **1 (test)** | **2 (test)** | **3 (test)** | **4 (test)** | **5 (test)** | **6 (test)** | **7 (test)** | **8 (test)** | **9 (test)** |
| **Prediction** | BUY | SELL | SELL | BUY | SELL | SELL | BUY | SELL | BUY |
| **Confidence** | 59.2% | 53.5% | 61.6% | 51.7% | 50.3% | 58.1% | 52.6% | 54.7% | 55.2% |
| **Outcome** | ✓ | ✓ | X | X | X | ✓ | X | X | ✓ |

*Table 3.7 - Sandbox Trading Results (Live Data - 15/03/18)*

Table 3.7 shows that the sandboxed DNN recommendations are reasonable. Estimating a performance from 9 trades would not be statistically significant but the DNN trades seem to be within expectations. Apart from the exception at 13:30, most of the correct trades had high confidences while most of the incorrect trades had low confidences.

The script works as designed although it highlights one key overhead which is the speed of executing a trade. For example, in test trade 1 the algorithm fetched the new data at 12:30. It then computed the new prediction and output its recommendation before 12:31. However, executing the trade on the Poloniex exchange is usually instantaneous except if the order book is thin. If so, trades can take slightly longer (up to 1min usually). This will introduce error in the trading price and this source of uncertainty will be discussed further in the limitations in section 5. In addition, although it seems that DNNs outperform conventional NNs, further investigations are required to look into deeper DNNs with more than 2 hidden layers.

A new stage has nevertheless been reached where an optimal DNN contender is finally ready for live trials outside a sandbox, using real money on a real exchange. The performance of this real-life DNN trial should be similar to the performances of 58% detailed in this section. However, before doing this, it is worth reviewing all the models explored so far.

# 4. Model Evaluation

## 4.1. Evaluation of all Model Performances

Comparing all the models implemented thus far allows for the examination of their respective performances when applied to the DASH cryptocurrency. The first few models (AR, ARMA, ARIMA, ARIMA-GARCH, SOM & NARMAX) were all initially applied to the BTC time series so must be rerun on the DASH time series. Indeed, this occurred as the DASH blockchain had not yet been identified as the target for prediction. The results of this extended cross-model analysis are summarised below.

| Model | | Predictability | NMSE | Notes |
|---|---|---|---|---|
| AR (Yule) | AR(2), AR(4), AR(5) | 0.498 | 0.0218 | *Mean of top 3 models (by predictability)* |
| AR (Burg) | AR(2), AR(3), AR(5) | 0.493 | 0.0264 | *Mean of top 3 models (by predictability)* |
| ARMA | ARMA(2,2), ARMA(3,1), ARMA(2,4) | 0.515 | 0.0203 | *Mean of top 3 models (by predictability)* |
| ARIMA | ARIMA(2,1,1), ARIMA(3,1,4), ARIMA(1,1,5) | 0.529 | 0.0358 | *Mean of top 3 models (by predictability)* |
| ARIMA GARCH | ARIMA(5,1,0) GARCH(1,1), ARIMA(4,1,0) GARCH(1,1), ARIMA(3,1,5) GARCH(1,1) | 0.530 | 0.7256 | *Mean of top 3 models (by predictability)* |
| SOM | Time Delayed Inputs | 0.544 | 2.3585 | *Size: 10x10* |

*Table 4.1 - Comparison of best model performances (DASH time series only)*

Table 4.1 shows the average performances of all the models investigated, applied to one year of the DASH FTS. This was usually repeated over multiple runs, especially for the SOM model, to get performances as representative as possible.

| Model | Predictability | Notes |
|---|---|---|
| **NARMAX** | 0.542 | *Exogenous input: volume* |
| **NN (with exogenous BTC input)** | 0.564 | *Using 30min lags of BTC data* |
| **DNN (with exogenous BTC input)** | 0.576 | *Hidden layers: 14,2* |

*Table 4.2 - Comparison of best model performances (DASH with exogenous BTC)*

Table 4.2 shows the average performances of all the models investigated which used more than just DASH data. The NARMAX, NN and DNN models also used volume data or BTC data to compliment the DASH FTS being predicted. Both tables 4.1 and 4.2 offer significant insight on blockchain-based FTS and their predictability potential. Indeed, while predicting cryptocurrencies seems to be more challenging than fiat currencies, this has now been shown to nevertheless be feasible.

