Maynoth University

Final year project
Computer Science and Software Engineering

Evaluating Generative Adversarial Networks for Time series forecasting of Bitcoin Price

Benedict McGovern

Supervised by
Dr. Phil Maguire

April 9, 2019

Abstract

Artificial Intelligence (AI) has begun to reshape many aspects of our society as a whole. One area that has been greatly affected by AI has been the financial sector, notably in the area of predictive price forecasting, with the use of Machine Learning. First released in 2009 Bitcoin, is the best know cryptocurrency which has captured the public attention and is used in many global trading activities and has become a major financial asset. Its exchange rate with the major global currencies has also been highly volatility it reached a peak value of over €17,700 in 2017. However, it is also considered to be separated from many of national and political issues that influence the volatility of other major currencies such as Euro, Dollar and Sterling. Hence predicting its future asset value may be less problematic these currencies.

The purpose of this project is to compare current machine learning processes to the new wave of adversarial learning. In order to assess this, two predictive models were created for the function of timeseries forecasting for Bitcoin price prediction. The models chosen was a popular and high performance model, LSTM and a newer, Generative Adversarial Network (GAN). Generative Adversarial Networks Models have gained attention in image generation due to their ability to replicate data of high quality based on limited input data.

For this study the models were trained on a 10,000 hour sample of historical price data. They we then tested on a smaller set of unseen data of 4000 hours to compare performance. The main predictive performance indicator selected to assess the trading profile of both models was Root Mean Squared Error (RMSE). The LSTM model gave a RMSE value of 0.0078 while GAN model RMSE value was 0.107. Therefore, as constructed for this project, the LSTM out preformed the GAN model in its predictive capacity. Analysis of the factors which contributed to this difference was not possible for this project. However, when used in conjunction with a simple trading system, both models were able to return a positive result with a return of 8.9% for the LSTM model and 8.3% for the GAN model.

Chapter 1

Introduction

1.1 Summary

Prediction systems and tools are central to a range of scientific and professional activities. The financial sector has been one of the leading users of predictive systems. There is a long history of predictive systems which have been developed with the aim to identify future asset and commodity prices. Predicting future prices of assets, over short, medium and longer term, as well as currency exchange rates is at the core of the financial world. From the nature of trading, being able to predict shifts in prices and exchange rates has been key to determining the right time to buy and sell any financial asset in order to yield of financial benefit.

Computational methods, analytical tools and measurement systems have always been a central feature of such predictive systems. The sophistication of these has grown with technological developments. The first computing systems developed by IBM were designed to assist in accounts management through enhanced data processing over shorter time periods. The intersection of research of both finance and computer science can be traced back to as early as the 1950s to the work of Nobel prize winner Harry Markowitz. Markowitz attempted to produce an optimal performance selection using mean-variable optimization. Since then extensive work has been done to create trading systems that utilise computers to analyse risk and execute trades in a matter of seconds. In recent years, the introduction of artificial intelligence into financial tools and trading systems have become increasingly popular.

Artificial Intelligence (AI) as an academic research topic was founded in 1956 with some early researchers involved with developing solutions for problems such as pattern recognition and path finding. However, today artificial intelligence is a more popular research topic, coming alongside the development of higher performance and efficient computation as well as the wide scale collection of data. Currently, the majority of stock and asset trading are done with AI computer aided systems. Roughly between 50% and 60% of trades made on the US equity market are done through AI algorithmic systems. Artificial intelligence is an aspect of computer science that is involved in the creation and training of self-learning systems that are seen as intelligent in a solution

finding aspect.

A subsection of AI, machine learning will be the main area of focus of this report. Machine learning is the name given to the area of computer science that is involved with allowing computers learn automatically given large amounts of explanatory data. Machine learning differs from AI in terms of computational learning by the application not having to be explicitly programmed. Instead an algorithm is used, one which could be applied to many different applications. A target is defined, and the algorithm will attempt to derive an optimal solution to explain the target by means of assigning weights to the input data. Machine learning has been applied to many different use cases including self-driving vehicles and image recognition.

This report will first review the development of computational trading system through the use of machine learning with a focus on Cryptocurrency markets. Cryptocurrency is the name given to digital assets which are designed to work as a medium of exchange or tender i.e. a currency. A main point of interest of these digital assets is the focus on creating encryption protocols embedded in the currency itself. This helps protect the currency from fraud and theft. Cryptocurrency is also based on a distributed ledger that keeps a public but anonymous ledger, which acts as a transparent transaction database for the currency. Bitcoin was the first released cryptocurrency over ten years ago under an anonymous alias known as Satoshi Nakamoto in 2009. When released in 2009, a single bitcoin had little to no value. A single bitcoin reached an all-time high value of over 17,700€ in December 2017. Bitcoin has remained the largest cryptocurrency since its release with a market cap of 61 billion euro in March 2019. Along with bitcoin there are an estimated 4000 alternatives to bitcoin each with their own set of pros and cons.

This report will review and evaluate different methods of computation and data science that have been popularly used for timeseries forecasting. Time series forecasting is the name given to the type of forecasting which takes time as an element into explaining the occurrence of events. The application in question for this report be in relation to the price of bitcoin. The benchmark will be set by a long short-term memory (LSTM) neural network as this is seen as the gold standard to predicting timeseries data in recent years. Then an experimental model will be made with a generative adversarial (GAN) network, a modern adversarial deep neural network introduced in 2011 by Goodfellow et al. [**?**]. A formula will be proposed on how GANs may be used for timeseries analysis of this nature. In the results section the models will be compared in various different ways to gain an understanding on their prediction accuracy as well as real world utility of the models.

1.2 Motivation

The motivation behind this report stems from the lack of research published on time series prediction utilising Generative Adversarial Networks (GAN). GANs have been shown to prove significant performance over creative applications such as image creation and domain switching,for example video creation with cycleGANs. GANs application for a forecast application with timeseries data are sparse. One paper has been published

on the topic of stock prediction with GAN [**?**], however to date no research papers have been identified on the topic of cryptocurrency forecasting with GANS. The motivation is to further the research of GANs by exploring how GANs performs as a predictive tool for cryptocurrency price prediction by comparison with the current leading predictive methods i.e LSTM models.

1.3 Problem Statement

The predictive capacity of complex and chaotic systems such as financial markets is impacted by the predictive models ability to take account and draw conclusions from internal and external influential factors. For financial systems the ecosystem in which trades are carried out is complex with an large number of variables which may influence an assets price. Many believe most price movement stems from crowd psychology to determine the value of assets. However, it is also believed that repeating trends emerge in trading markets. The aim is to capture these trends through looking at price data and other derived data and then transforming this data in to features to a machine learning model.

The problem statement is that, given a comprehensive set of features that capture the variability of bitcoins price, can a generative adversarial network be utilised to build a model of price forecasting that outperforms other machine learning methods, notably LSTM models. The model will predict in the hourly time price frame and the dataset will consists of hourly price data looking back 10,000 hours.

