Honours Project (CCIS) Final Report
2017-2018
Submitted for the Degree of:
BSc Computer Games (Software Development)

Project Title: A Comparison of MLP, CNN and LSTM Variant Neural Networks for Modeling Stock Market Time Series Data
Programme : BSc Computer Games (Software Development)

Student : EE_1718

"Except where explicitly stated, all work in this report, including the appendices, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award"

Abstract

The world's stock markets are known to be a source of great wealth for anyone able to succeed in determining the future values of any given stock though the risk and volatility associated with this endeavor is often too great for simple guesswork and unrefined processes. Due to this Artificial Neural Networks have seen frequent usage in algorithms for predicting and modeling financial time series data however the scope of such networks is truly vast and it is not always clear what variation of network might be best suited to any given task. This paper looks to give insight into the suitability of three different architectures of neural network for predicting 5 different financial time series datasets: the feedforward MLP model, the Convolutional Neural Network model and the LSTM variant recurrent Neural Network model. Each of these models are created using the Keras API on top of the Tensorflow Neural Net framework, trained on 5 different sets of financial time series data over a range of 5 years for each dataset, optimised for highest possible accuracy by varying network variables and tested against unseen sections of each time series dataset. The paper takes an in depth look at the training process for each network and how each variable affects the final accuracy of each model. The final testing results are then evaluated and explained with references to similar studies of the effectiveness of different models showing the benefits and issues inherent in each model for predicting future financial time series data.

1. Introduction

This section of the report will give a background into the project field and describe the basis of the research and development objectives before proceeding into the main literature review.

1.1 Project Background

The world's stock markets have always been a source of great possibilities for those few who can accurately predict and manipulate the seemingly endless amount of data generated each day dictating each stock's value. Countless methods have been generated over the years to try to reduce the level of uncertainty inherent in trading stocks and shares from pure intuition to complex mathematical models aiming to predict future prices based off of previous data and certain "markers" indicating a possible up or down swing in the stocks market value. Each of these methods has come with varying success though as the years have went on these past models have needed to be redesigned and redeveloped as the nature of the stock market itself evolves and adapts. What works one day might not work the next as other traders take into account the methods and strategies employed by their rivals and use this information to generate models to exploit their rivals systems. One of the most recent attempts at predicting future stock values is a technology called Artificial Neural Networks.

Artificial Neural Networks are software algorithms that are designed in essence to predict a specific output or action based upon a set input or state. They are formed of multiple interlinked components called 'neurons'. Each of these neurons carries a weight that is applied to any given input with the combined weightings of each neuron contributing to the overall output function of the network. These weights are updated regularly through training in order to optimise the output function. Through training a network can theoretically be taught to model and mimic any system and due to this is a topic of focus in data science.

Artificial Neural Networks were first introduced in 1943 (Warren McCulloch et al., 1943) and since then have grown in popularity within the data science field. The first application of neural networks for prediction was introduced by Hu in 1964 for the purpose of weather prediction (M.J.C. Hu, 1964). This was later adapted by Kimmoto to be used to predict stocks on the Tokyo Stock Exchange in 1990 with a resultant accuracy of 63% (T. Kimmoto et al., 1990).

Since being introduced as a possible method of accurately predicting stock prices the popularity of using neural networks within a trading environment has increased year after year. This growth in popularity is partly down to the fact that they are inherently nonlinear in nature. This is a contrast to old prediction models such as autoregressive integrated moving average (ARIMA) that relies on the data series being linear in order to achieve satisfying results. It has been shown through studies that real world systems are normally nonlinear in nature (Zhang et al., 1998) thus giving the potential advantage to neural networks as a primary means of predicting stock market data sets.

Along with the benefit of neural networks being able to model nonlinear data sets they have other clear advantages to older methods of prediction. As neural networks are self adaptive and data driven they can learn complex functions with little or no prior input regarding the model of study. Their ability to learn without input ensures they can be easily updated and adapted with changes in the data trends.

Within the scope of artificial neural networks there exist various different types of network providing unique characteristics that can be applied to the forecasting of data. Various studies have been ran on the differing types of neural network and their application within time series forecasting.

White studied the application of a simple, feedforward artificial neural network (White, 1988). Feedforward neural networks are artificial neural networks in which the connections between the layers do not form a loop, i.e. the data is moved from input to output through the hidden layers and is unable to move backwards at any point. Feedforward neural networks have mostly seen usage in pattern recognition and are generally considered the simplest type of artificial neural network. Feedforward neural networks come in two main types: single-layer perceptron and multi-layer perceptron. Single-layer networks consist of a single layer of outputs and the inputs are fed directly into this output layer. Multi-layer networks consist of multiple computational or hidden layers connected in a feedforward fashion.

Other networks have been studied such as multi-step river flow forecasting (Atiya and Shaheen, 1999), stochastic model based neural network (Poli and Jones, 1994), convolutional neural network (Borovykh, 2017) and recurrent neural networks (Connor 1994). Each of these studies showed varying levels of accuracy for each type of network and each network seems to be suited to different aspects of the prediction algorithm.

Convolutional neural networks are a variation of feedforward mostly used for image recognition. The main difference of convolutional network is the convolution operator. This convolutional operators purpose is to extract interesting features from the data set. In cases of convolutional neural networks applied to time series data the main purpose is to identify certain indicators that the data might be moving up or down in relation to its previous state.

Recurrent neural networks can have signals moving in both directions through the network by looping the data any number of times within the network. This ability to loop data can provide higher accuracy at the cost of making the network significantly more complicated than its simple feedforward counterpart. The network learns from not only the data but also the past computation within the previous loops.

It is worth noting that combining multiple types of neural network can enhance the probability of accurate time series forecasting by utilising and combining each models unique capabilities. Zhang found that combining different neural network models can have a positive effect on the predictive capabilities of an algorithm when utilising models that are quite different to each other (Zhang, 2007).

Another approach to increase effectiveness in predicting accurate fluctuations in time series data is to combine both linear and nonlinear based approaches within the same algorithm (Taskaya and Casey, 2005). For instance combining previous working linear models such as ARIMA with feed forward neural networks could help to create an algorithm that fits both linear and nonlinear aspects of a time series data set.

One of the largest advancements to neural network accuracy has been the introduction of back propagation to feedforward networks to reduce the error within the trained network model to as low as possible. The idea was first introduced in 1969 (Bryson and Ho, 1969) and became widely appreciated in the late 1980s. The overall idea of the network is to regularly update the functions within the model by repeatedly comparing the predicted outcome with the actual outcome based upon past data. This theoretically is able to reduce the error within the algorithm to effectively zero. While in theory this can produce one hundred percent accurate results, in actuality the accuracy is less than perfect due to the time series data forever changing though despite this the process can still drastically improve performance.

As neural networks have shown to be most accurate at predicting high frequency time series data their main usage within the trading industry is currently what is known as HFT (High Frequency Trading). This accuracy is due to the increase in available information in which to train the network. In HFT companies typically employ algorithms to make thousands of trades per second monopolising on minute changes in each shares listed price. It is estimated that between 2009 - 2010 sixty to seventy percent of all US Stock Market trading was attributed to HFT (Shobhit Seth, 2015). This usage of high frequency data could be used to create models that are able to predict long term changes in time series data with potentially higher accuracy than longer term frequency data with the downside of drastically increased training times for each model and increased cost due to availability of training data.

1.2 Research question

Can more complex neural networks such as convolutional and recurrent models provide higher accuracy than that of a simple multi layer perceptron model when used to predict stock market share price values for low frequency trading ?

1.3 Project Outline

It is the purpose of this paper to measure the effectiveness of the above methods of time series prediction for predicting the prices of five different datasets: the Apple share price, the Google share price, the IBM share price, the Microsoft share price and the SAP share price. It is the hope that targeting five different datasets within the stock market that the all round effectiveness of each model will be tested and the possibility of a false result will be reduced. Predictions will be calculated for daily intervals to determine the accuracy for usage in long term or “swing” trading”.

The algorithm will consist of three artificial neural network models - mainly a simple multi layer perceptron model with back propagation, a convolutional model and a recurrent model (LSTM variant).

The models will be created and trained using the Python programming language in conjunction with the keras and tensorflow open source frameworks. Each model will be trained on the same sets of data sourced from Yahoo finance API via the pandas python library to ensure the results are as accurate as possible.

During the development process each model will be trained on the first 90% of time series data and will be tested on the last 10% of the data to optimise the network parameters. Once each network is fully optimised the models will be trained on the full set of time series data before predicting future share price values for each stock. They will then be tested by comparing the predicted share price values against the real share price values for daily intervals. The accuracy of each model will be tested using the mean squared error and percentage error based upon its real price.

It is worth noting that no real money will be used within this project and any profitability that could occur within these trades could be offset by the cost of making such trades.

1.4 Project Objectives

The objectives of the main literature review are:

1. Investigate the current prediction solutions employed for forecasting time series data.

    In order to fully investigate the effectiveness of the selected networks there must first be research into the methods currently used in forecasting time series data inside and outside of finance. This will involve looking at linear and nonlinear models along with the various types of neural networks available for time series forecasting.

2. Investigate the factors involved in determining a stock's market share value.

    To provide a more accurate result of the effectiveness of each network and what the network is actually predicting there will be research into the main factors influencing a particular stocks market share value such as company net worth and earnings per share. This data could be used to help train each model and will give greater understanding of the underlying data within the time series.

3. Investigate the implementation options of using neural networks for time series prediction.

    Lastly there will be research into the methods for implementing, training and testing neural networks for time series prediction. This will involve researching the various neural network frameworks available and looking into examples and tutorials for the development of neural networks with the purpose of predicting time series data within finance.

The main objectives of the project to be completed are:

1. Create a requirements specification for the application.

Documentation will be produced detailing each models specifications and requirements. This will include information on the type of network model, how it will be trained, the input parameters the expected output format and how each model will be tested in terms of accuracy and efficiency. This specification will also include details on the stocks being used within the algorithm.

2. Develop the neural network models.

Develop the training and prediction aspects of the neural network models along with the ARIMA linear model applied to each set of predictions. This will require different prediction methods for each prediction timeframe i.e. daily, weekly and monthly.

3. Train the neural network models on first 90% of historical data.

Import and process the raw market data for training on the neural networks before training and optimising each network model for the highest accuracy possible. During this training the mean squared error will be kept track of to adjust parameters in order to improve accuracy.

4. Predict the last 10% of historical data and compare against actual prices.

Run the prediction methods for the last 10% of the historical data to gauge accuracy for data the application has not seen. The error from this will then be used to back propagate each model for optimisation.

5. Predict share price data for future daily intervals.

Predictions will be made for each target stock at set intervals for each network model. The outputs will then be noted for comparison against future share price values from live markets.

6. Evaluate results based upon live market data.

Once predictions have been made and checked against live market share price data the accuracy will be calculated. Each model will then be compared across multiple factors to determine the best performing variant and its feasibility for live usage.

2. Literature review

This section of the report will detail the findings of each or the primary research objectives listed in section 1.4 under the main literature review objectives.

2.1 Investigate the current prediction solutions used for forecasting time series data.

Autoregressive moving average or ARIMA for short is one of the longest standing linear time series data forecasting algorithms and models are generally estimated using the Box-Jenkins approach (Box, Jenkins, 1970). The models typically take on the form of ARIMA(p,d,q) where p is the number of autoregressive terms, d is the number of non seasonal differences required for stationarity and q is the number of lagged forecast errors within the prediction equation (R. Nau, 2017). For more than 50 years ARIMA models have dominated time series forecasting (M. Khashei, M. Bijari, 2010) this is due to the flexibility and statistical properties inherent within ARIMA models. Their main drawback however is that the model assumes a linear data structure within the time series data set and this is not always the case.

Artificial Neural Network models on the other hand are a flexible nonlinear forecasting method studied in depth for their usage in predicting time series data (Zhang et al. 1998). One of the major advantages of using ANN's for time series prediction is that there is no need to specify what form the model will take as the model will form based upon the data itself. This advantage is useful for data sets that are abstract or nonlinear in nature as it can pick up features that might not be easily determinable using other models.
Artificial Neural Networks are information processing algorithms that have been inspired by the human nervous system and how it processes information. The key link between Artificial Neural Networks and the human nervous system is the structure of the algorithm which is built up of a number of interconnected processing elements called neurons that work in unison to solve a given problem (C. Stergiou, D. Siganos, 2017). They also learn by example the same way as the human brain so once a network has been trained it can recognise new data points it hasn't seen before and relate them to what it has been shown previously.

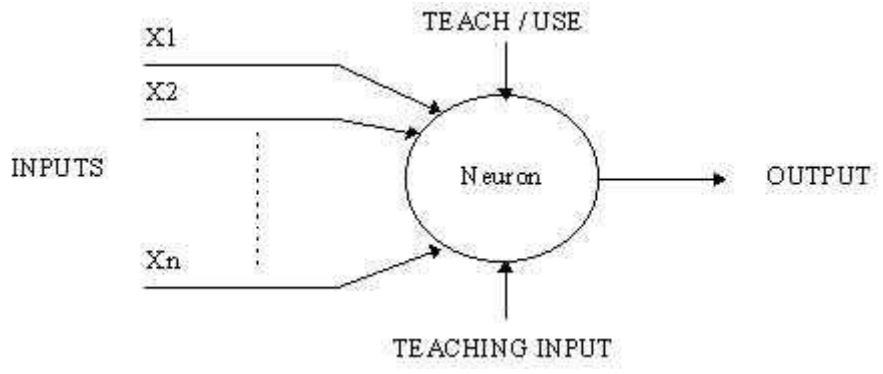


Fig. 1. A simple neuron (C. Stergiou, D. Siganos, 2017)

Figure 1 above shows a typical neuron setup within a simple feed-forward neural network. The model is designed to take in various inputs shown on the left hand side as x1, x2, …, xn. The neuron will then apply a weight function to each of the inputs (generally denoted w) in order to produce the desired output shown on the right hand side of the diagram. This process is achieved through training the network by passing in the desired output (see teaching input above) for each set of inputs. This process is repeated multiple times improving the weighting each iteration until the model is able to accurately predict the desired output on its own.