Consistent performances of up to 54% were achieved by only using the DASH time series to predict future DASH prices. The best model from those investigated in table 4.1, was the SOM using time delayed inputs. There seems to be a trade-off however between performance and NMSE as this SOM also had the worst NMSE by far at 2.3585. On the other hand, drastic performance improvements are reached when using external data such as DASH volume or the BTC FTS. The best model from those investigated in table 4.2, was the DNN using an exogenous BTC input. This model consistently reached performances of up to 57.6%. The least performant models were the AR models which only achieved performances close to random guesses. This may be due to their over-reaching assumptions and limitations as discussed in equations (1) to (3).

The results presented and analysed above indeed highlight the significant theoretical advantages of deep neural networks over their simpler counter parts. In fact, it seems that the complexity and volatility of cryptocurrencies can be better tackled by deeper models. Although only two hidden layers were investigated here because of time constraints, perhaps deeper networks would have even more of a performance increase. This warrants a final proof-of-concept live trading session to reinforce the potential of DNNs on predicting the close price of the DASH blockchain FTS.

## 4.2. Final Model Analysis and Live Testing

One of the most convincing approaches to demonstrating the efficacy of a FTS predictive model beyond historical data is by subjecting it to live trading. The optimal DNN discussed in section 4.1 was therefore used during a series of three live trading sessions on the Poloniex cryptocurrency exchange. The trade log of this activity is shown in full detail in Appendix 5 and summarised in the tables below.

| **Time** | **17:30** | **18:00** | **18:30** | **19:00** | **19:30** | **20:00** | **20:30** | **21:00** | **21:30** |
|---|---|---|---|---|---|---|---|---|---|
| **Trade number** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Prediction** | BUY | BUY | SELL | SELL | SELL | BUY | BUY | SELL | SELL |
| **Confidence** | 50.6% | 64.8% | 57.7% | 60.3% | 60.2% | 53.6% | 55.1% | 51.2% | 50.2% |
| **Outcome** | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | X |

*Table 4.3 - Live trading session #1 (18/03/2018)*

| **Time** | **11:00** | **11:30** | **12:00** | **12:30** | **13:00** | **13:30** | **14:00** | **14:30** | **15:00** |
|---|---|---|---|---|---|---|---|---|---|
| **Trade number** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** |
| **Prediction** | SELL | BUY | SELL | BUY | SELL | SELL | SELL | SELL | BUY |
| **Confidence** | 52.6% | 51.3% | 55.4% | 57.6% | 52.3% | 59.9% | 55.3% | 58.1% | 56.0% |
| **Outcome** | ✓ | X | X | ✓ | X | ✓ | ✓ | X | ✓ |

*Table 4.4 - Live trading session #2 (29/03/2018)*

| **Time** | **6:30** | **7:00** | **7:30** | **8:00** | **8:30** | **9:00** | **9:30** | **10:00** | **10:30** |
|---|---|---|---|---|---|---|---|---|---|
| **Trade number** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** |
| **Prediction** | SELL | BUY | BUY | SELL | SELL | SELL | SELL | BUY | SELL |
| **Confidence** | 58.8% | 50.9% | 56.7% | 57.5% | 51.0% | 52.5% | 61.3% | 59.1% | 54.5% |
| **Outcome** | X | ✓ | ✓ | ✓ | X | X | ✓ | ✓ | X |

*Table 4.5 - Live trading session #3 (11/04/2018)*

In each of the trading sessions shown in tables 4.3 - 4.5, a total of 9 trades were executed each session for a total of 27 trades overall. These sessions were run at different times of day, as widely spread as possible, over the course of two months of 2018. By averaging all the trades together, this results in an overall live prediction performance of around 59% without use of any confidence intervals.

Before any analysis is discussed, it is worth prefacing with the statement that a sample of 27 trades is not necessarily statistically significant. However, as these trades take significant time to run (over 13 hours), record and collate, further live trials are set to be undertaken in summer 2018. Nevertheless, the 27 trades thus far will give a rough estimate of the success or failure of the live DNN deployment onto the Poloniex exchange.

A total of 10 “BUY” predictions and 17 “SELL” predictions were made, with an average confidence level of 55.7%. Although the average performance of 59% is perhaps not representative, this suggests that the DNN deployed has a significant bias toward accurate trading recommendations and is not overconfident.