Chapter 2

Related Work

2.1 Predictive Forecast systems

Predictive forecasting is an essential component of todays world. For example weather, food production systems and societal planning for energy, transport and housing. These systems are used in combination with other professions. Weather forecasting is perhaps the most commonly understood application of predictive forecasting. A large component of weather forecasting come in the form of numerical prediction where models are created that aim to generalise current state of weather in order to predict the possibility of future weather events of occurring. This can be placed under the heading of quantitative forecasting. This type of forecasting is involved with using past data normally at set time periods i.e. one hour or one day, in combination with stochastic variables to provide estimates to future events. These can be analysed to provide a probability forecast around any particular outcome over periods of hours to days and weeks.

Traditionally economic forecasting was achieved by using statically driven models. These models can be referred to econometric models. These were based on analysis of correlations between economic activities and other factors. The aim was to identify and capture the statically relationship between various variables and a stochastic event. One such popular econometric model for timeseries data is ARIMA. ARIMA refers to auto regression integrated moving average. Auto regression is the process of determining the strength of variables based on their lagged self. The moving average component considers the dependency of an observation and the residual errors from a moving average applied to lagged observations.

A second type of model used in traditional forecasting are soft computation based models. These types of models include artificial neural networks, and algorithms such as support vector machines and random forests. These models aim to generalise a problem before being applied to a real world event however, in terms of financial forecasting this can be a difficult task due to the volatility of financial prices. In recent years the most popular soft computation approach is deep neural networks which, essentially makes use of multiple layers of neural network. In this paper McNally, the author used a recurrent neural network with Bayesian optimisation [**?**]. His network boasted a 6.37%

root mean square error (RMSE) which he compared against an ARIMA model with a similar training time with a RMSE of 53.74%. This showed a significant increase of forecast accuracy by using a deep neural network over the econometric, ARIMA model.

Genetic programming has become a prevalent strategy when it comes to financial predicting. Genetic programming is a subsection of artificial intelligence where a programme is altered over evolutions by a changing algorithm which is aided by a heuristic fitness function. In a 2011 paper Pei-Chann Chang wrote about using a genetic programming approach to detect turning points in financial time series data [**?**]. These turning points would aim to indicate buy and sell extrema points. The genetic algorithm would use piecewise linear representation (PLR) to determine turning points in historical and use profit to determine the fitness of a given point. The evolutionary programme was trained to determine an optimal threshold to signal a buy or sell decision. This intelligent PLR strategy for determining future turning points in timeseries data was successful with significant improvement over other strategies such as random walk, in terms of rate of return with stock options.

2.2 Long Short-Term Memory Models

Long short-term memory (LSTM) is a form of recurrent artificial neural network in the field of deep learning. The long-term memory refers to the networks ability to consider past data and its possible effect on explaining your target function. The target function refers to the output you are trying to predict. The short term refers to the networks ability to consider the most recent data input and learn its significance of explaining the target function. LSTMs major difference is that it handles a sequences of multiple vectors as an input rather than a singular vector of values. This is more akin to how human learning would occur. We for example in speech, take a full sentence of words into account to understand what is meant, not just the last word. The aim of the network is to appropriately assign weights to each vector in a sequence which optimally explain the target function. We can see how this type of network would be beneficial when undertaking a timeseries prediction as we want to learn how past events can be used to determine a future event.

LSTMs were introduced in 1997 by Sepp Hochreiter and Jrgen Schmidhuber [**?**]. One major improvement introduced two years later by Felix Gers [**?**], was the introduction of the forget gate. This was the way the LSTM cells tackle the vanishing gradient problem. The vanishing gradient problem is a problem associated with deep learning models. As you train multiple layers of a model, you must calculate an error or otherwise known as the loss function which signifies how well a sample predicted a target. This error is generally a number between 0.0 and 1.0 which is produced by a squashing function. The loss is moved backwards through the network which the earlier layers learn from and change their weights accordingly, this process is called backpropagation. The problem with this is that, if the loss is close to zero, when the loss is backpropagated the loss become smaller and smaller therefore vanishing. In the contrary, if the loss is close to 1.0 we have a similar problem called the exploding gradient problem. A LSTM cell tackles

this problem by making use of a forget gate. This gate is able to hold the loss of a past states, then current state then becomes a derivative of the backpropagated value and the past states. This eliminates the vanishing gradient problem by never reducing the loss to 0.0, or in terms of the exploding gradient problem, 1.0.

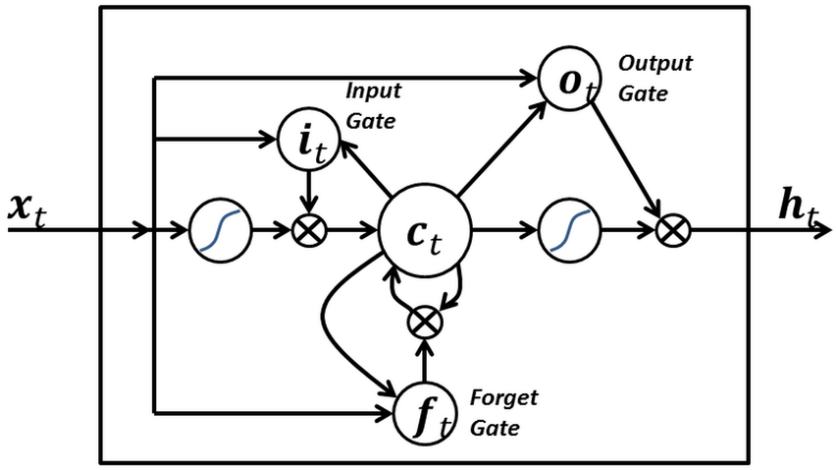


Figure 2.1: A LSTM cell showing the input, output and forget gate

2.3 Generative Adversarial Networks

Generative adversarial networks (GAN) are a relativity new approach to deep learning and were first introduced in 2014 by Ian Goodfellow et al.[**?**]. GANs are a form of adversarial machine learning which is a process where two models are trained simultaneously. The two models are in competition with each other in a zero sum game. GANS architecture consists of two models: a generative model G, with an aim to generate data according to a target data distribution, and a discriminator model D, with a single goal of distinguishing whether the inputted data came from the training set or from the generator model G. The process can be simplified to, the goal of the generator is to create data such that it is as close as possible to the real data. The discriminator then must decide if the data is real or fake, meaning is the data coming from the training set or has it been generated by a model.

The two models which are easiest implemented as multi-layer perceptrons, otherwise known as deep neural networks. They have a common value function which one agent is trying to maximise while the other is trying to minimise. This is different to the traditional deep learning approach which attempts to solve an optimisation problem. GANs are optimised when there is a saddle point that is at a minimum with respect to one agents strategy and a maximum with respect to the other agent. The discriminator acts as a classifier which classifies data as either real or fake. When it flags data as being fake it passes the loss, which signifies why the data was classified as fake, back to the generator through backpropagation. The generator then readjusts its weights

based on the discriminators loss which creates a better model capable of fooling the discriminator. The discriminator is commonly pretrained on a subset of the data. This means it has already learned how to generally classify the data, however the discriminator is continually trained throughout the GAN training process, making it a better classifier. Through the minimising and maximising of the models, we end up with a model G which is very good a recreating the data distribution of the real data. Due to the nature of the generation of new data that appears to be like real data, GANs have been increasingly popular in the computation creativity field, most notably image and video generation.