Artificial Neural Networks over the years have taken various different forms and come in a variety of different models each specialising in a specific aspect of data analysis. The simplest of these is generally considered a feedforward neural network. This model consists of an input layer, an output layer and at least one hidden layer that processes the information based on the learned function. The defining feature of a feedforward network is that all data within the network moves from input to output and is unable to move backwards at any point.

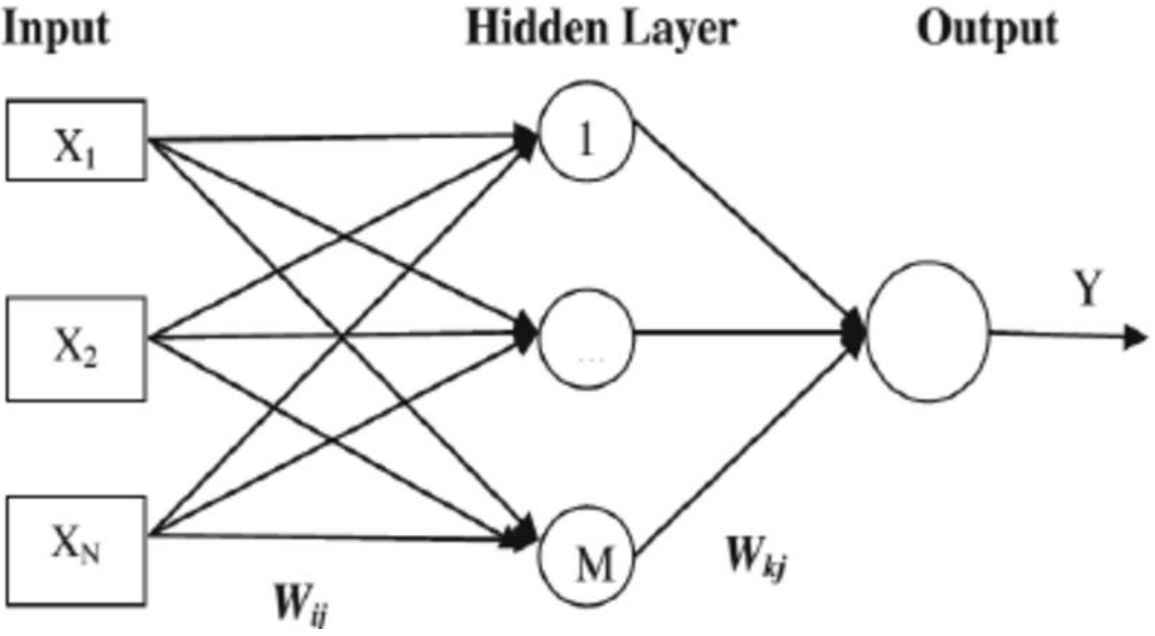


Fig. 2. A typical multi-layer feed-forward neural network architecture(A. Yilmaz et al, 2015)

The number of hidden layers can be increased as much as desired though the best number of hidden layers is usually a topic of debate in data science. George Cybenko stated that feed forward neural networks with one hidden layer is capable of universal approximation (g. Cybenko, 1989). Opposed to this several studies have shown the benefit of multiple hidden layers as opposed to the single hidden layer proposed by Cybenko, one such paper states deep networks can compute an output to the same accuracy as shallow models while requiring an exponentially lower number of training parameters (H. Mhaskar, 2016)

For any given network, assuming each input has a weight within the algorithm, the output of a feedforward neural network is determined by some function of the weighted sum of the input values multiplied by their weights.

$$y = f(v) \qquad where \qquad v = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

Where x indicates input and w indicates the weighting applied to the input within the neuron.

It is possible to increase the accuracy of the network by introducing back propagation. Back propagation is a method of reducing error within the algorithm by calculating the cost or error for each cycle of data moving through the network and updating the weights for each input based upon the derived error from the previous data cycle. This method is explored in the paper “Theory of the Backpropagation Neural Network” (R. Hecht-Nielsen, 1989). Using this method it is possible to reduce the cost function or error to effectively zero for any given data set.
The method for reducing the error within the network is defined by the following function with E being the combined network error.

$$\nabla E = (\partial E/ \partial w_1 , \partial E/ \partial w_2 , \ldots, \partial E/ \partial w_n).$$

With each weight in the network being updated incrementally.

$$\Delta w_i = -\gamma \, \partial E/ \partial w_i \, for \; i = 1, \ldots, n,$$

(R. Rojas, 1996)

Convolutional Neural Networks or CNN’s are a variation of feedforward neural networks that have seen usage mostly in analysing visual imagery though its usage does spread to applications such as language processing and even automated game AI agents. The main distinguishable feature from typical multilayer perceptron feedforward models is that the network is built up of layers of neurons in a two or three dimensional arrangement of filters as opposed to the two dimensional arrangement in typical feed forward networks. Within the network each neuron is only connected to a small selection of inputs and other neurons instead of being connected to each neuron in the other layers of the system.

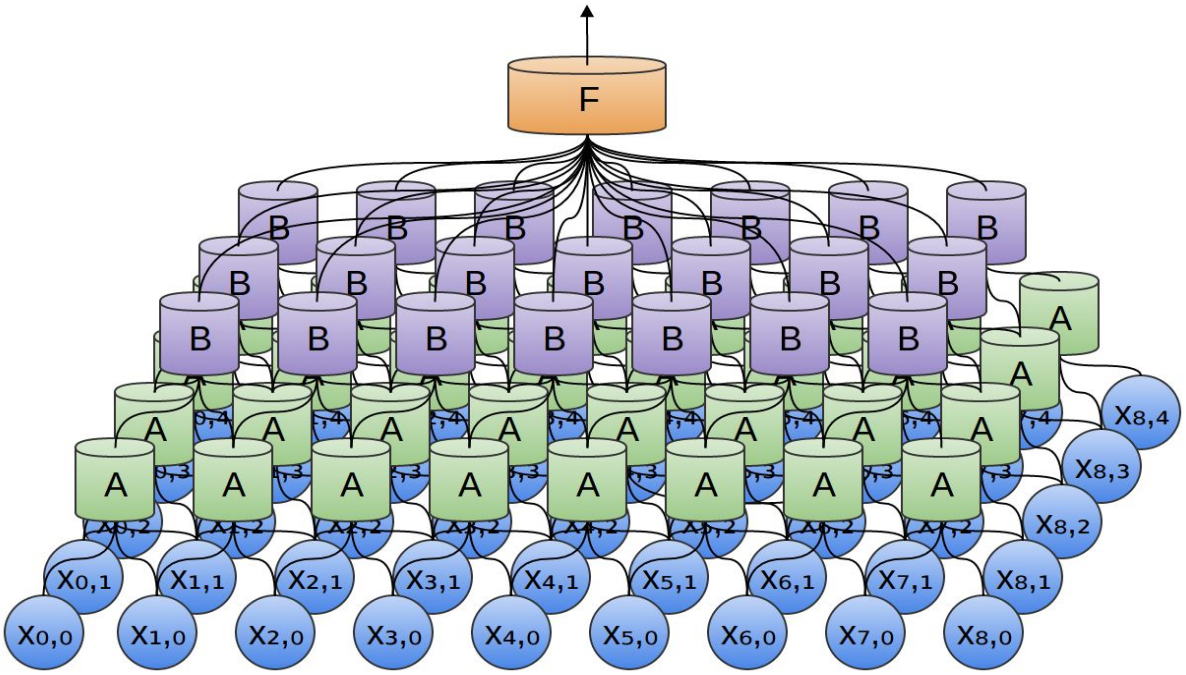


Fig. 3. An example convolutional network layout showing neurons with shared weights. C. Olah, 2014

Figure 3 shows a typical convolutional neural network layout. The network is structured with the inputs (indicated in the figure as the blue x's) being processed by only a small selection of the overall number of neurons for each layer. These neurons (sometimes referred to as filters) have shared weights within each layer. This means that each neuron in layer A will share weights with every other neuron in layer A and so forth. These outputs are then joined in the convolution layer (indicated with the orange F in the above figure) before finally being returned as an output.

While seemingly complex, CNN's can be simplified by thinking of them as networks built using layers of identical neurons with shared weights each looking at different aspects of the input data (C. Olah, 2014). This is particularly useful when dealing with large data sets such as images and video.

Although Convolutional Neural networks are primarily used for tasks such as image processing and recognition as shown in various papers such as ImageNet Classification with Deep Convolutional Neural Networks (A. Krizhevsky et al., 2017) and Face Recognition: A Convolutional Neural-Network Approach (S. Lawrence, 1997) there has been some study into its usage within time series forecasting. Wang and Yan discuss the usage of fully convolutional networks for classifying time series data in a comparison with MLP and residual variant models. The study shows promising findings regarding the accuracy of fully convolutional networks with the highest accuracy showing across 18 out of the 28 datasets (Z. Wang, W. Yan, 2016). Borovykh presents a study on the usage of deep convolutional WaveNet architecture for forecasting time series data and while the study concludes that the convolutional model is less accurate than a state of the art LSTM model there is however room for improvement on an already reasonable accuracy rating (A. Borovykh, 2017).

As with MLP's it is possible to introduce back propagation to reduce the error found within the CNN algorithm in much the same way as described for the previous simple feed forward example with the exception of weights being shared across multiple neurons.

Recurrent neural networks have recently become a popular method of time series forecasting with recent studies such as Speech Recognition With Deep Recurrent Neural Networks (A. Graves et al. 2013) in which the recurrent neural network proved to be a highly accurate method of predicting speech and is described as “state-of-the-art”.
At their simplest level the basic structure of a recurrent model is similar to a multi layer perceptron model in with inputs moving through multiple hidden layers to achieve an output based upon the weightings of each neuron within the network. The largest difference between feedforward and recurrent models is that while data for each input can only move one way through a feedforward model from input to output, a recurrent model will take into account the inputs that have came before it and update the neurons weights accordingly giving the network a form of memory (D. Britz 2015). A typical layout of a recurrent model is as shown below.

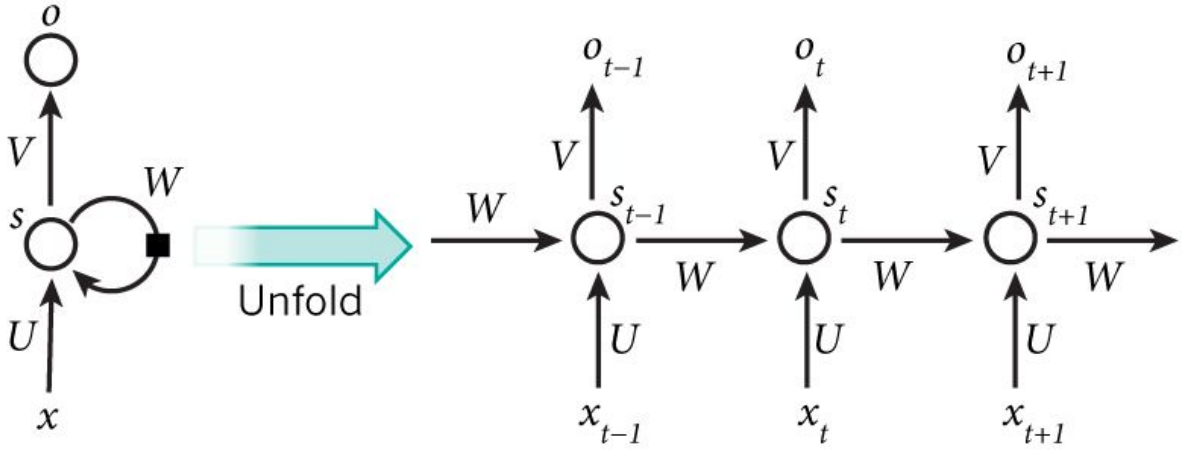


Fig. 4. An example Recurrent neural network. D. Britz, 2015

This memory that is inherent within recurrent network models allows for a more complete overview of the data that is being presented allowing for an output with a potentially higher accuracy than with no context in which to make its decision.

The output of a basic recurrent network is formed of various factors with the input vector ‘x’ represented as:

$x(t) = w_t + s(t-1)$ where s is the output of the context layers neurons (T. Mikolov, 2010).