The first live trading session had the highest confidence peaks, 64.8%, but also the lowest confidence dips, 50.2%. In this period the DNN also had the highest average confidence. Indeed, this yielded the highest performance amongst the three sessions, without confidence intervals being used. Subsequent trading sessions were less confident and achieved less high performances as they had one more wrong trade than the first session. If the confidence feature had been activated at a threshold of 55%, 16 out of 27 trades would have been executed resulting in a performance of 75%. By contrast, if the confidence had been activated at 60%, only 4 trades would have been executed with a performance of 100%. However, as the former strategy would have earned more money, perhaps a proportional implementation of confidence intervals would have been better (i.e. trading a volume directly proportional to the confidence level).

As the net sum traded every time was a fixed amount (roughly 1 USD worth), the total net earnings from all three trading sessions were calculated to be $0.47 (see appendix 5). Since these 3 sessions amount to approximately 13 hours, one full working day of trading would yield returns of roughly 2.73%. However, after considering Coinbase fees, Poloniex fees, and blockchain transaction fees, the overall trading returns do not break even. This is discussed further in section 5.

These results are encouraging and in line with expectations, but significantly more live trials are needed to express a performance that is statistically significant. This section also highlights the need for further research into confidence interval trading strategies.

# 5. Conclusion

## 5.1. Project Achievement and Limitations

Although the live trading trials yielded high returns of 2.73%, this failed to cover all fees from Coinbase (USD conversion), Poloniex (trading fees) and blockchain nodes (wallet transaction fees). However, these fees can be decreased drastically by trading larger volumes and converting back to USD at the end of a year's worth of trading rather than after only one day's worth of trading. With these economies of scale, trading returns are expected to be consistently above 1% per day. This is encouraging as the live trials have effectively demonstrated the practical viability of the developed algorithm. This proof-of-concept is strong evidence to support the achievement and surpassing of the original aim of this thesis.

The research goal behind learning the theory of FTS analysis was only partially met. Indeed, the mathematical concepts behind FTS forecasting were extremely challenging to grasp and the huge range of different models means that it is almost impossible to cover all the background literature, especially with limited time. The data collection and processing goal was fully met as a rich database of various features was collated in clean .csv files. These files will no doubt continue to be of use for the future of this project, as the intention is to pursue this investigation beyond the scope of a bachelor thesis. The goal of implementation of algorithms in R and TensorFlow has also been partially met. For similar reasons as before, this goal was limited by time and there is always more learning to be done, especially with libraries as vast as TensorFlow. DNN architecture investigations and Granger Causality analysis enabled the tuning of DNNs and the identification of DASH as the target cryptocurrency hence these goals were fully met. Furthermore, the DNN performance goal of 55% has been met and surpassed as the final model consistently achieved performances of up to 58%. This means that the algorithm developed exceeds the level of current DNN competitors.

There are nonetheless some limitations to this thesis that are important to note. The main limitation is that although backtesting and live testing give reasonable estimates of future performance, there is no method that guarantees that the performances demonstrated currently will hold for the foreseeable future. Indeed, market dynamics are constantly changing and this principle holds true for the volatile cryptocurrency market. The best example of this would be the dependence of the final DNN algorithm on the causal link between BTC and DASH. If this causal link were to stop holding true, due to changing market dynamics, the DNN's performance would be expected to drop significantly. In addition, only up to two hidden layers were investigated but perhaps deeper DNNs would perform even better.

The second main limitation of this thesis is the viability of trading via USD Tether (USDT). USDT is the only way of trading DASH directly for dollars on the Poloniex exchange. However, while USDT is a cryptocurrency which claims to be fixed to the price of the US dollar by backing each USDT to one real US dollar, this is subject of much controversy. There is in fact little evidence of such backing with real US dollars and thus the price of USDT is highly speculative. In fact, if it was discovered that USDT is not actually backed, its price would collapse. In addition, having to go through USDT markets rather than the more popular option of trading DASH for BTC means that order books are often thin. As seen in section 4, thin order books can cause trading delays if orders cannot be met immediately. This can be a major concern due to its impact on the final trading price achieved hence final performance.

The immediate implication of the work achieved in this thesis is the practical demonstration that volatile cryptocurrencies can indeed be forecast and that deep learning algorithms outperform conventional NNs in cryptocurrency FTS forecasting. This thesis' achievements are particularly notable given the multiple challenges encountered throughout and given the fact that DASH has been found not to follow natural time series in its volatility characteristics.