Much has come from generative adversarial networks. NVIDA recently published a paper which applied GANs to preform image generation to generate the human face [**?**]. The researchers used and proposed an alternative generator architecture which was built up progressively to first generalise the image and then add fine details though the addition of additional layers. The model was trained on the CelebA dataset, which contains over 202,000 face images and produced very realistic face images which can be seen below in the figure 2.2.

Figure 2.2: This is an example of the generated photorealistic faces created by a GAN created by Nvidia. All faces here are high resolution images were generated by a GAN in this experiment

Chapter 3

Method

In this chapter will outline the steps took to build and train models for bitcoin price prediction. The reasons and methodolgy behind decisions will be explained. Steps for creating both the LSTM model and GAN are both the same except in respect to training the algorithm GAN model. Both models will be trained on the same feature dataset created for this experiment. The models were trained on a sample 10,000 historical hourly bitcoin data.

3.1 Building the Dataset

The dataset for this project would require financial bitcoin market data. This data is commonly referred to as OHLC data, which is an abbreviation for open high low close. These referrer to the price of an asset at different intervals. This data for bitcoin can be extracted in a few different ways, most popular of this is with downloading directly from the exchange. For this project I decided to use the cryptocurrency exchange, Binance. Binance provided a rich API where historical data is available. This decision was made mainly because the high limits on the Binance API which is something of benefit when making large number of requests. Python was chosen to interact with the API as pythons package, pandas was a good option to manipulate and store the data, due to the scale of the data required. A program was made to download historical data which could be specified in different ways depending on what data was needed. This program could be specified to which time period would be downloaded i.e. minute data, hour data, etc. It also could be specified to how many time periods to look back i.e. 1,000 minutes, 100,000 minutes, this feature also queried if there was an already existing file under the same conditions in which case the program only downloaded the data needed to extend the file, so it would be up to date.

This program would handle the downloading and saving the dataset into a csv file. As the dataset would consist of many correlated assets that effect bitcoin price, the program would save each file separately. At a later section we will explain how these files were combined to produce a comprehensive dataset of bitcoin and correlated assets data.

A target function was then created at this time. This function would act as the predicted column when it came to the training of our model. As this model would be a price forecast, the target function would be a one-step lookahead of price. Given that we will be predicting in the time scale of hours, this means the target at a given step Xt would try and predict the price of bitcoin at time Xt+1.

3.2 Feature Creation

3.2.1 Technical Indicators

Technical indicators are generally mathematic formulas that describe the trend or movement over time of an asset with more accuracy. The simplest technical indicator would be a simple moving average formula which would be the average price of an asset in a rolling window fashion. Technical indicators are generally derived by the assets price or volume. Technical indicators have been used to understand price movements among investors and hedge fund managers for many decades. Technical indicators have been more recently introduced into machine learning models as they more accurately describe movements compared to raw OHLC data [**?**] [**?**].

The bellow figure 3.1 shows a series of graph each displaying different technical indicators for a range 400 hours of BTC/USDT data. Top being a mix of close, moving average and bollinger bands. Middle is the 6 Hour RSI. Bottom being the MACD signal and histogram.

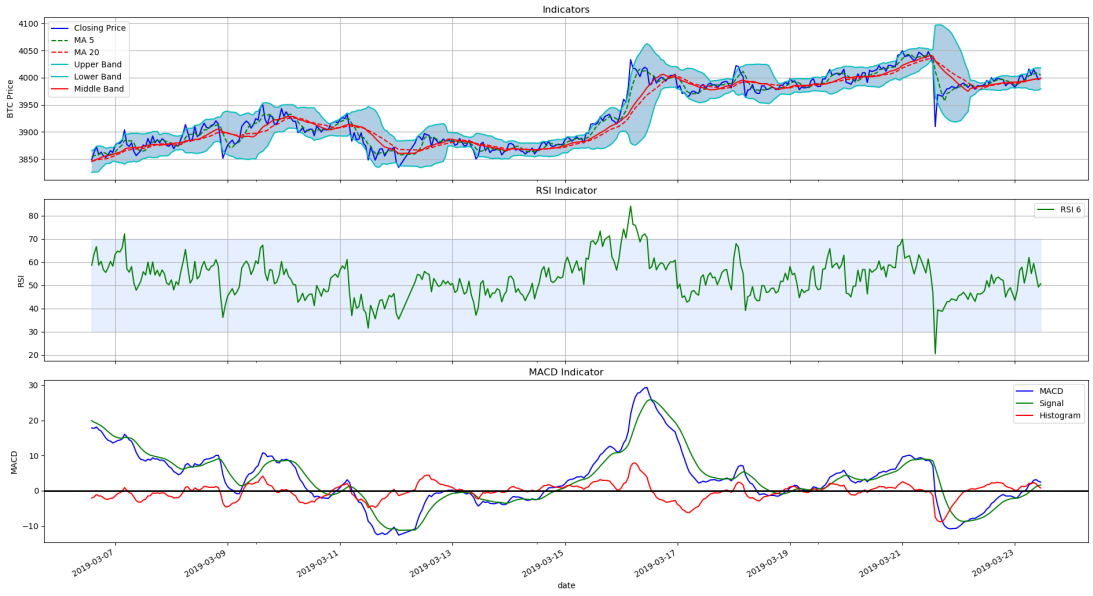


Figure 3.1: Three graphs showing various different technical indicators that will be used as features in the machine learning model

Technical indicators were used as features in this project. The indicators chosen were

Table 3.1: Technical Indicators

Opening Price	Close Price
High Price	Low Price
5 Hour Moving Average	10 Hour Moving Average
RSI 6	RSI 12
MACD	Williams Distribution
Bollinger Bands	Stochastic Indicator

popular indicators. Some of which chosen were the 10, 20 step moving average, MACD, Bollinger bands. The full list of indicators can be seen below. These indicators were created with the TA-Lib python package [**?**]. TA-lib is an open source python package which holds over 200 technical indicators formulas. TA-lib was easily introduced in the existing code as it natively reads and returns pandas dataframes. A program was written to combine the indicators chosen. The programme reads in the dataset file which was described above and programmatically created a new column in the dataframe for each indicator. This dataframe was then complied and saved into a csv file. This file would act as a master dataset for the machine learning models.

3.2.2 Fourier Transform for Trend Analysis

The Fourier transform created by Joseph Fourier in the year 1822 showed that some functions could be written as an infinite sum of harmonics. The Fourier transform attempts to decompose a function of time or signal into the frequencies that make it up. This is achieved by taking a function and creating a series of sine waves that when combined approximate the original function. By decomposing the signal, we can extract a derived wave that follows the original signal with less granularity although retains the overall trend of the signal.