Within the scope of recurrent neural networks there exist multiple variants with differing attributes and advantages to each with the most prominent of these being the Long short-term memory recurrent model (S. Hochreiter, J. Schmidhuber, 1997).
LSTM variant recurrent networks were developed as a model that bypasses the vanishing gradient problem (S. Hochreiter et al. 2001). The vanishing gradient problem as outlined in Hochreiter’s paper is the problem of error signals during neural network training either vanishing completely or blowing up exponentially when used with backpropagation and gradient based learning systems. This occurs for instance when the gradient used for

training the neural network becomes so small that each iteration makes a very low impact on the neurons weighting that the number of iterations required for meaningful improvements becomes an infeasible length (R. Kapur, 2016).
LSTM models avoid the vanishing gradient problem by introducing what is known as a LSTM unit or block. LSTM models can be made up of multiple blocks and each block comprises of an input gate, an output gate and a forget gate. These gates compute a linear function followed by a logistic function giving an output between 0 and 1 and allows control over the information retained and passed through the block (R. Grosse, 2017). Each gates value at any point will dictate the behaviour off the LSTM block. For instance an input gate value of 1 will pass the input value to the memory unit and a forget gate value of 1 will remember the previous value stored within the memory unit. This behaviour will lead to the values being combined when the forget gate is on or the value being overwritten if the forget gate is off when the input is passed into the block. This structure allows the LSTM variant to effectively avoid a vanishing or exploding gradient as the new hidden state of the neuron has an identity function with a derivative of 1 meaning the gradient will remain constant in place of vanishing or exploding (D. Britz, 2015).
The method of calculating the new hidden state for each neuron/LSTM block is as follows where U and V are input/output vectors respectively, g is the candidate hidden state and c_t is the internal memory of the LSTM unit:

$$
\begin{aligned}
i &= \grave{o}(x_t U^i + s_{t-1} W^i) \\
f &= \grave{o}(x_t U^f + s_{t-1} W^f) \\
o &= \grave{o}(x_t U^o + s_{t-1} W^o) \\
g &= tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} f + gi \\
s_t &= tanh(c_t) o
\end{aligned}
$$

(D. Britz, 2015)

2.2 Investigate the factors involved in determining a stock's market share value.

Due to the nature of the world's stock markets determining what factors directly influence a share's value is an incredibly difficult task though some factors have been seen to correlate with increases and decreases in a typical market share price.
Such factors include the earnings per share and P/E ratio (the ratio of price per share over the earnings per share) these two factors are described as fundamental factors for determining prices in an efficient market (D. Harper, 2017). The Price per share is simply the payment a shareholder would receive in dividends per share of stock owned through investment in the company. The P/E ratio is the price for any given share within a company over the earnings per share in the company. This figure then gives an estimation of how much money is required for each pound or dollar return. For instance take a company with a share price of £40 and an earnings per share of £2. The resultant P/E ratio would be as follows:

$$P/E = 40/2 = 20$$

Indicating a required investment of £20 for each pound of return in dividends.

A study on the Bahraini stock market listed 8 separate factors when determining a market share price of a stock. These factors include the return on equity, book value per share, earnings per share, dividend per share, dividend yield, price earnings, debt to total assets and the firm size (T. Sharif et al., 2015). A similar study into the market price of stock in Jordanian taka looked at factors including net asset value per share, dividend percentage, earnings per share, lending interest rate, inflation rate and gross domestic product (F. AL-Shubiri, 2010). Between these two studies there are some similarities in the factors listed though not all listed factors are the same across both studies. This hints at the idea that various stocks will have differing influencing factors based on market location, market type and economical situation with potentially no definite rule to cover all basis.

Another way to determine factors indicating a share prices future value is look at patterns and momentum. When looking at any time series data over a long enough period of time certain patterns are prone to develop that could potentially be used to indicate an upward or downward swing for instance a stock in a company that focuses on summer time equipment or services might see an increase in its stock price leading up to the summer season in anticipation for increased earnings. Momentum is also a potential indicator of a stock's future price and is discussed in the paper Profitability of Momentum Strategies: An Evaluation of Alternative Explanations (N. Jegadeesh, S. Titman, 2011).

For the purposes of this paper the algorithms will look at the past and present market share values only for each stock to forecast future market values due to availability of reliable data and to keep the networks as simple as possible to ease analysis between the variants.

2.3 Investigate the implementation options of using neural networks for time series prediction.

Neural network frameworks with their increase in usage and popularity come in various different options and are designed for the usage of various languages. Some of the most popular open-source frameworks in use currently are Theano, Torch, Caffe and TensorFlow.

Theano is an efficient and flexible library built from the ground up for the Python programming language. Using Theano it is possible to attain speeds rivaling C implementations for problems involving large amounts of data (Theano documentation, 2017). Using Theano a computational graph is defined for the network and the flow of information takes place once the graph has been compiled. This can pose problems as it is harder to debug information during run time. Another issue with Theano is the requirement to specify that the algorithm is to be processed on the GPU when other frameworks automatically compute on the GPU if available. It is also worth pointing out there is currently no support for Theano backend to utilise multiple GPUs when paired with APIs such as Keras.

Torch is a neural network framework with similar performance to Theano. The framework is built for the Lua language and due to this it lacks the standard data processing capabilities of Theano when used with the well known Python language. Recently an effort to create a Python oriented version of torch has seen improved popularity in the Torch library. While still in beta the library has seen usage and improvement from companies such as Nvidia and Facebook (PyTorch documentation, 2017). Due to this still being in early development the level of documentation when compared to frameworks such as TensorFlow or Theano is somewhat lacking. PyTorch does however fix the issues of data processing in original Torch and works in a similar fashion to numpy. For this project one of the more established frameworks will be used though future work implementing the algorithm in PyTorch could be an option.

Caffe is a neural network framework that has seen usage primarily in image processing. The framework is written for C++ though can be used in conjunction with a Python wrapper for easier prototyping (Caffe documentation, 2017). The advantages of using Caffe as a framework is the speed and usability for developing convolutional networks though its usability outside of convolutional networks drops significantly as it is primarily built for using CNNs for classifying images.

TensorFlow is an open-source neural network framework developed by researchers working on the Google Brain team and since its release as an open-source library it has seen increased usage within the community and is used in over 6000 open-source repositories online (TensorFlow Documentation, 2017). The framework has a flexible architecture that allows computation across multiple CPUs or GPUs within desktop, server or mobile devices through a single API. Due to the backing of Google in particular the continued support and development of TensorFlow is assured (Slant Community, 2017). One downside of using TensorFlow is that only some aspects of the framework have been made open source meaning less control than other fully open source frameworks.

Alongside the above frameworks there also stands the Keras Python deep learning library. Keras is a neural network API that is capable of running on top of TensorFlow or Theano with a focus on fast experimentation and prototyping (Keras Documentation, 2017). The main purpose of Keras is to allow for easier and faster prototyping than the standard Theano or TensorFlow libraries while keeping the efficiency and flexibility of each framework. It is due to this increased simplicity that it will be used for the purposes of this paper as an API running on top of the TensorFlow framework.

As mentioned most neural network frameworks are capable of running on both CPUs and GPUs when processing data. With the introduction of the CUDA parallel computing platform developed by Nvidia, processing large datasets using the GPU has become not only an extremely efficient but also simple option beating the performance of CPU units. This is due to the architecture of CPU and GPU modules.
CPU Units are primarily made up of a few cores optimised for sequential processing while GPU units consist of thousands of smaller cores in a parallel architecture designed for running multiple tasks simultaneously (Nvidia Accelerated Computing, 2017). Due to this increased efficiency the neural network models studied within this paper will be developed

and trained on a GPU utilising the Nvidia CUDA technology. This is a process that is made simple within Keras for as long as the machine training the network has CUDA installed in conjunction with an Nvidia CUDA enabled graphics card the TensorFlow backend will automatically enable GPU computing.

2.4 Hypothesis

It is my hypothesis that both the convolutional neural network and the LSTM variant recurrent neural network models will perform with a higher forecasting accuracy than the simple feedforward multilayer perceptron model. It is expected that out of the convolutional model and the recurrent model the recurrent model will have a the best overall accuracy due to its ability to resist the vanishing rule.

3. Methods

This section of the report will focus on the methods that will be used to fulfill the project topics.

3.1 Neural Network development

Development for the project will follow a prototyping strategy wherein an initial prototype for each model used within the study. The application will follow a class based object oriented structure to allow ease of use, readability and efficiency. External data files will be stored by and processed within the application. This will include the time series data in the form of a csv document which will be downloaded from the Yahoo! Finance API using the Pandas Python library (Pandas Documentation, 2017).

3.1.1. Data formatting.

The time series data will be formatted to ensure a uniform structure when it is passed into the algorithm for training. At this point in the application the data will be split into two parts with the first 90% of the data being stored in an array of initial training data and the last 10% of the data being stored in another array for use in testing the forecasting capability of each network during training.

3.1.2. Model creation

Each of the three neural network models will be instantiated and each parameter will be added accordingly. Using the Keras Python library each model will be instantiated using the Sequential() command. This will create a blank sequential neural network model with as an object within the application. Layers will then be added to the blank model to customise it in the way required with options including dense (simple feed forward), LSTM and convolutional layers.
An optional dropout layer will be added to each model to prevent overfitting. Overfitting is a problem inherent in complex neural networks where a model will optimise to perfectly fit the training dataset but when the model is tested against new data the accuracy becomes very

poor as the algorithm has overfitted itself to the training dataset including the noise within data and so does not generalise well (N.Buduma, 2015).
Dropout layers combat the effects of overfitting by removing neurons and their connections from the network during training and effectively training multiple smaller models in varying combinations of neurons from the original model. The results of each trained model are then combined with a new weighting associated with their dropout chance during training to fit the parameters originally set with a higher degree of accuracy. This increase in accuracy is due to the model behaving as a combination of models as model combination nearly always improves performance of machine learning methods (N. Srivastava, 2014).
Once each layer is added to the model object the model is then compiled with an optimiser and a specified loss function (in this instance mean squared error).

3.1.3. Training

After compilation the model is ready to begin training with the first 80% of the imported dataset. This can be done using the fit() function passing in the training data, number of epochs and batch size with each epoch indicating one pass over the entire dataset and batch size indicating the number of data samples being trained at any time in parallel. The number of epochs and batch size will initially be set at an estimated value for the first iteration of training each model before optimizing to determine the optimal parameters for the network. Once training is complete the evaluate function will be called passing in the last 20% of the time series dataset. This in turn will output the mean squared loss of the algorithm based upon the test data passed into the evaluate function. This will be used as a basis to compare future implementations of the model in order to optimise the input parameters to reduce the evaluated loss.

To optimise the algorithm certain criteria will be modified and tested against the MSE output from the initial prototype checking for any improvements in performance and accuracy. To achieve this the algorithm will iterate through the number of epochs and the batch size before returning a MSE loss output for each iteration.

The number of neurons will be kept as low as possible to reduce the chance of homogenisation resulting in the algorithm overfitting the data set.

The number of epochs will need to be fine-tuned through several iterations as accuracy improves up to a point before the algorithm starts to overfit the dataset, an early exit option will likely be placed once it is apparent the MSE is increasing with increased epoch size.

Batch size will be a compromise between accuracy and efficiency as larger batch sizes will give higher accuracy at the cost of drastically increased training times. This increase in training time can become exponentially larger when combined with an increased number of epochs so another early exit will be created to prevent extreme training times.

Once the most optimal MSE loss output is determined by iterating through the number of epochs and batch sizes the last 20% of the time series dataset will be used to further train each model. On completion of training the full dataset each model and its associated weightings will be saved to file for later use in forecasting. This model can then be loaded in

at any point or a new model can be created with the saved weightings by loading the weights from file after declaring a new sequential model.

This process will be repeated for each network model and for each time series dataset with each having its own unique file to ease usage in forecasting.

3.2 Time series forecasting

Once each model has been trained and optimised for accuracy a series of predictions will be made using the algorithm for daily intervals for each time series dataset that is being analysed. Each forecasted price will be recorded along with the net increase or decrease. This will allow the algorithm to be tested on accuracy to exact price and accuracy in predicting increasing and decreasing price.
The forecasting will be calculated using the predict() command for each model passing in the historical dataset and the required interval. This will produce an output in terms of share price for the specified time series dataset.

3.3 Forecast analysis

For each prediction forecasted by the algorithm a record will be kept and compared against future live market prices for each time series dataset. The results will be compared in terms of accuracy against the live price and accuracy in terms of predicting an increasing or decreasing price. This will be measured over each time period forecast and the resulting price accuracy will be calculated in mean square error.

3.4 conclusion and feasibility of each forecasting model

Once all data has been collated and analysed the results will be arranged in a series of graphs and tables to illustrate the accuracy of each model for each time series dataset. Utilising these graphs and tables an overall accuracy will be determined for each model across the range of time series datasets indicating which model gives the higher percentage accuracy.

3.5 Note on randomisation

It is worthwhile pointing out that during this paper the training of each neural network will be in some part random resulting in a varying output performance from the network model. This is due to the input vectors for each model being randomised to increase the network's overall performance as training on identical batches can lead to overfitting the dataset and the algorithm not being generalised. Weights are also initialised at random as initialising each neuron with the same weights will lead to all units in the hidden layer being the same.

4. Design and Implementation

This section of the report details the process undertaken when designing and implementing the algorithm that will be used for the predictions. This process includes the decisions made when creating and training each model and the reasoning behind each decision made.

4.1 Design Specification

In order to allow ease of use when training and testing the models on 5 large datasets an object oriented class based approach was decided upon to facilitate the sorting of the data and handling of each model through encapsulation.

4.1.1 Data Processing

The data being used within the algorithm consists of 5 csv files containing 5 years worth of open and close prices for each day during that time period. Due to the nature of this the algorithm first requires a reliable way to load and process this data to a format compatible with the neural network models used for prediction. In order to achieve this in a practical and reusable way a processing class is required containing functions that will deal with the opening of a desired csv file, loading the desired data into an array and sorting the data for processing in the specified model. This decision was made after looking at various online tutorials and examples.