## 5.2. Challenges and Future Investigations

One of the main challenges of this report was the art of tuning a multitude of DNN model parameters without falling into the trap of overfitting. Indeed, there exists various algorithmic alternatives instead of manual selection or randomly generating pools of candidate models. One of the most promising alternatives may be found by investigating evolutionary algorithms. These algorithms are based on the principle of selectively removing underperforming tuning parameters and allowing only the best models to "reproduce". Although this requires far more computation than is currently available, these DNA-inspired selection methods could provide the key to complimenting artificial neural networks such as those implemented in this thesis.

Another main challenge was the issue of extensive computation time. While this was addressed in this report by using GPU-based computing with low level CuDNN algorithms using ADAM optimisation, this can also be addressed by modifying the type of deep learning model used. In fact, echo state networks are a promising alternative worth investigating to help reduce overall training time. These networks are initialised through random node connections organised in a reservoir. This reservoir architecture drastically decreases computational requirements without significantly compromising performance, due to its partially connected structure.

Another difficulty worth further investigation is the lack of comparison of the developed DNN to other deep learning alternatives. Indeed, recent cross-model studies have found that restricted Boltzmann machines are showing increasing promise in FTS forecasting. However, very little research has yet been undertaken to compare and contrast various deep learning algorithms within the application of blockchain-based FTS forecasting.

The last main difficulty of this report was finding a reasonable trade-off while exploiting confidence intervals. In fact, the higher the threshold, the better the overall performance, but the lower the overall effected trades. Further research could focus on optimising the trading volume to reflect confidence intervals for a better trade-off.

As highlighted throughout this thesis, machine learning is a field of great vastness. Such a wide range of models provides academia with the luxury of choice, but all models have both advantages and disadvantages. Further work on using DNNs as part of a larger ensemble learner could help exploit advantages of a variety of models without having to suffer from the disadvantages. Ensemble learning, such as the one used in Bayesian model combination, uses the outputs of multiple types of learning algorithms to help make a more informed prediction.

In addition, only a small range of features were examined here. Further research could attempt to use data mining techniques for large-scale feature selection. This would allow algorithms to exploit the full range of blockchain data provided by cryptocurrencies, which is their key differentiating factor from traditional fiat currencies and constitutes the principal opportunity offered by blockchain finance. This large-scale feature selection could also leverage natural language processing to allow forecasting algorithms to consider non-financial features such as social media hype, network traffic, cross-exchange arbitrage, GitHub commit comments, and much more. Indeed, such linguistic features have been repeatedly shown in the past to significantly impact the price of both cryptocurrencies and traditional stocks. This may also benefit multi-step prediction rather than the single-step prediction here.

## 5.3. Reflection

This bespoke Bachelor thesis successfully leveraged established machine learning concepts onto an application that has been previously unexplored until now. While operating at the cutting edge of research may have been an ambitious goal, significant planning, design alternatives and contingencies were made. This thorough focus on hard work and organisation allowed the project to not only meet its aim but also surpass it which reinforces any belief that this project was a success.

The literature reviewed thoroughly encompassed decades of prominent academic discoveries in the field of FTS forecasting. Around 30 pioneering books and academic papers were read and scrutinised, to be as informed as possible when developing a solution for cryptocurrency forecasting.

The theoretical work covered during this project was also extensive, with over 12 key FTS forecasting models identified and understood. The mathematical operations and theoretical assumptions of each model were critically examined and considered.

The practical work undertaken also covered a lot of ground. Over 200GiB of data was collected and compiled into quality datasets. All 12 forecasting models studied were implemented either from scratch or using industry-leading libraries such as TensorFlow. Their performance was then analysed and evaluated against each other. This allowed a thorough intuitive understanding how to build, train and deploy trading algorithms onto dominating cryptocurrency exchanges. This practical work also incorporated the learning of unexpected skills, especially when having to deal with challenges. The most significant issue being long training times which was adressed by learning about the CUDA parallel computing API and GPU-accelerated learning.

This project topic was initially chosen to help confirm or discard an ambition of further study into data science. Now that this thesis' goals are complete, the fascinating and vast field of machine learning has revealed itself to be the subject I desire the most to study further in the hope of ultimately being an asset to existing academic research.

This project provided a unique opportunity to study pioneering data science techniques and their application to FTS forecasting within the context of blockchain-based finance. With the help and support of Dr. Yin, an academic expert in neural networks, data mining and deep learning, this project was the most challenging, rewarding and enjoyable work that I have ever had the privilege to undertake.