Mathematically speaking the formula is as follows:

$$\hat{f}(x) = \int_{\infty}^{-\infty} f(x) e^{-2\pi i x E} dx \tag{3.1}$$

We can use Fourier transform to explain the local trends of bitcoins price movement. The function can approximate the long and short term trends depending on the number of components used in the transform. We can use these trends as features for our machine learning model with a goal of allowing the model to learn from the trends in order to predict future price. We can see from figure 3.2 that the lower component transformations such as the 3 component do a very good job an determining the overall trend of the price movement. On the opposite end we can the higher component transformations almost follows the price movements exactly. In our scenario the lower component transformations will better help the model to understand price movements.

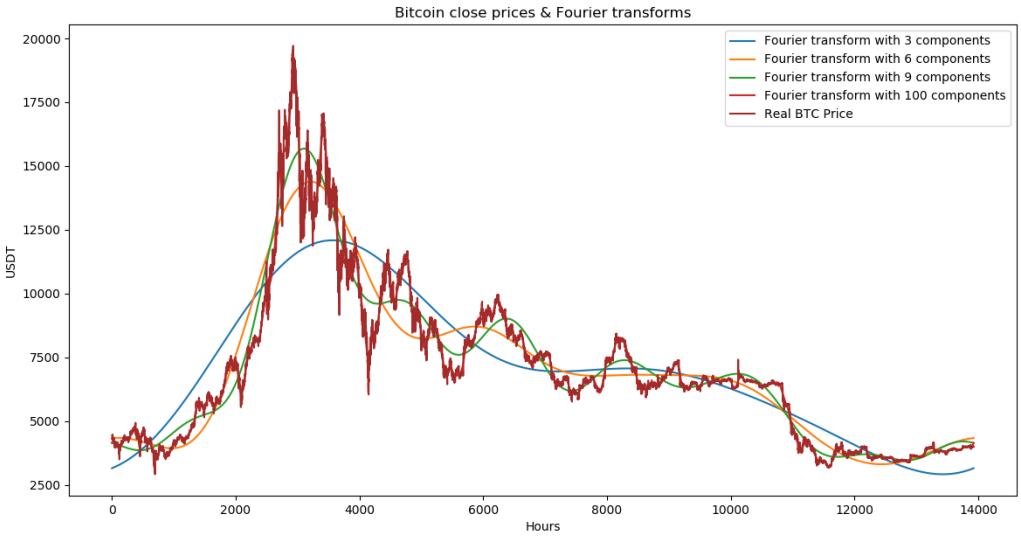


Figure 3.2: Fourier Transformations of multiple components plotted against Bitcoin price

3.3 Data Preprocessing

Data preprocessing is an essential step taken before any machine learning application is applied. This process generally involves normalising and scaling data of your dataset. This is important as machine learning algorithms benefit greatly from data which is consistent between all values in a dataset. For this application there was three main preprocessing steps carried out; null handling, standardisation, normalisation.

Null values occurred in several places throughout the dataset building process, although mainly in the creation of technical indicator values. For example, when creating a 7 step moving average (7MA) you must have at least seven previous values to calculate the first 7MA, therefore there is 7 null values at the start of this dataset. There were trimmed using pandas dropna function.

A normalising function was applied to all features of the dataset. The function was a percentage change function. This took the increase as a percentage between the last value in that column from the current value. The formula is as follows:

$$PercentChange = \frac{NewValue - OldValue}{OldValue} \tag{3.2}$$

Lastly a scaling formula was applied to the values of the dataset. This was applied to rescale the values in the dataset, so they would be in a range between 0, 1. This allows a neural network to interpret the values with less overhead. A MinMax scaling function was applied over the values. The scaled value, Xi is the difference between the original value of X and the minimum value of the dataset, maximum value minus the minimum

value.

$$Xi_{MinMax} = \frac{Xi - min(x)}{max(x) - mins(x)} \tag{3.3}$$

3.4 Feature Selection

Feature selection is a process where you select features for your prediction model which best explain your target function. The aim of feature selection is to reduce the attributes used in model creation to trim excess computation. These extra attributes will not have an impact on prediction performance. Feature selection will also give us an insight into which features have the largest significance on the prediction of the target. This can open up branches of analysis to better understand how your target can best be explained. In this section we will discuss two feature selection algorithms; XGBoost and random forest.

3.4.1 XGBoost

XGBoost is an open source library which provides a gradient boosting framework. The algorithm provides an edge over other tree boosting algorithms as it provides a more regularised model to control over-fitting. We trained a XGboost regressive model to calculate the importance of our features. The results can be seen as follows:

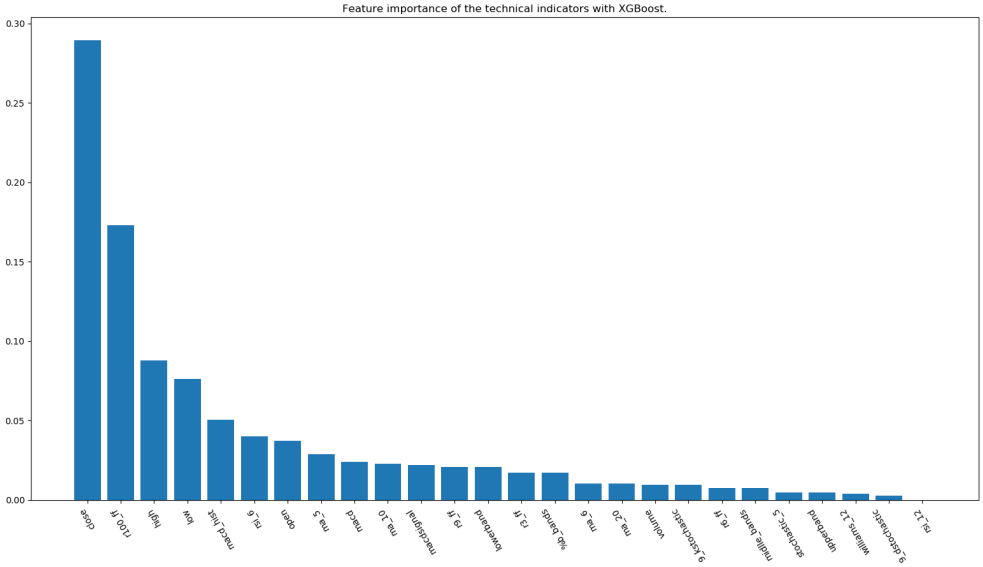


Figure 3.3: XGBoost results showing the *close* column has the highest significance an the following hours price. MACD is the best technical indicator predictor in this case

We can see from figure 3.3, that the *close* and *Fourier 100 component transformation* have the largest impact to the closing price of the next hour. We can conclude from this that the previous close price of bitcoin is closely related to the next close price. As the 100 component Fourier fit was almost modelled 1.1 to the bitcoin close price it makes sense that we see a similar significance here to our target. Other columns with high significance include the *high* and*low* price. We can rationalise this to conclude that the next close price is likely to be in the range of prices in the previous hour.