4.1.2 Model Design

To facilitate the usage of three differing models within the algorithm to train and test 5 different data sets a class structure was chosen to be the optimal approach to reduce complexity during implementation. This would consist of three separate classes, one for each model. These model classes would need a way to initialise and build the model given a set of parameters along with separate functions for training and testing the model. An option to load and save the models would also be required in order to prevent the need for recreating and retraining each model when testing as the training process can take a substantial amount of time. The basis for these models were designed due to the influence of certain tutorials and examples with expansion and modifications made to ensure the models fitted the scope of the investigation.

4.1.3 Training Script

Alongside the model classes the algorithm requires a way to join the separate scripts together in order to train each of the models. This script will need to consist of a way to create 15 models (one of each type for each data set), import and process all 5 datasets, load and train each of the 15 models on the appropriate dataset and produce detailed graphs and statistics for the training process.

4.1.4 Testing Script

Similar to the training script, a script will be needed to facilitate testing of the neural network models across the 5 datasets used within the algorithm. This script will follow the same rough layout as the training script in that it should be able to load the datasets and the trained models when requested and output the results to an acceptable format to assist in the evaluation of the algorithm.

4.2 Implementation

With the design and flow of the algorithm established the next stage of development is the implementation stage. This was achieved using the class based structure previously outlined and unit testing regularly to ensure correct functionality. The development of both the processing class and the network model classes took inspiration from a tutorial found on the online website “Medium”. The tutorial can be found at the following link: https://medium.com/machine-learning-world/neural-networks-for-algorithmic-trading-part-one-simple-time-series-forecasting-f992daa1045a

4.2.1 Data Processing

As detailed above the first task in the implementation of the prediction algorithm is the ensure the data is accurately loaded, stored and split for testing and training. These functions take place within their own python class script called “process.py”. The class requires usage of the numpy and sklearn libraries to achieve the desired results and in this script 4 functions handle the loading and processing of the data for further use. These consist of load_close, split, shuffle and training_data. The load_close function simply opens the targeted csv file and reads all lines from 1 till the end of the file storing them in an array called stockFile. The function then iterates through each line of the csv file stored within the array storing the close price and close date as variables and appending them to their respective arrays. This is accomplished by splitting the lines into smaller chunks through the commas in the file and assigning the data at the expected location to the appropriate variable. For instance the close price is stored within the csv file as the 5th element in each line and therefor when split will appear as the element [4] in the array of that line (with the first element starting at element [0]). Once each line has been read with the appropriate data stored the function then returns the close prices and the associated dates as arrays.
Once completed the function was unit tested passing in each of the downloaded csv files to ensure accurate results.

The split function takes in a data array and attempts to split it into two arrays containing training data in one array and teaching data in the other. This is achieved by creating two arrays within the function and by iterating through the data array in amounts of the desired timestep. The training array is then appended with the value from which to predict and the teaching array is appended with the correct value of the stock for the predicted point in the future. Within this function there is also the availability to scale the array using the preprocessing sklearn library. This removes the trend from the dataset ensuring the model

focuses only on the relative delta movements in the stock as opposed to any long standing upwards or downwards trends.

The shuffle function works simply to shuffle the values of each array passed in to ensure the model does not overfit the dataset by reducing variance and making the model more generalised. This is achieved by randomising the order of elements within the arrays using a permutation of the array length.

The training_data function is used to create the arrays of data that will train and evaluate the training of each model. It takes in two arrays and a floating point value for a percentage and splits the arrays by this percentage. The goal of this is to get accurate results of training by training on the first 90% of the array and evaluating the model on the last 10% before moving on to full predictions.

4.2.2 Model Classes

With the data processing class completed and tested thoroughly the next stage of implementation is the creation of each neural network model class in order to instantiate and manipulate each model during training and testing. Each model used within the prediction algorithm is separated into its own class with each class containing functions for building the model, training the model, evaluating the model and both saving and loading the models associated weights.

Each model is built using the keras api on top of the tensorflow neural network framework. All three models contain the same functions and are largely built using the same structure with variations in certain aspects of the model.

Starting with the Multilayer Perceptron model the class is initialised with an input shape of [20,1] and an output size of 1 before proceeding to the buildModel function. This buildModel function facilitates the generation of the models layers and the compilation of the model. The model itself is built using the Sequential command to initialise a standard blank sequential model. Once created a series of layers and activation functions are added to the model. The model consists of a dense input layer with an input array dimension of 20 and an output size of 500, a dense hidden layer with input size of 500 and output size of 250 and a dense output layer with input size of 250 and output size of 1 where dense layers are simple feedforward fully interconnected layers. The reasoning behind the large input size of the hidden and output layer is due to a shallow networks ability for universal approximation (see previous literature review) when the breadth of the layer is sufficiently large. All three layers within the network use linear activation functions with the input and hidden layer being rectified. The reasoning behind this is two-fold. Linear activation functions do not suffer as greatly from the vanishing gradient problem previously discussed in the literature review section. Another beneficial aspect of the rectified linear (relu) activation function is that it is computationally inexpensive when compared to non linear activation functions such as sigmoid or tanh as only half of the neurons will be firing at once drastically reducing training and running time. This can be seen in the below figures comparing a typical tanh function with a relu function.

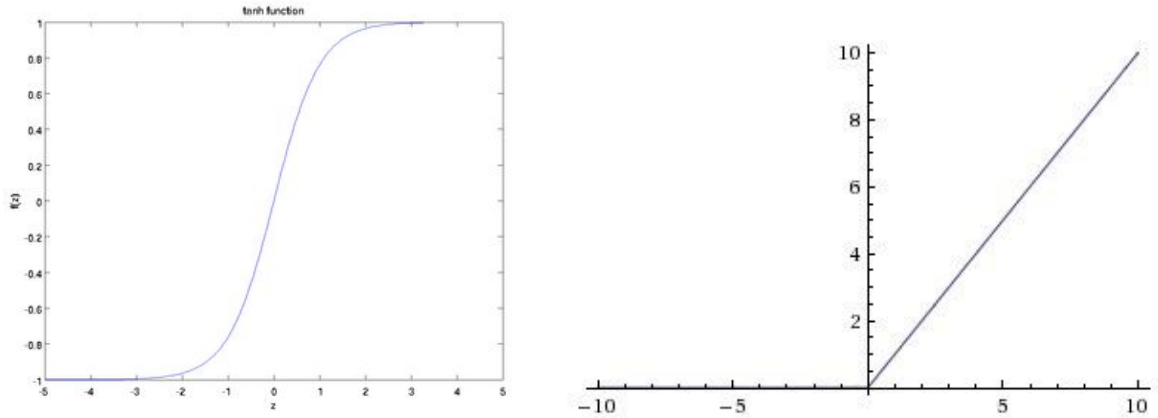


Fig.5 & Fig.6 indicating typical tanh and relu activation functions (L & R respectively) (A. Sharma, 2017)

The last layer is fully linear to ensure all neurons are firing when producing the output value for the network. Between the input and hidden layers a dropout layer is added to prevent overfitting. This is achieved by once again cutting the number of active neurons within any given pass of the system. The dropout layer in the MLP model is set to 0.25 indicating that 25% of the neurons in the hidden layer will be switched off for any given epoch of training. This helps the network become more generalised as the network cannot count on any given neuron being active forcing it to adapt and generalise better.

The model is then compiled using the Adaptive Moment Estimation (Adam) optimiser and finally returned as an object. Some benefits of the Adam optimiser is the optimisers resistance to vanishing gradients and the decaying average of past gradients working as a momentum like characteristic. Due to these characteristics Adam has been known to work well in practice and compares favourably to other adaptive learning-method algorithms (S. Ruder, 2016).

The trainModel Function operates to train the model to be able to predict a stock by showing it previous time series data for that stock. It takes in up to 4 variables with the first two mandatory variables consisting of the training data and the teaching data. The next two variables are for the number of epochs and the batch size and are set to default values within the model, this allows these variables to be optional when calling the function overriding the set values when provided while otherwise using the preset values. The function then uses these variables to train the model using the fit() command passing in the training data, the teaching data and the number of batches. Verbose is set to 1 to produce console output during training and the validation split is set to 0.1 to test accuracy during training. The validation split functions by taking, in this instance, the last 10% of the batch to test the training from the first 90% to determine the error.

The test Function within the class is utilised to evaluate the model outside of training. Similarly to the training function it takes in two arrays: an array of values to predict from and an array of real results. Evaluate is then called on the model passing the two arrays in as

variables with a batch size returning a mean squared error score regarding the models performance.

The last two functions within the class are the load and save functions. These functions when called load and save the current models weights to and from a file allowing a model to be reused between sessions removing the need to retrain the model each run of the algorithm.

The Convolutional model is set up largely the same way with an identical class layout. The main difference between the two models is found in the buildModel function. Within the buildModel function the model is again created using the Sequential command and adding the appropriate layers to the blank model. The layers are built up in this model with two convolutional layers followed by two dense layers. The two convolution layers take in an array of shape [20,1] across 64 filters with each filter having a length of 2 specified with kernel size. Padding is also set to be valid allowing any empty space in the filter to be padded with 0's. The convolutional layers strides are set to 1 to ensure the layer looks at each element couple within the array ie [0,1][1,2][2,3] as opposed to a stride of 2 with [0,1][2,3][4,5]. This is important to ensure the model has the highest accuracy possible by looking at every available element. Max pooling is also set for each convolutional layer with a pool size of 2. This effectively down samples the output storing only the highest result from each section looked at by the filter. An example of this process can be seen below.

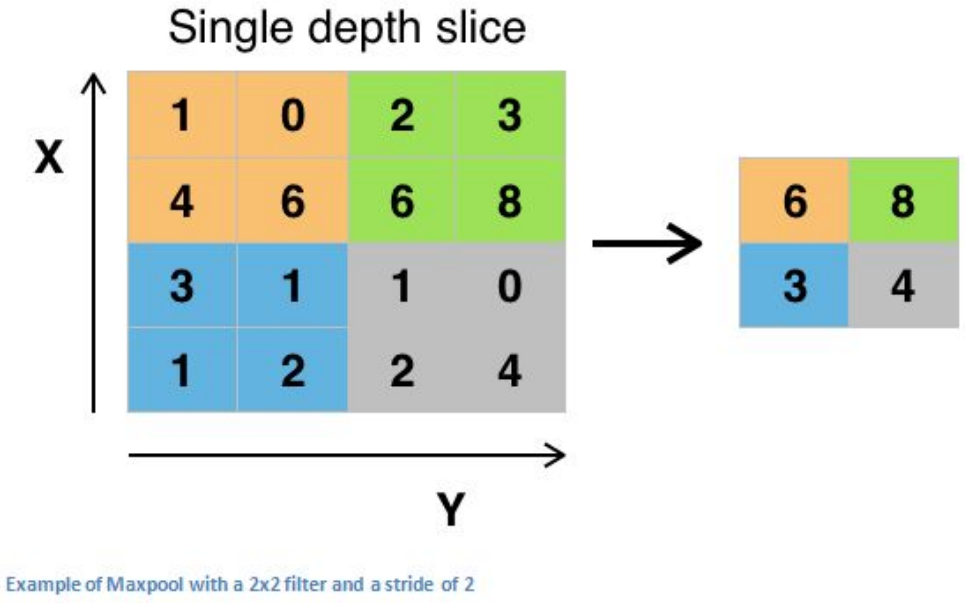


Fig. 7 Example of max pooling (A. Deshpande, 2016)

Figure 7 shows the process of applying max pooling using a 2x2 filter and a stride of 2 (as opposed to the stride of 1 used within this stock prediction algorithm). The filter is applied to each block of 4 elements on the left and takes the highest output from each storing them in a smaller array seen on the right.

Below these convolutional layers two dropout layers are added once again to improve generalisation for the network along with a call to flatten the output from the two convolutional layers before being fed to the dense interconnected layers. This is to allow the network to produce an appropriately sized output as opposed to the 2x2 pooled vector produced by the convolutional layers. The activation functions used for each layer are once again set to rectified linear and linear for the same reasons as previously.

Another difference between the MLP and Convolutional models is the input array dimensions. MLP models with dense layers are able to accept input with shape dimensions of 2 whereas convolutional layers require in an input shape with 3 dimensions. In order to adhere to this during training and testing of the model the training and teaching/evaluation arrays have their dimensions expanded on the axis value of 2 to ensure the arrays have an input shape dimension of 3. For instance given an array of shape (2,2) expanding the dimensions on axis 0 will result in a shape of (1,2,2), expanding the dimensions on axis 1 will result in a shape of (2,1,2) and so on.

Lastly the the LSTM model once again follows the same basic structure as the two previous models when it comes to its class structure. Much like the Convolutional model the LSTM variant recurrent model requires an input shape with 3 dimensions when training and testing therefore to accommodate this the dimensions of the training and testing arrays have been expanded similarly to the convolutional model on the 2nd axis of each array. Along with this the models layers once again differ from the previously described models. The LSTM model consists of 4 layers with 3 LSTM layers followed by a dense output layer. The first two LSTM layers are identical outputting vector sequences of dimension size 32 (these layers return sequences of vectors instead of singular vectors due to the return_sequences flag being set to true). The third LSTM layer receives a series of input vectors of dimension size 32 and returns a singular vector to the fully connected dense layer resulting in a singular output. Activation is once again linear for the dense layer with each LSTM not requiring a specified activation function due to the makeup of the LSTM blocks (see literature review). Once the layers have been added the model is compiled once again using the Adam optimiser before the model is returned as an object.