## 6. References

[1] Bill & Milinda Gates Foundation. (2015, Nov. 12). *Level One Project* [Online]. Available: https://leveloneproject.org/

[2] United Nations. (2017, Jan. 1). *World Food Programme* [Online]. Available: http://innovation.wfp.org/project/building-blocks/

[3] CoinMarketCap. (2018, Mar. 30). *Cryptocurrency Market Capitalizations* [Online]. Available: https://coinmarketcap.com/

[4] Y. Demyanyk, O. Van Hemert, "Understanding the Subprime Mortgage Crisis," *The Review of Financial Studies,* vol. 24, issue 6, pp. 1848-1880, Dec 2008.

[5] F. Black, M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, vol. 81, issue 3, pp. 637-654, Jun 1973.

[6] NYSE. (2007, Mar. 30). *NYSE Group Equities Streamlining* [Online]. Available: https://www.nyse.com/publicdocs/nyse/markets/nyse/CCG_Notification_Update1.pdf

[7] A. Gamble, "British Politics and The Financial Crisis," *British Politics*, vol. 4, issue 4, pp. 450-462, Dec 2009.

[8] J. Murphy, *Technical Analysis of Financial Markets*. New York: New York Institute of Finance, 1999.

[9] E. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," *The Journal of Finance*, vol. 25, issue 2, pp. 383-417, May 1970.

[10] H. Yin, "Random Processes and Signal Modelling," in Advanced (Digital) Signal Processing, unpublished.

[11] J. D. Hamilton, *Time Series Analysis,* vol. 2. Princeton: Princeton University Press, 1994.

[12] G. Walker, "On Periodicity in Series of Related Terms," *Proceedings of the Royal Society of London*, vol. 131, pp. 518–532, Jul 1931.

[13] J. P. Burg, *A new analysis technique for time series data*. New York: IEEE Press, 1968.

[14] R. F. Engle, "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica,* vol. 50, issue 4, pp. 987–1007, Jul 1982.

[15] T. Bollerslev, “Generalized Autoregressive Conditional Heteroskedasticity,” *Journal of Econometrics*, vol. 31, issue 3, pp. 307-327, Apr 1986.

[16] R. F. Engle, V. K. Ng, "Measuring and testing the impact of news on volatility," *Journal of Finance*, vol. 48, issue 5, pp. 1749–1778, Dec 1993.

[17] E. F. S. Pierre, "Estimating EGARCH-M Models: Science or Art," *The Quarterly Review of Economics and Finance,* vol. 38, issue 2, pp. 167–180, Feb 1998.

[18] N. Hamzaoui, B. Regaieg, “The Glosten-Jagannathan-Runkle-Generalized Autoregressive Conditional Heteroscedastic approach to investigating the foreign exchange forward premium volatility,” *International Journal of Economics and Financial Issues*, vol. 6, issue 4, pp. 1608-1615, May 2016.

[19] L. Hentschel, "All in the family Nesting symmetric and asymmetric GARCH models," *Journal of Financial Economics*, vol. 39, issue 1, pp. 71–104, Sep 1995.

[20] S. Chen, S. A. Billings, “Representations of non-linear systems: the NARMAX model,” *International Journal of Control*, vol. 49, issue 3, pp. 1013-1032, Apr 2008.

[21] Y. Chauvin, D. E. Rumelhart, *Backpropagation: theory, architectures, and applications.* London, UK: Taylor & Francis, 1995.

[22] R. Lippmann, “An Introduction to computing with neural nets,” *IEEE ASSP Magazine*, vol. 4, issue 2, pp. 4-22, Apr 1987.

[23] N. B. Karayiannis, "Reformulated radial basis neural networks trained by gradient descent," *IEEE Transactions on Neural Networks,* vol. 10, issue 3, pp. 657-671, May 1999.

[24] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, vol. 43, issue 1, pp. 59 – 69, Jan 1982.

[25] C. M. Bishop, *Neural networks for pattern recognition.* Oxford, UK: Oxford University Press, 1995.

[26] G.F. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Transactions on Information Theory*, vol. 14, issue 1, pp. 55-63, Jan 1968.

[27] Y. LeCun *et al*., "Backpropagation applied to handwritten zip code recognition," *Neural Computation,* vol. 1, issue 4, pp. 541-551, Dec 1989.

[28] D. P. Kingma, J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980, Dec 2014.

[29] S. Nakamoto. (2008, Oct. 31). *Bitcoin A Peer-to-Peer Electronic Cash System* [Online]. Available: https://www.bitcoin.org/bitcoin.pdf

[30] Dash. (2015, Apr. 21). *Dash Whitepaper* [Online]. Available: https://github.com/dashpay/dash/wiki/Whitepaper

[31] A. Gervais *et al*., "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 3-16.