3.4.2 Random Forest

Random forest is a popular machine learning algorithm used for both classification and regression problems. It is popular for the ease it is to apply to problems and its often accurate results without the need of much hyperparameter tuning. It works by creating multiple decision trees which split training data into subsets based on a value of a single attribute. The ease of use and high performance nature of the random forest algorithm make it a good feature extraction algorithm. We applied random forest to our dataset of features and extracted feature importances to which the results can be seen as follows:

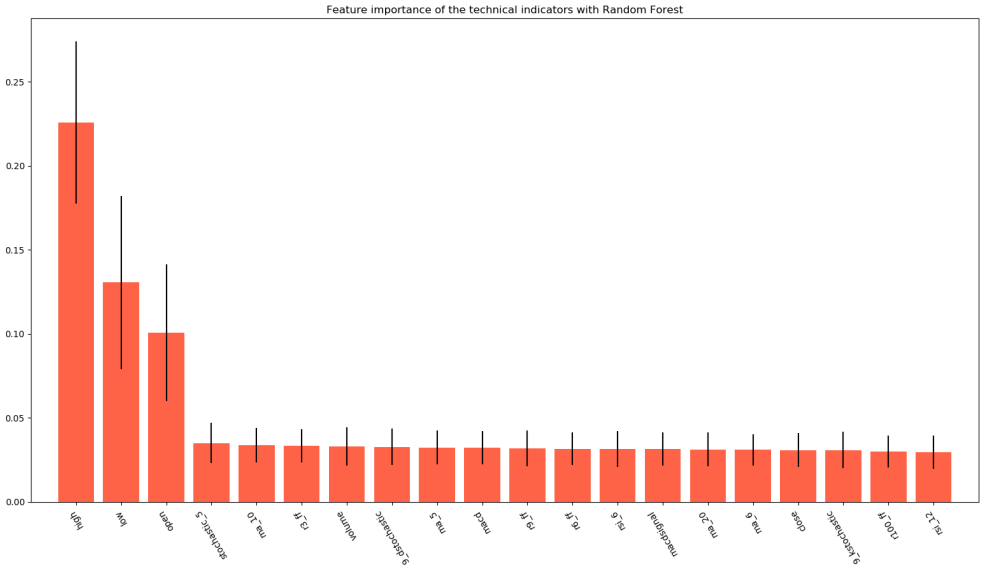


Figure 3.4: Random forest results showing the *high* column has the highest significance an the following hours price. Stochastic is the best technical indicator predictor in this case

We can see from figure 3.4, that the *high* and *low* have the largest impact here to the closing price of the next hour. This shows that the two algorithms returned different results of feature importance's however many of the features were presents in both algorithms top results.

3.5 LSTM Model Creation

As discussed above a Long Short Term Memory deep neural network will be used as a benchmark model to compare against in our results section. This is due to LSTM networks outputting high performance forecasts in the area of finance in recent years. Deep learning models are considered the gold standard of predictive analytics in modern days due to the availability of GPU accelerated training of models with larger banks of data. Keras, a deep learning library built on top of TensorFlow was used to build the LSTM model.

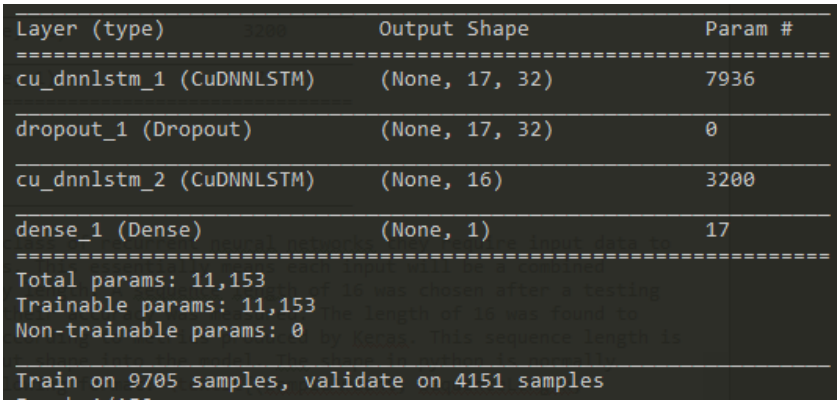


Figure 3.5: LSTM Model created for the benchmark test

As LSTM neural networks are a subclass of recurrent neural networks they require input data to be in the form of sequence vectors. This essentially means each input will be a combined sequence of inputs of an arbitrary length. A sequence length of 17 was chosen after a testing batch of models were trained and their accuracy was measured. The length of 16 was found to produce the most accurate model according to metrics produced by Keras. This sequence length is an important attribute of the input shape into the model. The shape in python is normally represented by a tuple of the following format: *(sampleAmount, sequenceLength, numberOfFeatures)*

The model consisted of LSTM, Batch Normalisation and Dropout layers. LSTM layers are layers made up of a series of LSTM cells which work as described above. The default activation of LSTM cells is hard-sigmoid. Batch normalisation applies a normalising function to the output of the previous layer. Dropout is a method to prevent overfitting which is a problem of machine learning where the model created to tailored too much to fit the training set. The model last layer is a dense layer which outputs one value which is the prediction price, y.

3.6 GAN Model Creation

The creation and training of the generative adversarial network for timeseries prediction was the most challenging part of this project. This is mainly down to the lack of research

and existing applications of applying GANs to forecasting timeseries data. The model was based on Xingyu Zhou et al. model described in his 2018 paper [**?**]. However, this paper did not discuss at length how a model was created it was some help to train a model for our application. Most applications online that make use of GANs are ones that preform image classification or image generation. These were helpful to understand how the training of a GAN was programmed however there was a major difference in the way no random noise would be used in our application. Instead noise data would be sampled data coming from the dataset we described above.

The idea of how a GAN could be applied for our application of bitcoin price prediction was one that required much thought. While it was inspired by Xingyu Zhous paper, their paper left out much detail on how such a GAN could be programmed and trained. Therefore, the decision on topics such as GAN topology, generator/discriminator network types, and training algorithm had to be thought out.

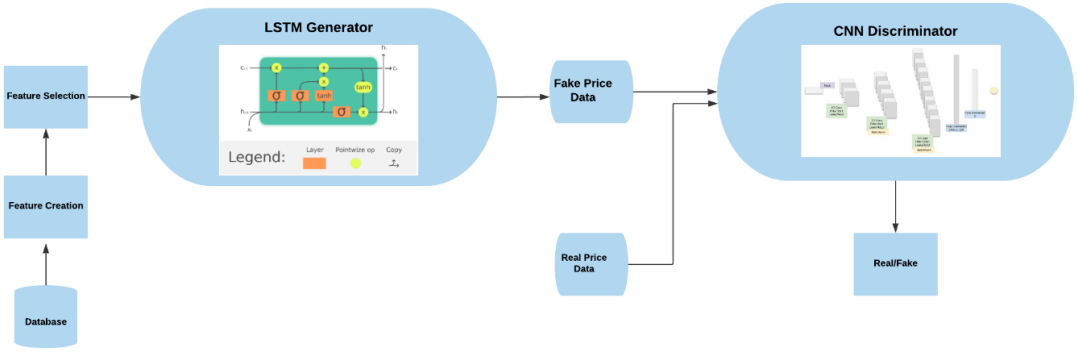


Figure 3.6: Illustrates the framework of the GAN. Shows the generated price alongside real price data being fed into the discriminator.