4.2.3 Training Script

With the model classes and data processing class completed the next stage of implementation is to combine the functionality within these scripts to train a set of 15 models (5 of each variation, one for each dataset). In order to achieve this each of the previous classes required imported along with matlplotlib pylab to produce graphical outputs for the training process. Once imported a series of lists are instantiated holding names for the various neural network models and the stock csv file names along with blank lists to hold the time series data and dates. 15 models are then created consecutively and stored within three lists (one list for each model type) and the training variables are set (training array dimension size, prediction timestep in multiples of days and the lag/time step between predictions). When each of the initial global variables have been initialised the time series list is populated by iterating through each of the csv files stored within the stock list and using

the process scripts load_close method appending the time series list with the result. Following this Training commences on each of the time series datasets.

Iterating through the time series list three empty local lists are created to hold the score of each model after training has been evaluated. The data sets are then processed and split into appropriate scaled training and test arrays. These training arrays are then used to train each model calling the trainModel function on each model within the three model lists with the score of each evaluation following and being appended to the relevant score lists. These score lists are then drawn to a graph for each dataset using pylabs's plot command with the graph being saved to file for reference.

4.2.4 Testing Script

The testing script is structured in a similar fashion to the training script in that it contains the same import requirements and setup variables. The testing script also requires the pandas package in order to output results to a csv file for interpretation. As stated the starting variables and lists are nearly identical to the training script however once each model is loaded in the testing script it is loaded from file using the models load function passing in the trained models file name. This is to prevent the need for retraining each model every time the testing script is ran. Two new lists are also instantiated with a list of csv file names in which to save each csv file and and array of arrays in which to store each of the 5 sets of results during testing.

Once the initial variables and lists are set up the stock data is loaded in an identical fashion to the training script wherein the time series list is populated by iterating through each of the 5 stock files and calling the load_close function.

Iterating through this list of stock data the data is once again processed and split into appropriate training and testing data while also generating an unscaled version of each in which to create complete results by scaling each models output to match the original unprocessed format.In order to do this an array of parameters is instantiated and for each time step in the unscaled testing the mean and standard deviation of the time step is recorded in the parameters array to allow for the scaled neural network outputs to be adjusted bringing them in line with real results. The scaled test data is then fed into each model using the predict command generating a result on the corresponding timestep. These values are then paired with their corresponding elements in the parameter array to adjust the stock price values to real values by multiplying each prediction by the standard deviation and adding the mean.

Following this an array is made up of the actual stock value at the predicted date along with each models adjusted prediction for that date. This array is then stored as a pandas dataframe and saved to a csv file. Along with the csv output a series of graphical representations of the predictions are also created against the actual stock value with both scaled and unscaled representations and saved to file.

4.3 Model Training

For the training process there are many variables that can affect the accuracy of the models prediction and each one requires fine tuned to determine the optimal settings for any given task. Two of the main variables when training a network are the number of epochs of the system and the batch size of the training samples. The number of epochs is how many times the training is ran over any set of batches, too few and the model won't be able to build an accurate enough picture of the time series and the predictions will be inaccurate, conversely if the number of epochs is too high the risk of overfitting the dataset increases and the model could suffer in performance when predicting previously unseen data points. The batch size variable also suffers a similar problem, too short a batch size and the model will have a less accurate picture of the gradient whereas too large a batch size can once again lead to the model overfitting the problem and becoming less generalised.

In order to determine the optimal number of epochs and correct batch size each of these variables were tested in iterative loops with the accuracy of each network being tested regularly to determine the best cut off point for each. Firstly each network was trained with a default batch size value of 128 elements per sample (this number was determined through previous experience in developing neural networks and through various online examples) and a variable epoch count starting at 10, increasing by 10 each iteration and ending at 200. This training produced the following results in accuracy.

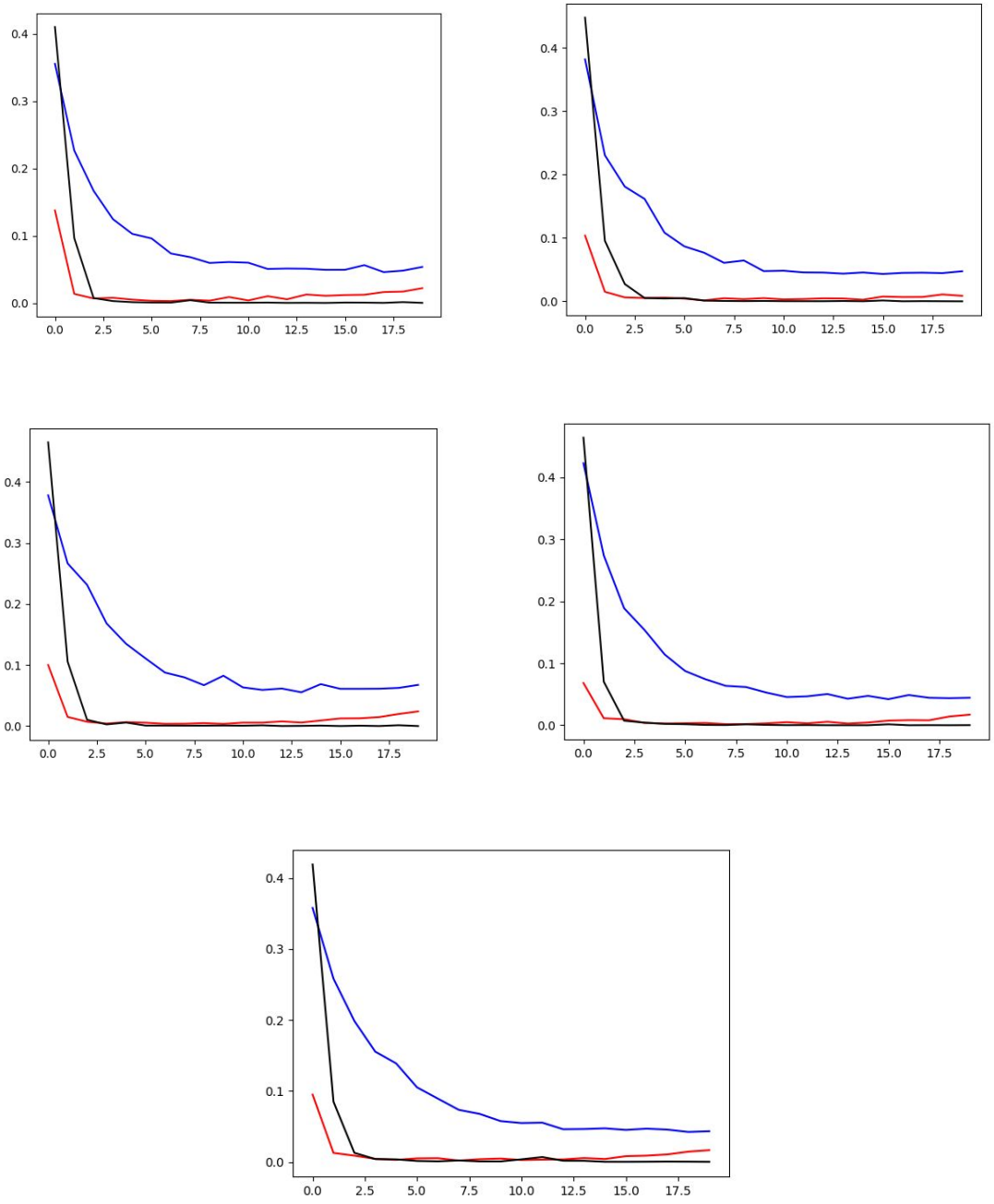


Fig. 8 Epoch training graphical representations

The 5 graphs above feature each of the models accuracy in mean squared error (left hand axis) against the number of iterations (bottom axis, units are in multiples of 10 starting at 10). The graphs show the differences in each models accuracy after training for so many epochs with the multilayer perceptron model being shown in black, the convolutional model shown in blue and the LSTM variant recurrent network depicted in red for each graph. As can be seen in the graphs starting at the first iteration of training with 10 epochs the MLP network has the lowest accuracy with a MSE greater than 0.4 with the convolutional model having slightly better accuracy averaging a MSE just short of 0.4. The LSTM has a remarkably low starting MSE value averaging at 0.1 showing that at least for low exposure systems LSTM would

likely produce significantly better results than the other networks models. This high base accuracy is due to the additional information of sequence within the LSTM model giving the model effectively more information to base its decision than the opposing network models. Iterating through the number of epochs during training an improvement can be seen in the accuracy of all three models to a certain point wherein the accuracy either stagnates or diminishes. The point where this happens for each network is seen to be the optimal number of epochs for training to gain accurate results without overfitting the problem. Both the MLP model and the LSTM model reach this value quickly with an optimal epoch value for each being determined at 60 epochs. The Convolutional network however takes significantly longer to plateau due to the complex structure of the network and its shared weightings between neurons. Due to this the optimal epoch count for the Convolutional model was taken at 150 epochs opposed to the 60 epochs of the other two models.

Following on from determining the optimal number of epochs for training the same process was applied to the batch size during training. This time the network was trained with a variable batch size starting at 16, progressing through iterations of 16 up till a maximum batch size of 160 with each network being trained through its previously determined optimal number of epochs. The following graphs were created as a result of this process.

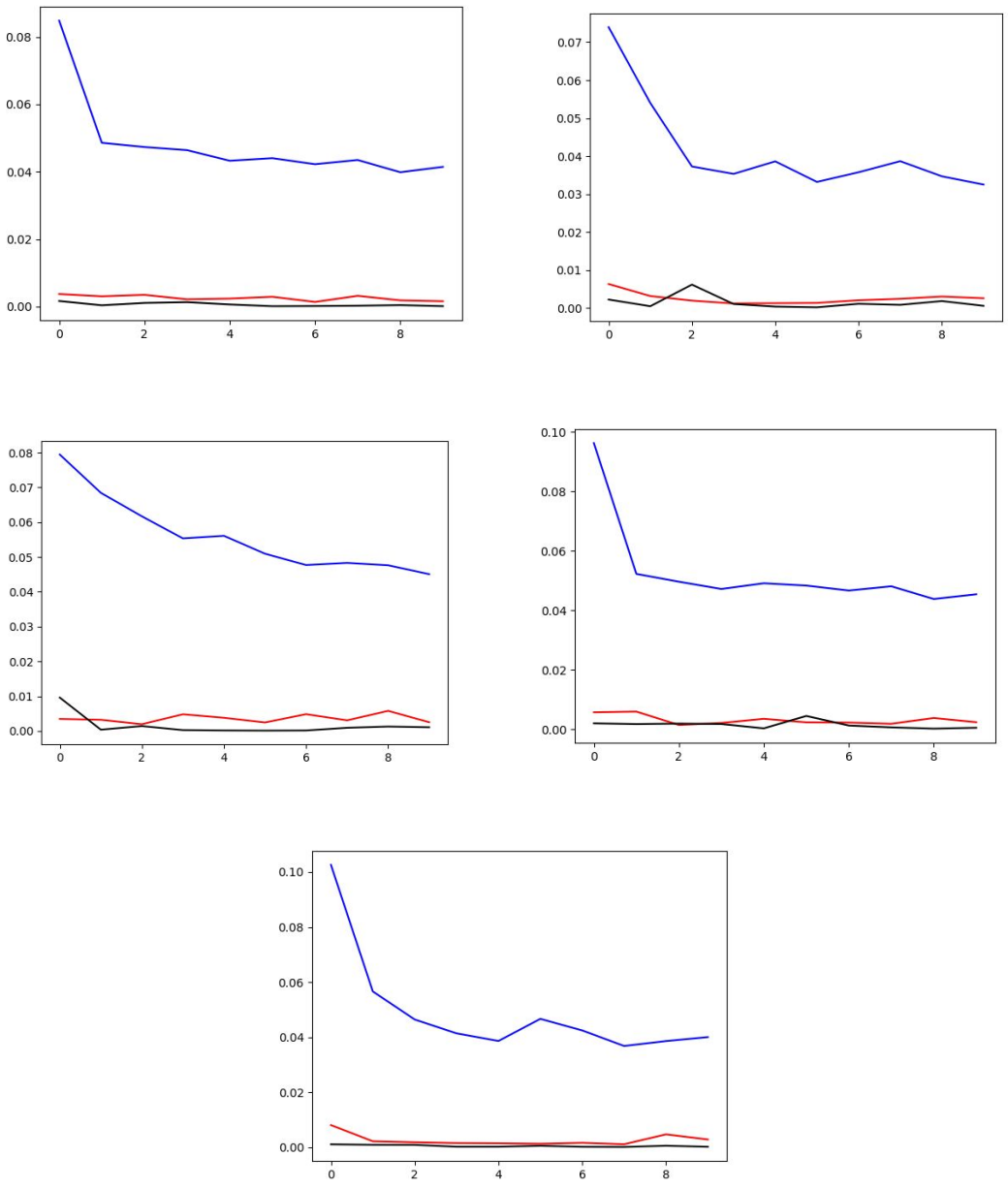


Fig. 9 Batch training graphical Representations

These graphs with are once again laid out in the same format with the MSE on the left hand side and the number of batches on the bottom axis (units are in multiples of 16 starting at 16 and ending at 160). Once again the Convolutional network model has a high starting MSE with low batch size and the error drops with increased batch size. This is due to the nature of the convolutional network and the way it analyses data. Due to the filters within the convolutional network looking over an array for certain characteristics a smaller batch size makes it harder for the network to notice any trends or markers that a stock price is going to shift one way or another. The LSTM and MLP models remain fairly consistent with their MSE value throughout the range of batches trained so the original value of 128 was kept for all three models as this coincided with the lower error values found when changing the batch size for the convolutional network.