[32] A. Kiayias *et al*., "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*, Springer, Cham, 2017, pp. 357-388.

[33] Poloniex. (2018, Apr. 2). *Poloniex - Digital Asset Exchange* [Online]. Available: https://poloniex.com/

[34] I. Madan, et. al "Automated Bitcoin Trading via Machine Learning Algorithms," unpublished.

[35] S. McNally, "Predicting the price of Bitcoin using Machine Learning," M.S. thesis, Dpt. Data Analytics, National College of Ireland, Dublin, 2016.

[36] Z. Jiang, J. Liang, "Cryptocurrency Portfolio Management with Deep Reinforcement Learning," unpublished.

[37] A. Greaves, B. Au, "Using the Bitcoin Transaction Graph to Predict the Price of Bitcoin," *Stanford Network Analysis,* Dec 2015.

[38] H. Jang and J. Lee, "An Empirical Study on Modelling and Prediction of Bitcoin Prices With Bayesian Neural Networks Based on Blockchain Information," *IEEE Access*, vol. 6, pp. 5427-5437, Dec 2017.

[39] J. Chao *et al.*, "Forecasting exchange rate with deep belief networks," in *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, 2011, pp. 1259-1266.

[40] F. Shen *et al.*, "Forecasting exchange rate using deep belief networks and conjugate gradient method," *Neurocomputing*, vol. 167, pp. 243-253, Nov 2015.

[41] T, Kuremoto *et al*., "Time series forecasting using a deep belief network with restricted Boltzmann machines," *Neurocomputing,* vol. 137, pp. 47-56, Aug 2014.

[42] T. Hollis. (2017, Oct. 16). *Bachelor-Thesis* [Online]. Available: https://www.github.com/PsiPhiTheta/Bachelor-Thesis

[43] HistData. (2018, Apr. 5). *Free Forex Historical Data* [Online]. Available: http://www.histdata.com/

[44] N. Frisiani. (2018, Jan. 2). *HFCrypto* [Online]. Available: https://github.com/NFrisiani/HFCrypto

[45] Google Cloud Platform. (2018, Mar. 1). *Google Cloud Compute Engine* [Online]. Available: https://cloud.google.com/compute/

[46] R Documentation. (2018, Apr. 18). *fGarch* [Online]. Available: https://cran.r-project.org/web/packages/fGarch/fGarch.pdf

[47] R Documentation. (2018, Apr. 7). *tsDyn* [Online]. Available: https://cran.r-project.org/web/packages/tsDyn/vignettes/tsDyn.pdf

[48] R Documentation. (2018, Apr. 20). *MSBVAR* [Online]. Available: https://www.rdocumentation.org/packages/MSBVAR/versions/0.9-2/topics/granger.test

[49] Yann LeCun. (2013, May 14). *The MNIST Database of handwritten digits* [Online]. Available: http://yann.lecun.com/exdb/mnist/

[50] R Documentation. (2018, Apr. 10). *neuralnet* [Online]. Available: https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

[51] T. Hollis. (2017, Aug. 6). Machine Learning Algos [Online]. Available: https://www.github.com/PsiPhiTheta/Machine-Learning-Algos

[52] TensorFlow. (2018, Feb. 18). *An open-source machine learning framework for everyone* [Online]. Available: https://www.tensorflow.org/

[53] Keras. (2018, Feb. 18). *Keras: The Python Deep Learning Library* [Online]. Available: https://keras.io/

[54] NVIDIA CuDNN (2018, Feb. 21). *GPU Accelerated Deep Learning* [Online]. Available: https://developer.nvidia.com/cudnn

# 7. Acknowledgements

I would like to first and foremost express my sincere gratitude to Dr Hujun Yin for his interest, dedication and great help throughout the project's lifecycle. His insights and vast knowledge in the field of financial time series modelling and prediction were instrumental in guiding the scope of this project to one that was truly captivating.

I would also like to thank the School of Electrical and Electronic Engineering for allowing this Bespoke project to be accepted as the basis for a Bachelor Thesis. Their open mindedness, encouragement and expertise in pioneering industry projects of all fields were of great help, without which the project would not have been able to go as far as it did.

Last but not least, I would like to express my most humble thanks to my friends, family and all members of staff both at the University of Manchester and at prior education establishments as it is by benefitting from the hard work and advice of others that I could give this thesis my utmost.