3.6.1 Wasserstein GAN

The Wasserstein GAN described by Martin Arjovsky in his 2017 paper [**?**] is an attempt to improve GANs performance through introducing an improved cost function which increases models ability to converge on a satisfiable result. The Wasserstein GAN is based on the Wasserstein metric which is defined by the minimum cost of transporting mass in converting data distribution q to the data distribution p. GANS were introduced with two common cost functions, the KL divergence and JS divergence. These functions rely on adding noise to smooth the gradients of samples with a high variance. The Wasserstein GAN applies a simple clipping to restrict the maximum gradient. Therefore, the exploding gradient problem is limited to a certain range controlled by the hyperparameters. The Wasserstein GAN cost formula can be defined as,

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

sup, is the clipping bound which is the 1-Lipschitz function following this constraint:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

As Wasserstein GANS have a reliable cost function the decision was made to use a Wasserstein GAN in our application to avoid any possible model collapse or non-divergence.

3.6.2 Generator

As explained above a generative adversarial network is made up of two neural networks working in an adversarial fashion against each other. The first component of this is the generator. The generator is responsible in outputting data samples which closely mimic the data distribution. A decision must be made when creating a GAN on which type of neural network to use. The main point to consider here is what type of neural network could most accurately generate data which matches the dataset data distribution. As we are building an application to timeseries forecasting a recurrent neural network would suit as they are built to capture patterns developing through time.

As a LSTM model was created in the above section for the benchmark test, we can base our model around the model we created above. The generator in this case would take sequence vectors of our training set, then preform a price forecast based on this data. The sequence length of 17 was kept the same benchmark model, as we previously had the best results with that length. The key difference in the generator model and our benchmark LSTM model was in terms of the output value of the model. As the generator model would need to feed its output into a second discriminator model, the output would need to be a sequence of values.

To make this change to the model we would need to output of the generator not be a singular value of forecasted price but a sequence of real price data with the forecasted price appended to the list. This was achieved by creating a model with two inputs, one being a one dimensional vector containing a series of real price data looking back 17 time periods prior to the forecasted price, the second being a sequence of vectors taken from the training set, just like the benchmark model. This was achieved by having the 17 values passed through a linear activation function then the forecast was appended to the end of the list. Therefore, the output was a sequence of 18 values, this output was reshaped into a 3x6 array. This was done as the discriminator network would only accept inputs of a two-dimensional shape.

3.6.3 Discriminator

A convolutional neural network (CNN) was used for the discriminator model for this application. A convolutional neural network is a form of deep neural network most commonly used in visual applications with either images or video data. We will be using CNNs as our discriminator network for their ability to extract local trends amongst global trends. They work by taking small samples of the input data, learning features from it and then through repetition learns to classify explanatory features of the input

data. CNNs have an ability to extract local trends to detect irregular price movements in the sequential bitcoin price data. These irregular trends would suggest the input data is fake when compared to real bitcoin price data.

The CNN model is made up of one dimensional convolutional layers which takes the input sequence and breaks it down into smaller sequences. Together with a corresponding layer known as a filter or kernel, matrix multiplication occurs to reduce the sequence into a more generalised format.

3.6.4 GAN Training

Various GAN models were trained for lengths of epochs ranging between 25 and 250. The discriminator was pretrained both on a batch of 32 real and generated values. This was done before each epoch to update the weights of the discriminator network before the passing inputs to the combined GAN model. The loss function was created to according to the Wasserstein loss function and was applied to the GAN model. All preprocessing steps and training set manipulation steps were taken before the GAN model was trained. Below in Algorithm 1 we can see the sudo code for the GAN training algorithm.

```
Preprocess_data()
while epoch in range do
    Pretrain Discriminator:
    while pretrain steps do
        Generator_predict()
        Discriminator.train_on_batch(real);
        Discriminator.train_on_batch(fake);
        Set Discriminator Weights;
    end
    Train GAN:
    Gan_predict()
    Update Weights;
end
```

Algorithm 1: GAN training algorithm

Chapter 4

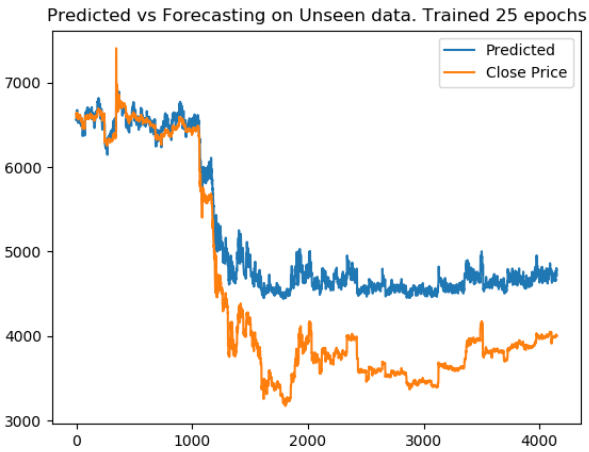
Results

In this section we will discuss the results and performance metrics which are generated from both the LSTM model and the GAN model. The models will be then used in a real life trading scenario. The validation set will be used for these results. This is data that remains unseen in the training of the model and consists of 30% of the total dataset. This data consists of around 4'000 hours of bitcoin data. The results will be unscaled and then plotted. The plotted results are the actual forecast against the predicted forecast. The Root Mean Square Error (RMSE) will be used a measure to validate predictive performance. RMSE is the standard deviation of the prediction errors. Multiple results will be shown for models trained for various lengths of time. These will be referred to as epochs, which is an iterative measure of time.

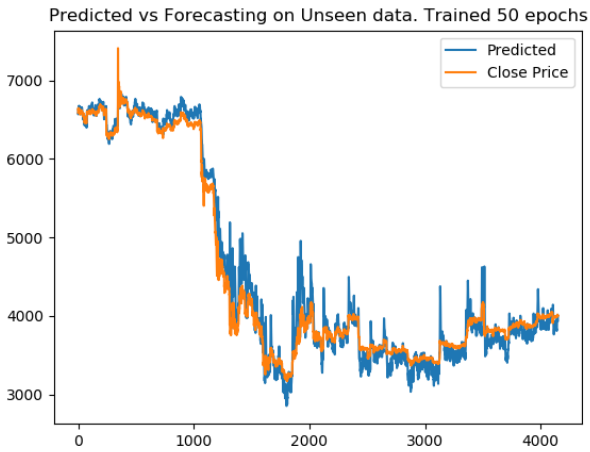
For the real life scenario, a simple trading strategy will be assumed. The aim was to apply a simple trading strategy to use the forecast pricing in a real world application. The simple trading strategy will perform a buy operation when the predicted price is higher than the current close price. The strategy would hold the position until either the forecasted price was lower than current close price or if the current close price was over 5% less then the price of the buy position. The buy positions will be indicated by a green triangle and a sell position being a red triangle. The strategy will assume a 500$ starting balance and will not consider trading fees.