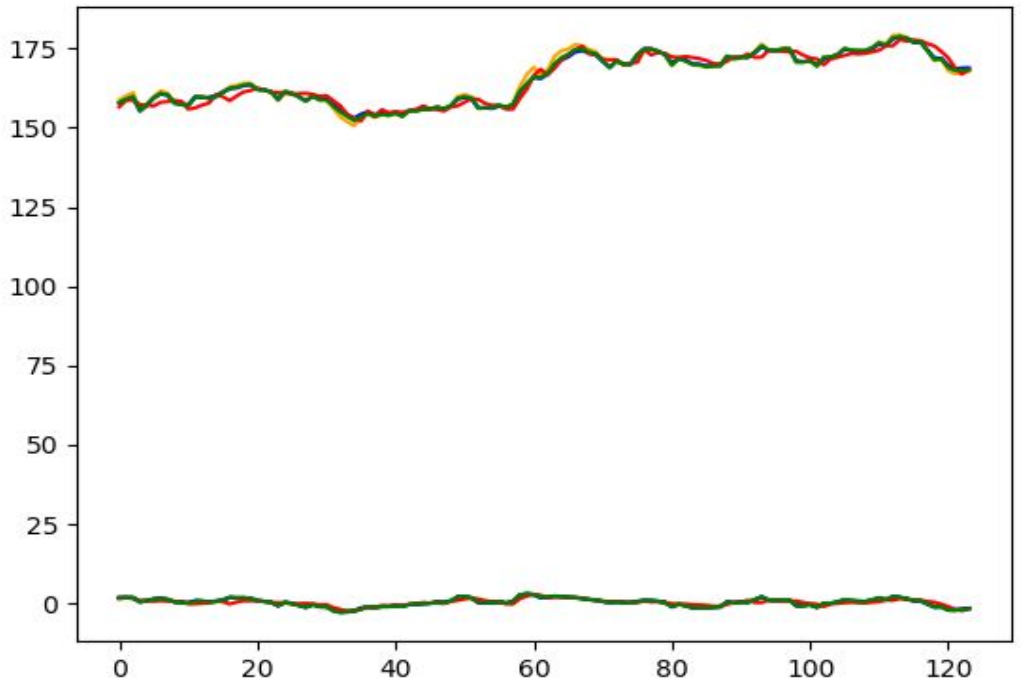
Once each of these values had been optimised all 15 models were trained once more and saved to file for usage in testing the models on unseen data.

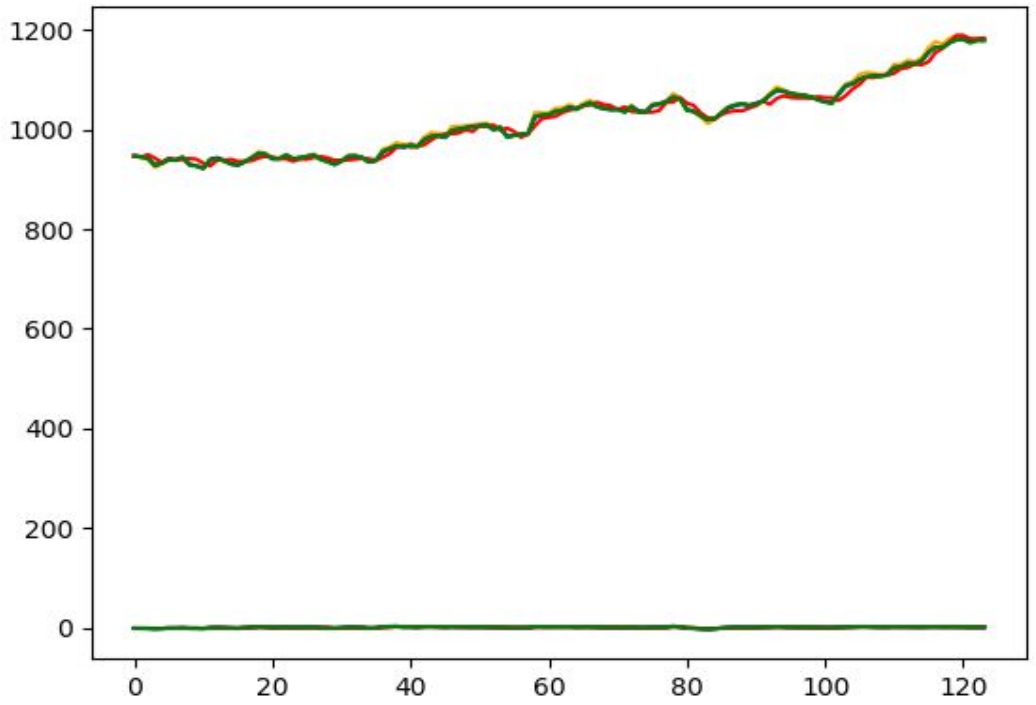
5.0 Testing and Evaluation

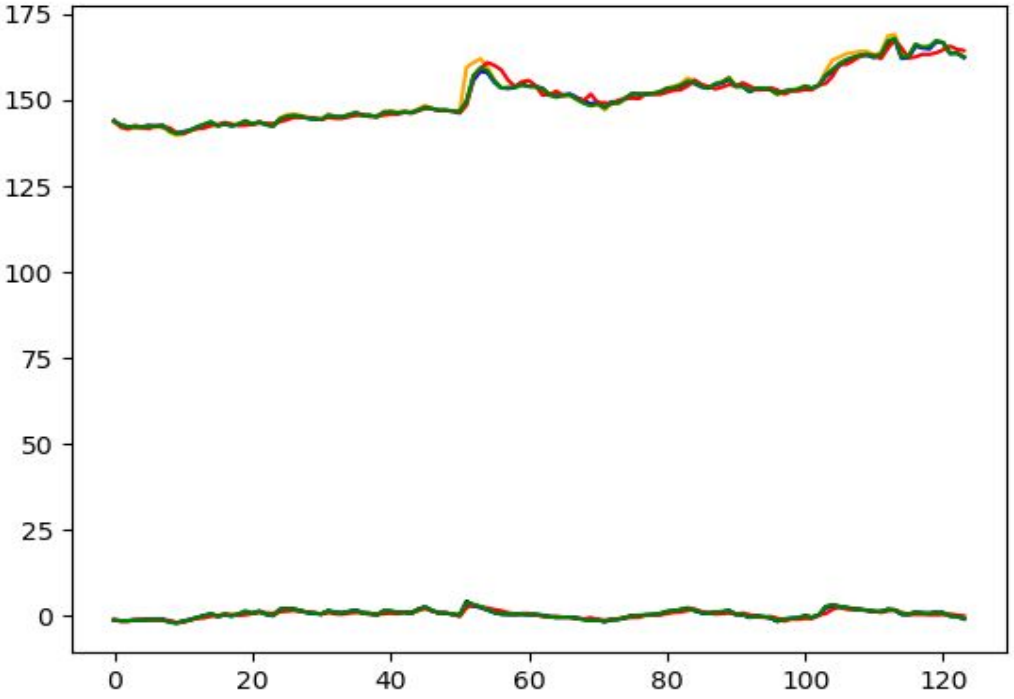
This section of the report covers the testing environment and success criteria for each model. The suitability of each model will then be evaluated and compared against the others to answer the research question and an explanation for each models performance will be given. This section of the report will also discuss the possibility for future work in this field and outline the possible direction for such a study.

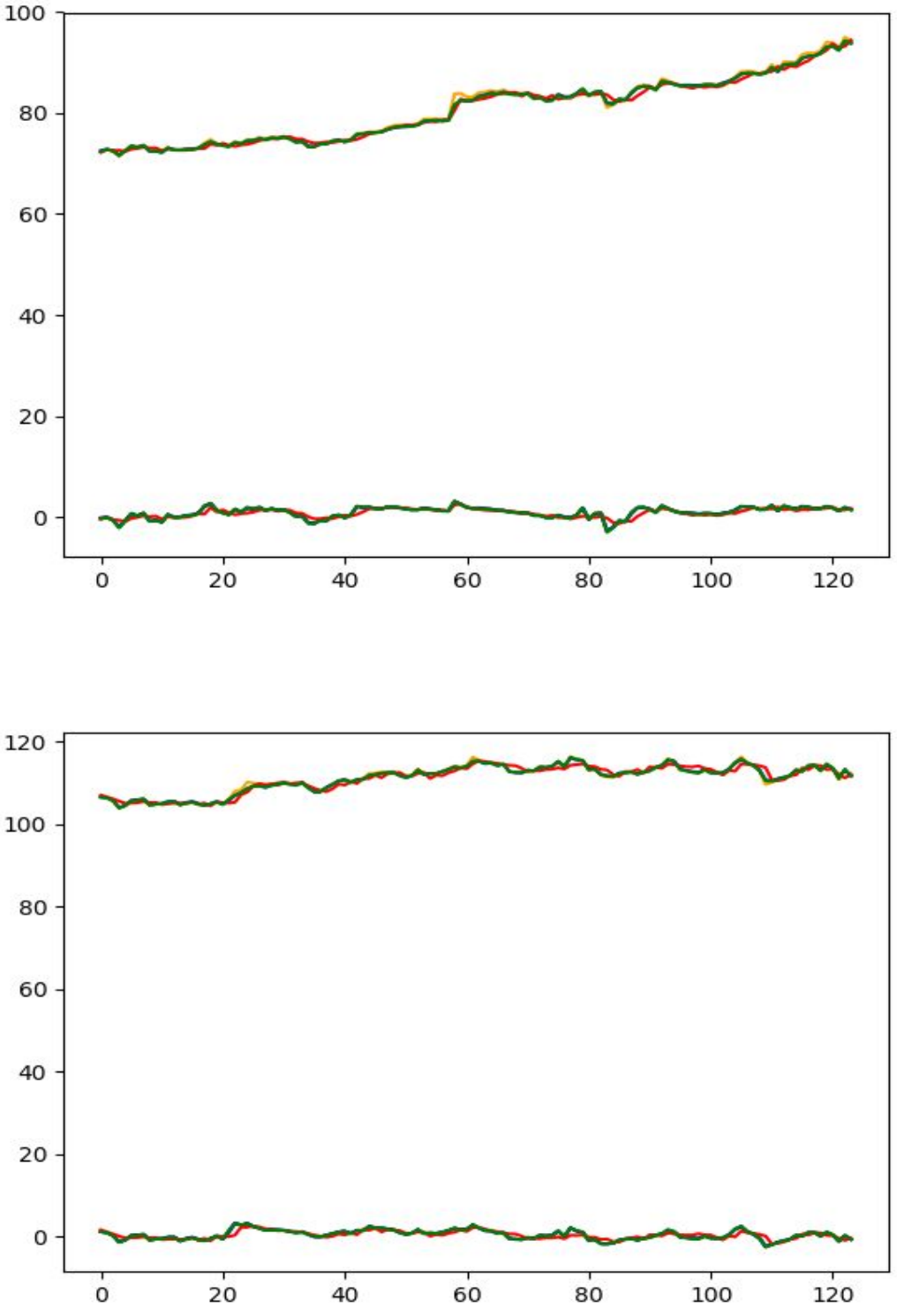
5.1 Model Testing and Evaluation

Testing of each model was completed by taking the last 10% of each time series dataset stored within the test arrays generated from the training_data function and calling each models predict command passing in this test array. Due to the models being trained and evaluated so far only on the first 90% of the data sets this last 10% is entirely new to the model and as such cannot be predicted due to familiarity. The outputs of each predict command are stored within several lists with scaled versions being created for comparison against real stock market values. These values are all then plotted against each other for each dataset resulting in the following graphical representations.


175
150
125
100
75
50
25
0
0
20
40
60
80
100
120


1200
1000
800
600
400
200
0
0
20
40
60
80
100
120


175
150
125
100
75
50
25
0
0
20
40
60
80
100
120



Figs 10-14. Graphical outputs of predicted stock values against live values

Although difficult to see, each network model is colour coded on the graphs with stock values on the left hand side against time series steps on the bottom axis. The topmost lines on each graph represent the live prices and the restored predictions whereas the

bottommost lines represent the scaled price and the network prediction outcomes. The live price of the stock is indicated in orange, the scaled live price indicated in black, the MLP models predictions (scaled and unscaled) are represented in blue, the convolutional models predictions depicted in red and the LSTM models outputs shown in green.

As can be seen from each of the graphs each models output predicts the changes in share price remarkably well and appears to be highly accurate when comparing curvature alone. It is worth pointing out however that although a series of outputs are given and each output is a prediction they are not sequences of compounded predictions. Each prediction made for the next time step this test is done using an actual result for the current time step i.e. the model is predicting one time step ahead at each iteration.

The results of each set of predictions is also stored as a set of values in csv file format for further analysis as graphical representations alone are difficult to make accurate conclusions from especially as the predictions are very similar.

These csv result files were then collated and analysed generating percentage and mean squared error figures for each dataset which can be found below.