4.1 LSTM Model

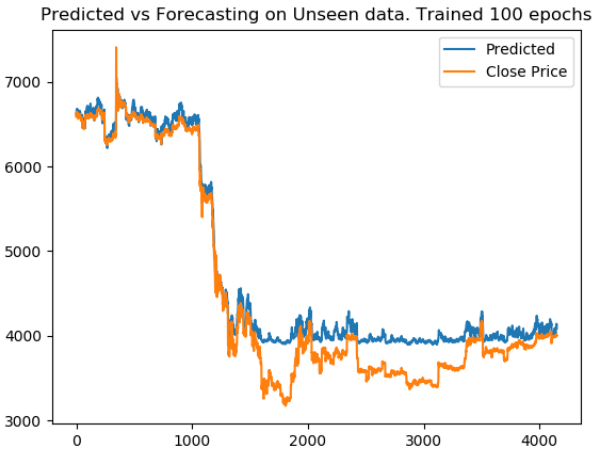
LSTM models were trained for a series of epochs as can be seen in figure 4.1. The epoch lengths were 25, 50, 100, and 150. Graphs were plotted of the predicted price against the actual close price. The data was unscaled before plotting meaning data was brought back to its original values. This makes it visually easier to comprehend. We can see that the model does significant better visually at following the actual price with a longer training time.



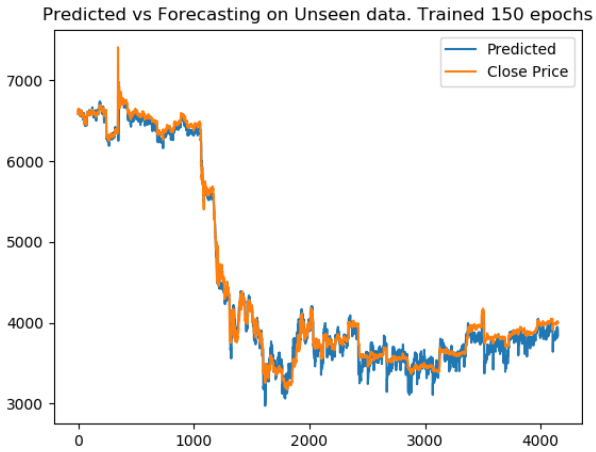
(a) 25 Epochs, RMSE of 46.26



(b) 50 Epochs, RMSE of 34.58



(c) 100 Epochs, RMSE of 47.10



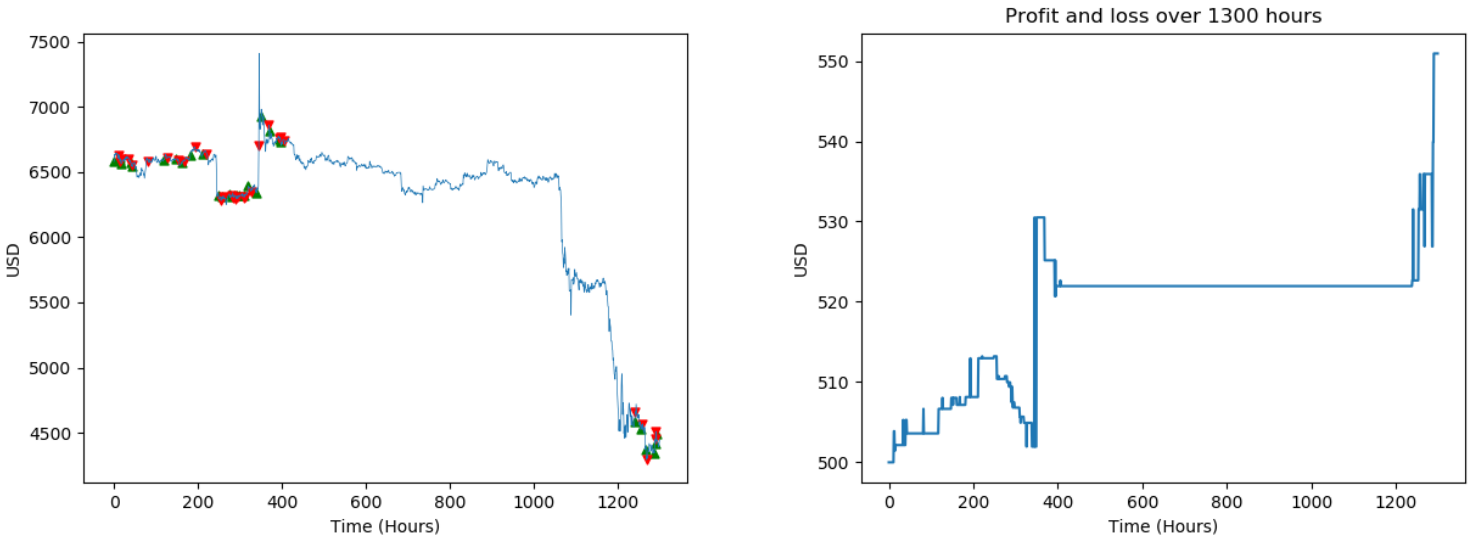
(d) 150 Epochs, RMSE of

Figure 4.1: Results of LSTM Networks

LSTM RMSE		
Epoch	Scaled	Unscaled
25	0.046	772.63
50	0.011	182.03
100	0.016	274.33
150	0.0078	130.81

4.1.1 Trading Strategy Results

We can see from figure 4.2 (a) that the strategy triggered many buy and sell positions in a volatile market seen in the first 400 hours. However, in a declining market between 400 and 1200 hours, the trading strategy did not place any buy positions. The strategy resulted in a profit of 44.64$ over 1300 hours of trading with a starting balance of 500$. This is an 8.9% return. We can see the in terms of profit and loss in figure 4.2 (b).



(a) Buy and sell graphs of trading strategy over 1300 hours

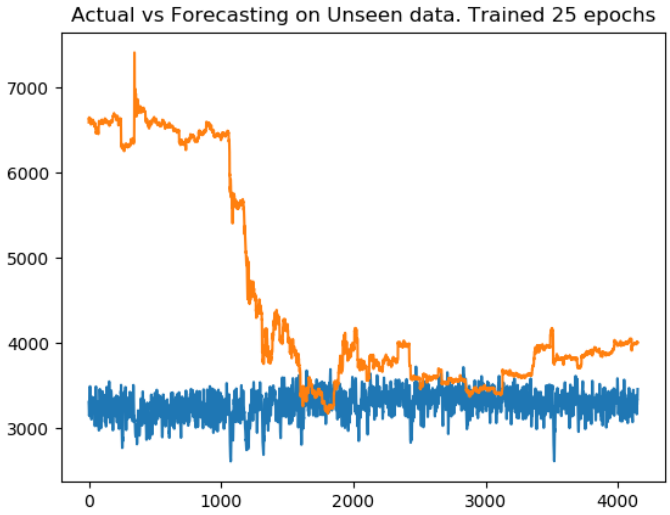
(b) Profit and loss graph of traing strategy