TOTAL MSE	
MLP	18.04993984
CONV	117.2847921
LSTM	16.13550099

MSE Per Stock	Stock1	Stock2	Stock3	Stock4	Stock5
MLP	0.8853748371	15.32925425	1.584690924	0.1556430585	0.09497677083
CONV	3.80296345	108.4250572	3.381935231	0.6584614628	1.016374687
LSTM	0.4907045028	14.23798449	1.177279967	0.1531673655	0.07636466329

Average % Error Per Stock	Stock1	Stock2	Stock3	Stock4	Stock5
MLP	0.3712951143	0.276656032	0.3550128971	0.3048075758	0.1805927642
CONV	0.955299936	0.8191997291	0.764162989	0.732727986	0.6955296373
LSTM	0.2424281584	0.2692191742	0.2474589093	0.3074090796	0.1374447639

Fig. 15-17 Error tables for testing on all 5 datasets

From the above figures it is much easier to compare the results of each network models prediction accuracy. The top figure shows the combined MSE for each network across all 5 datasets with the middle figure showing the MSE for each network model for each dataset. In each table for each set of results the prediction accuracy is pretty consistent between each model with the LSTM recurrent model performing with the lowest MSE with MLP slightly behind and the Convolutional model falling far behind. The lower table shows the average percentage error of each prediction compared against the live price and once again consistently shows the LSTM network model ahead of the other two and the Convolutional model bringing up the rear in terms of its error percentage. It is worth noting however that although the Convolutional Neural Network is producing a significantly higher error score in each scenario that error score is still remarkably low considering the application. With the Convolutional model providing the worst accuracy levels across the testing at 0.955% and total MSE across effectively 10 percent of 25 years of data (roughly 913 samples) it becomes apparent that all three models perform exceptionally well in single step time series prediction for usage in stock market share price prediction.

The LSTM network model as expected performs the best in almost every scenario through testing when compared with the other two models. This is due to the sequential nature of the model and its ability to make predictions based upon previous observations and outputs along with the current state giving it a slight advantage over non recurrent models. This increase in performance is in line with other studies comparing the effectiveness of LSTM networks against MLP networks such as a study by Zachary Lipton et al in its usage for clinical diagnosis (Z. Lipton et al, 2017) and a comparison of recurrent and MLP networks for determining harmonic contributions from non-linear loads (J. Dai et al, 2008).

The MLP model performs exceptionally well with error scores slightly behind the recurrent LSTM model. This is higher than originally expected when conducting this study and is likely due to the large breadth of the hidden layer allowing for very detailed analysis of the dataset as seen in previous studies such as “Artificial Neural Network: Deep or Broad? An Empirical Study” wherein the breadth of the network not the depth of the network leads to a lower bias in the model (N.Liu, 2016). The usage of rectified linear activation functions also reducing the chance of overfitting the problem while training allowed the model to better generalise the fundamental approximation function.

Although the Convolutional model performed to a high degree of accuracy it was not able to match or beat the performance of the other two models used within this study. This is likely due to the difference in approach to detecting increases or decreases in price as opposed to the other two models. Due to the filters used within the network each prediction is based upon certain markers within the data that might indicate an upward or downward trend as opposed to making a prediction based entirely off only the current state like in the MLP model or based off state and momentum used within the LSTM model. Similar studies have been done indicating high performance with Convolutional models due to sudden changes in the datasets (S. Selvin, 2017). This ability to predict off certain markers can be beneficial for identifying anomalous spikes or dips indicating a more drastic change (this can be seen at multiple points through the prediction outputs where the convolutional model drastically

outperformed the other models) but can be detrimental when such clear markers are not apparent thus producing the lower overall accuracy.

5.2 Conclusion

Based upon the results of the study it would have to be concluded that the Long Short Term Memory variant Recurrent Network model does indeed perform with higher accuracy than the standard feed forward Multilayer Perceptron model. However contrary to the previous hypothesis the Convolutional Neural Network model on average performs with a lower accuracy than the standard feed forward Multilayer Perceptron model though could potentially be more useful in a more volatile stock or market with more distinct markers indicating an upward or downward trend.

5.3 Future work

Based upon the findings of this study there is scope for further work in the effects of combining different model architectures such as the Convolutional network model and the LSTM recurrent model. This could potentially increase effectiveness giving the model an accurate picture of the overall trend through its ability to track momentum while having the benefit of being able to distinguish certain anomalous markers. As mentioned previously the outputs of each network were predicted for one time step ahead at daily intervals (some testing was done on single time step prediction at weekly intervals with similar results). Another focus of study could be to attempt prediction for multiple timesteps. This could be interesting to see the long term predictive effects of the network though risks compounding errors.

Another option is to introduce other variables to the network to attempt to improve accuracy though this also drastically increases the complexity of the problem and will likely vastly increase training costs.

6.0 References

W. McCulloch and W. Pitts (1943). “A logical calculus of the ideas immanent in nervous activity” http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf [Online][Last Accessed 15/11/2017]

M. Hu and H. Root (1964). “An adaptive data processing system for weather forecasting”. Journal of applied meteorology, vol. 3, issue 5, pp.513-523.

T. Kimmoto, K. Asakawa, M. Yoda, M. Takeoka (1990). “Stock market prediction system with modular neural networks”. https://pdfs.semanticscholar.org/25b0/d0316ece493899d74cfb98ce7b77dca8352e.pdf [Online][Last Accessed 15/11/2017]

G.Zhang, E. Patuwo and M. Hu (1998). “Forecasting with artificial neural networks”. International journal of forecasting, vol. 14, issue 1, pp.35-62.

H. White and A.Refenes (1988). “Economic prediction using neural networks: the case of IBM daily stock returns”. https://pdfs.semanticscholar.org/a059/ffef2efc6f7f2efd68a5bbf313dcd3072c42.pdf [Online][Last Accessed 15/11/2017]

C. De Groot and D. Wurtz (1991). “Analysis of univariate time series with connectionist nets: a case study of two classical examples”. Neurocomputing, vol.3, issue 4, pp.177-192.

K. Chakrabotry, K. Mehrotra, C. Mohan and S. Ranka (1992). “Forecasting the behaviour of multivariate time series using neural networks” https://surface.syr.edu/cgi/viewcontent.cgi?referer=https://www.google.co.uk/&httpsredir=1&article=1092&context=eecs_techreports [Online][Last Accessed 15/11/2017]

I. Poli and R. Jones (1994). “A neural net model for prediction”, Journal of the american statistical association, vol. 89, issue 425, pp.117-121.

A. Borovykh, S. Bohte, W and C. Oosterlee (2017), Conditional time series forecasting with convolutional neural networks” https://arxiv.org/abs/1703.04691 [Online][Last Accessed 15/11/2017]

J. Connor and D. Martin (1994), “Recurrent neural networks and robust time series prediction” http://www.macs.hw.ac.uk/~dwcorne/RSR/00279188.pdf [Online][Last Accessed 15/11/2017]

G. Zhang (2007), “A neural network ensemble method with jittered training data for time series forecasting” http://www.sciencedirect.com/science/article/pii/S0020025507003003 [Online][Last Accessed 15/11/2017]

T. Taskaya and M. Casey (2005), A comparative study of autoregressive neural network hybrids.” Neural Networks, vol.18, issue 5, pp.781-789.

A. Bryson and Y. Ho (1979), “Applied Optimal Control: Optimization, Estimation, and Control”, IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, issue 6, pp.366-367.

S. Seth (2015) “The world of high frequency algorithmic trading”, https://www.investopedia.com/articles/investing/091615/world-high-frequency-algorithmic-trading.asp [Online][Last Accessed 15/11/2017]

G. Box and G. Jenkins (1970). “Time Series Analysis: Forecasting and Control” https://s3.amazonaws.com/academia.edu.documents/30152084/176871241.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1510925340&Signature=BFIPSkVqO20QiiH9ch42wtnk34g%3D&response-content-disposition=inline%3B%20filename%3DTime_series_analysis_forecasting_and_con.pdf [Online][Last Accessed 15/11/2017]

R.Nau, 2017, “Statistical forecasting: notes on regression and time series analysis” https://people.duke.edu/~rnau/411home.htm [Online][Last Accessed 15/11/2017]

M. Khashei and M. Bijari (2010), “An artificial neural network(p,d,q) model for timeseries forecasting”, Expert Systems with Applications, vol. 37, issue 1, pp.479–489.

C. Stergiou and D. Siganos (2017) “Neural networks”, https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html [Online][Last Accessed 15/11/2017]

A. Yilmaz, C. Aci and K. Aydin (2015), “MFFNN and GRNN Models for Prediction of Energy Equivalent Speed Values of Involvements in Traffic Accidents” http://ijaet.academicpaper.org/article/view/1072000110 [Online][Last Accessed 16/11/2017]

R. Hecht-Nielsen (1989), “Theory of the Backpropagation Neural Network”, https://pdfs.semanticscholar.org/4d3f/050801bd76ef10855ce115c31b301a83b405.pdf [Online][Last Accessed 16/11/2017]

C. Olah (2014), “Conv nets: a modular perspective” http://colah.github.io/posts/2014-07-Conv-Nets-Modular/ [Online][Last Accessed 16/11/2017]

A. Krizhevsky, I. Sutskever and G. Hinton (2017), ImageNet Classification with Deep Convolutional Neural Networks http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf [Online][Last Accessed 16/11/2017]

S. Lawrence, C. Giles, A. Tsoi and A. Black (1997), “Face Recognition: A Convolutional Neural-Network Approach”,

http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Lawrence_et_al.pdf [Online][Last Accessed 16/11/2017]

R. Rojas (1996), “The backpropagation Algorithm” https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf [Online][Last Accessed 16/11/2017]

D. Britz (2015), “Recurrent neural networks tutorial” http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/ [Online][Last Accessed 16/11/2017]

T. Mikolov (2010), “Recurrent neural network based language model” http://www.fit.vutbr.cz/research/groups/speech/servite/2010/rnnlm_mikolov.pdf [Online][Last Accessed 16/11/2017]

S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber (2001), “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies” http://www.bioinf.jku.at/publications/older/ch7.pdf [Online][Last Accessed 16/11/2017]

R. Kapur (2016), “The vanishing gradient problem” https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b [Online][Last Accessed 16/11/2017]

D. Harper (2017), “Forces that move stock prices” https://www.investopedia.com/articles/basics/04/100804.asp [Online][Last Accessed 16/11/2017]

T. Sharif, H. Purohit and R. Pillai (2015) “Analysis of Factors Affecting Share Prices: The Case of Bahrain Stock Exchange” http://ccsenet.org/journal/index.php/ijef/article/view/45844 [Online][Last Accessed 16/11/2017]

N. Jegadeesh and S. Titman (2011), “Momentum” https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1919226 [Online][Last Accessed 16/11/2017]

Theano documentation (2017) http://deeplearning.net/software/theano/introduction.html [Online][Last Accessed 16/11/2017]

Pytorch (2017) http://pytorch.org/ [Online][Last Accessed 16/11/2017]

Caffe documentation (2017) http://caffe.berkeleyvision.org/ [Online][Last Accessed 16/11/2017]

Tensorflow Documentation (2017) https://www.tensorflow.org/ [Online][Last Accessed 16/11/2017]

Keras Documentation (2017) https://keras.io/ [Online][Last Accessed 16/11/2017]

Nvidia Accelerated Computing (2017) http://www.nvidia.com/object/what-is-gpu-computing.html [Online][Last Accessed 16/11/2017]

Pandas Documentation (2017) http://pandas.pydata.org/pandas-docs/stable/ [Online][Last Accessed 16/11/2017]

N.Buduma (2015), “Preventing overfitting in neural networks” https://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html [Online][Last Accessed 16/11/2017]

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov (2014), “Dropout: a simple way to prevent neural networks from overfitting” https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf [Online][Last Accessed 16/11/2017]

Slant Community (2017) https://www.slant.co/options/14689/~tensorflow-review [Online][Last Accessed 16/11/2017]

G. Cybenko (1989) “Approximation by Superpositions of a Sigmoidal Function” https://www.researchgate.net/profile/George_Cybenko/publication/226439292_Approximation_by_superpositions_of_a_sigmoidal_function_Math_Cont_Sig_Syst_MCSS_2303-314/links/551d50c90cf23e2801fe12cf/Approximation-by-superpositions-of-a-sigmoidal-function-Math-Cont-Sig-Syst-MCSS-2303-314.pdf [Online][Last Accessed 18/04/2018]

H. Mhaskar (2016) “Learning Functions: When Is Deep Better Than Shallow” https://arxiv.org/pdf/1603.00988.pdf [Online][Last Accessed 18/04/2018]

S. Ruder (2016) “An Overview of Gradient Descent Optimization Algorithms” http://ruder.io/optimizing-gradient-descent/index.html#adam [Online][Last Accessed 18/04/2018]

Z. Lipton et al (2017) “Learning to Diagnose with LSTM Recurrent Neural Networks” https://arxiv.org/pdf/1511.03677.pdf [Online][Last Accessed 18/04/2018]

J. Dai et al (2008) “A Comparison of MLP, RNN and ESN in Determining Harmonic Contributions from Nonlinear Loads” https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4758443 [Online][Last Accessed 18/04/2018]

Nian Liu and Nayyar Zaidi (2016) “Artificial Neural Network: Deep or Broad? An Empirical Study” https://link.springer.com/chapter/10.1007/978-3-319-50127-7_46 [Online][Last Accessed 18/04/2018]

S. Selvin, V.Gopalakrishnan and V. Menon (2017) “Stock Price Prediction Using LSTM, RNN and CNN-Sliding Window Model” https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8126078 [Online][Last Accessed 18/04/2018]