Figure 4.2: Results of LSTM Trading Strategy

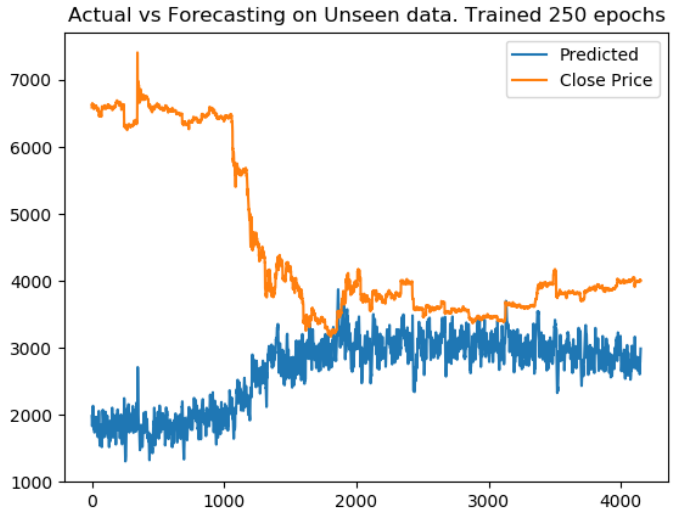
4.2 GAN Model

In this section we will look at the results of the GAN model that was created. As we saw from the comparing the 25 epoch and 150 epoch LSTM model, training time significantly improved the models performance, however such was not the case in the GAN model. Results from model trained for 25 epochs and 250 epochs had similar performance. The RMSE for multiple GAN models can be seen in the below table.

GAN RMSE		
Epoch	Scaled	Unscaled
25	0.107	772.63
50	0.107	78783198.55
100	0.107	2.22e+16
150	0.107	3.74e+20



(a) GAN model trained for 25 epochs

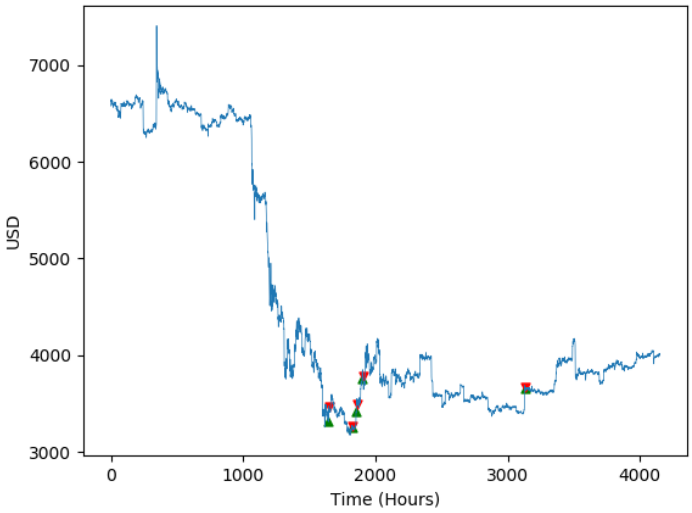


(b) GAN model trained for 250 epochs

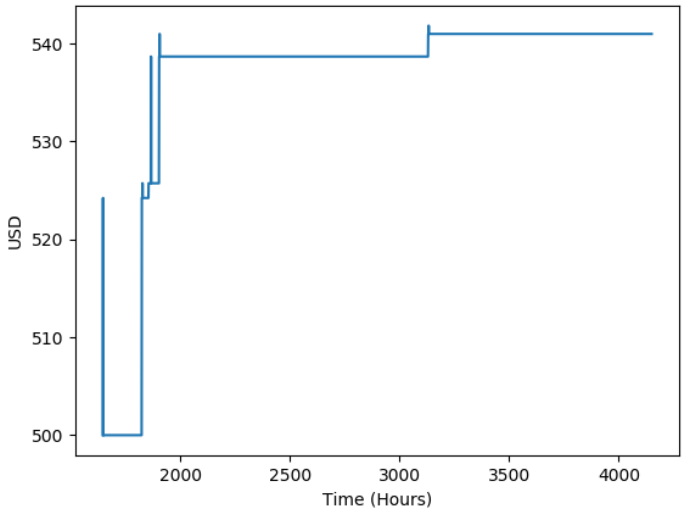
Figure 4.3: Actual vs Forecasted GAN model. Shows little difference in performance with longer training time

4.2.1 Trading Strategy Results

We can see from figure 4.4 (a) that the strategy triggered many buy and sell positions in a volatile market seen in and around the 1500 hour mark. The GAN trading strategy also avoided making trades in the declining market similar to the LSTM model. The strategy resulted in a profit of 41.64$ over 4000 hours of trading with a starting balance of 500$. This is a 8.3% return. We can see the results in terms of profit and loss in figure 4.4 (b).



(a) Buy and sell graphs of trading strategy over 4000 hours



(b) Profit and loss graph of traing strategy

Figure 4.4: Results of GAN Trading Strategy

Chapter 5

Conclusions

To conclude, in this report we proposed an experimental Generative Adversarial Network for time series forecasting. A dataset used for this experiment was created from historical hourly bitcoin price data. The data was transformed into a comprehensive dataset which targeted features that would be explanatory of bitcoin's price movements. We decomposed bitcoin's price data with Fourier's transform function with an aim to extract local trends in the ever volatile bitcoin market. Feature importance was applied to the dataset to locate features that may have a high significance on our target forecast. Preprocessing steps were applied to the dataset to reduce computational overhead when training models. The models were trained for variable lengths of time on our hourly feature dataset consisting of over 10,000 entries. The model was then tested on a validation set of unseen data of around 4,000 entries.

The results concluded that the LSTM network outperformed the GAN model in this experiment in terms of prediction accuracy. The LSTM networked showed an improvement in terms of prediction accuracy corresponding to longer training times, with a peak performance of RMSE 0.0078 on scaled data with a training time of 150 epochs. In contrast, the GAN model did not perform better with longer training time. A model trained for 25 epochs having the same RMSE of 0.107, of a model trained for 250 epochs. While the LSTM clearly had a better prediction accuracy, when models were applied to a simple trading strategy it was found the two models had similar success. The GAN model proved that it could predict direction of price movements. The LSTM model had an 8.9% rate of return will the GAN model had an 8.3% rate of return.

This shows that the GAN model could capture trends in direction of input data, however could not produce a result that in any way accurately predict the close price. This is shown by the very large RMSE results of the GAN models on unscaled data. The GAN model here could be improved in many ways to increase performance. One of which could be the generator and discriminator models. Each model comes with their own set of hyper parameters which may have had an effect on performance, these include batch size, layers, learning rate etc. With the use of Bayesian optimisation, hyper parameters could be improved in future experiments. In terms of the overall GAN model, many different configurations could have been tested to see if they had

an improvement on results. Parameters such as loss functions, learning rate, sequence length, etc are all important factors when creating a model. These parameters may have been the reason for the results shown above for the GAN model and would need more experimenting to have fully conclusive results. Testing all of these factors was a task out of scope of this report. Therefore, it is inconclusive if a Generative Adversarial Network could be an alternative model for timeseries prediction, however results show a promising indication the Generative Adversarial Network are able to detect trends in timeseries data. Results show, that LSTM networks remain a viable predictive method for timeseries data forecasting.

,