Appendix

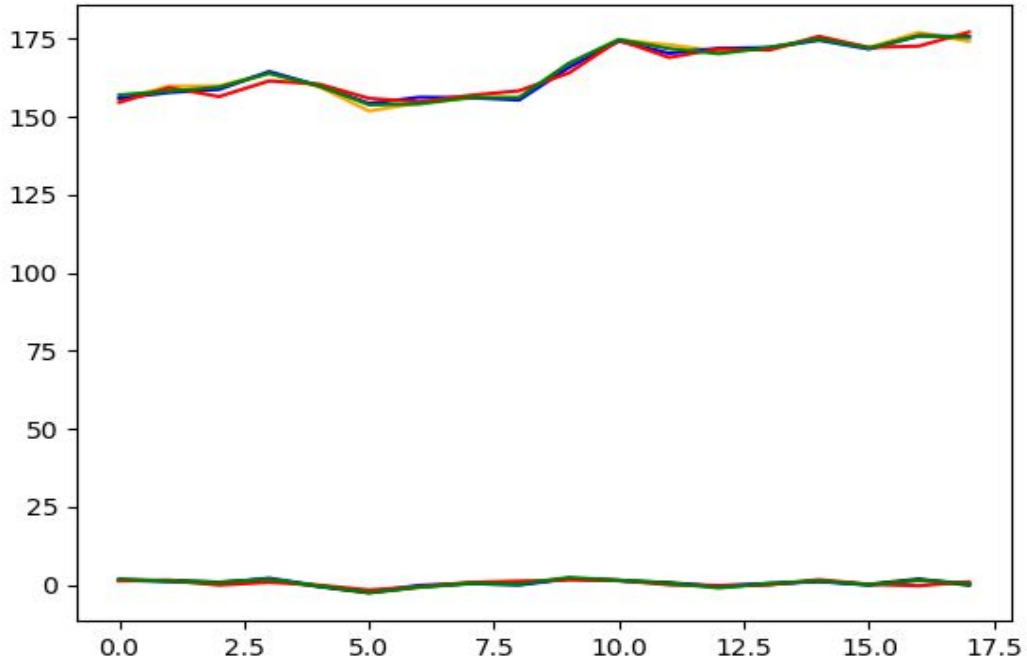
Appendix A - Code

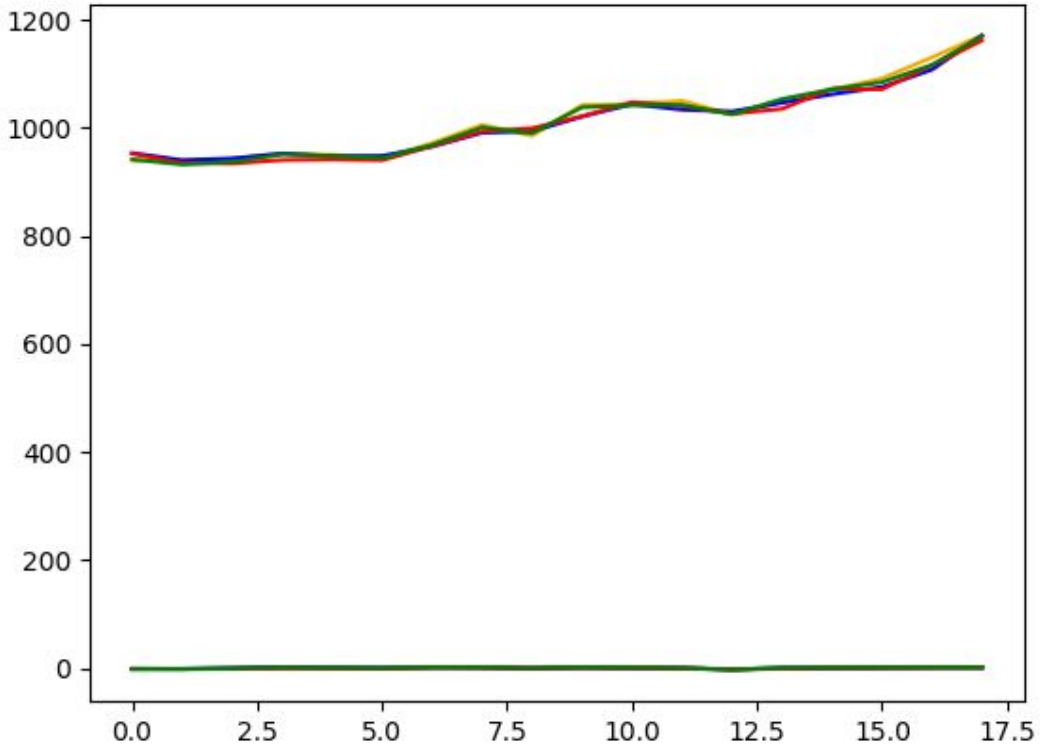
Appendix B - Results

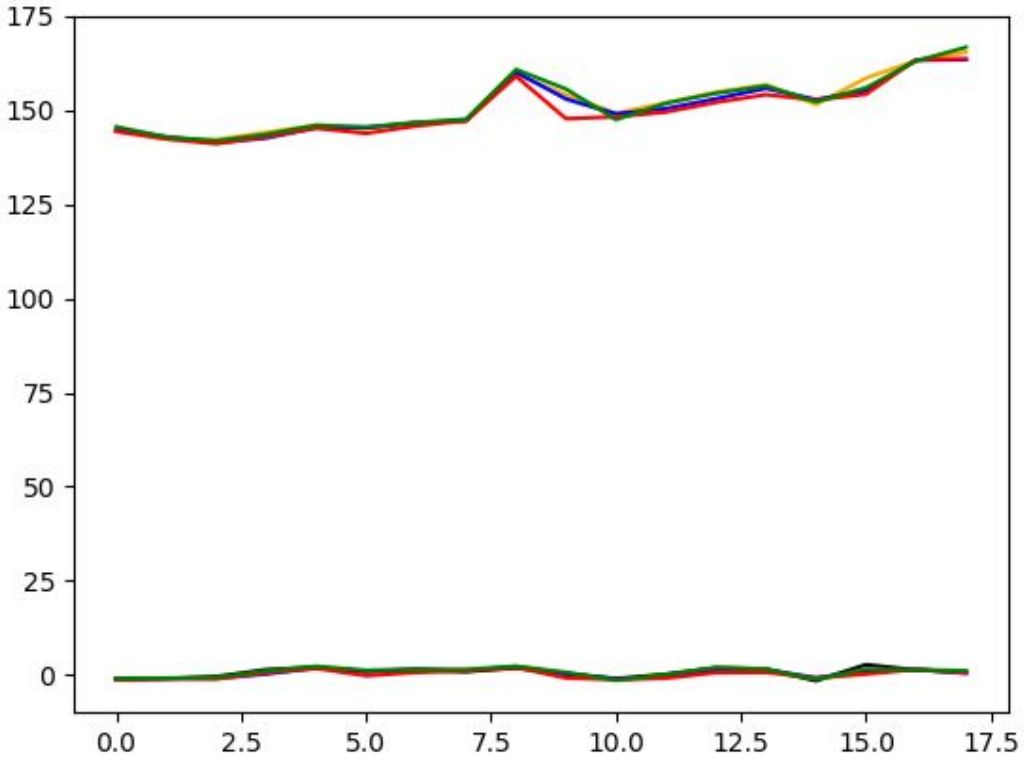
Target	158.81	160.68	161.66	155.32	157.48	159.85	161.6	159.95	157.96	157.5	157.21	159.79	159.98	159.27	159.86	161.47	162.91	163.35	164	164.05	162.08	161.91	161.26	158.83	161.5	160.86	159.05	158.29	159.88	158.67	158.73
MLP	157.84685	158.94414	159.37166	155.3469	156.96791	159.57393	159.66564	160.1699	157.81512	157.60891	157.13972	159.79188	159.48041	159.35171	160.16039	161.19493	162.3403	162.5303	163.25803	163.86555	162.69126	161.84629	161.3855	159.16301	161.18565	160.01835	159.79861	158.50718	159.90717	158.11711	159.12689
e^2	0.8076679225	1.29017794	2.850491966	0.00073261	0.262236161	0.226642649	0.872468038	0.82267881	0.0020142144	0.0120802081	0.0049392764	0.0007907344	0.2446192681	0.0793605241	0.0902341521	0.0756635049	0.32455809	0.67190809	0.9505164909	0.2346918025	0.0001267876	0.0046859481	0.01075025	0.2840996601	0.0598602225	0.0034047225	0.0220849321	0.6516018666	0.0007362089	0.1999073521	0.1575216721
% Error	0.5094784408	0.7096577211	1.048287726	0.0173190818	0.3251778004	0.297822859	0.5780074257	0.4902784834	0.0284302549	0.0697412896	0.0447845353	0.0173991989	0.3061573947	0.1786757436	0.1879081696	0.1703536261	0.3497022696	0.5016059382	0.6524207317	0.2953063361	0.0099471865	0.0435878352	0.0776245103	0.3360883213	0.1945201236	0.0362737844	0.0933848736	0.1435178165	0.8169939955	0.3817860969	0.35004095
MSE	0.8953748371																														
Average % Error	0.3712551143																														
CONVOLUTION	156.99108	158.62157	158.6937	157.27928	157.07861	156.84184	158.03856	158.21675	158.60294	158.337	155.83493	156.25039	157.12291	157.77552	160.40532	159.76022	158.52182	159.99835	161.2781	161.70921	162.49077	161.96394	161.10098	161.13074	161.01017	160.4305	160.78999	160.83351	160.5256	159.87854	160.1093
e^2	4.923605998	2.127018065	5.59937569	3.638778118	0.161113921	0.949026586	12.66385487	7.470655562	0.5819589436	0.700569	1.990617505	12.45821734	8.182963268	2.23045251	0.297379024	2.923347648	19.25612371	11.23355772	7.40873961	5.479297824	0.186731992	0.009095236	0.025073604	6.253700548	0.235060128	0.18447025	1.29729422	6.52041332	0.41679936	1.480868932	1.90246849
% Error	1.3972168	0.9110632184	1.449204023	1.261447335	0.2548831586	1.881864248	2.203801386	1.698136198	0.4785322058	0.5314285714	0.8746768225	2.209049644	1.785984488	0.9377032712	0.341123483	1.058884003	2.693622245	2.051821243	1.659895122	1.426875952	0.2534365745	0.0333148452	0.0986109388	1.576468947	0.3002043344	0.2670023623	0.7134293768	1.613286581	0.4038328521	0.7616888725	0.868959869
MSE	3.80296345																														
Average % Error	0.955299936																														
LSTM	157.87094	158.94707	159.91063	155.2847	157.34042	159.34663	160.83173	160.39288	157.58572	157.24402	156.95105	159.46442	159.69754	159.09277	159.65742	161.0588	162.29562	162.74403	163.39572	163.58366	161.99316	161.88223	161.27277	158.73076	161.476	160.82657	159.67737	158.43591	159.87658	158.74413	158.74588
e^2	0.6818036836	1.783503985	1.321051397	0.00124609	0.019482575	0.753381359	0.890238709	0.310382644	0.0753795184	0.0655357804	0.0678551025	0.0999907364	0.079783616	0.0314104779	0.0410366564	0.16908544	0.37746278844	0.3647797809	0.3651543184	0.2174729956	0.0007711779	0.0001630729	0.01045257776	0.000576	0.0011175649	0.0007491109	0.0743079391	0.0000153684	0.0004957609	0.0003521744
% Error	0.5913103709	0.7077273863	0.7136284614	0.0227272373	0.0886334727	0.3149014701	0.475414604	0.3481447655	0.1737488914	0.1625269841	0.1647115985	0.197509015	0.1785599599	0.1112784488	0.126723383	0.2546603084	0.3771284758	0.3607368225	0.3684634146	0.2842669918	0.0535784797	0.0171515092	0.0079188688751	0.0033188841	0.0148008111	0.0207820465	0.0171437516	0.0085026553	0.0024518387	0.0467196073	0.01000441
MSE	0.4907046028																														
Average % Error	0.2424261584																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Target	945.75	944.19	940.68	923.59	930.09	938.93	936.08	944.27	927.66	926.18	920.87	940.4	942.58	936.89	930.5	928.13	935.75	943.83	958.24	961.89	941.48	942.02	949.89	941.41	943.29	946.65	950.44	940.13	935.29	929.75	936.86
MLP	946.8576	945.54065	941.6531	926.25495	932.7111	939.14745	936.0562	943.0957	930.0311	926.7955	922.06085	939.2318	942.9468	938.1812	931.57535	928.1758	934.82847	942.6057	951.2343	949.61684	940.5805	941.132	946.0733	941.27716	943.33307	945.51625	948.47504	940.01105	936.4499	929.6784	936.30066
e^2	1.23677776	1.824255422	3.14386361	21.8551575	4.87016521	0.047288516	0.00096564	1.37898049	5.62211521	0.37894025	1.4181237	1.22931148	1.38469124	1.54643244	1.61594944	1.15829323	0.09209784	0.67491154	0.5499149	17.68791249	5.63141784	0.35843025	0.7865444	3.00339889	0.017644656	0.001855024	1.299030662	3.881067802	0.21141925	1.34538601	0.00512656	0.312981235
% Error	0.1171134021	0.1430448539	0.1686116075	0.5061715696	0.2818114376	0.0231604059	0.0027509948	0.1243600172	0.2556001121	0.0664557454	0.1293179276	0.1242237346	0.0389144688	0.135682951	0.1156743886	0.0049046535	0.0877937483	0.1085488016	0.4402767891	0.2492736268	0.0636763935	0.0942655453	0.1912537241	0.0141107467	0.0045659341	0.1233682464	0.2007421405	0.0126525055	0.1240150114	0.0077009948	0.0597036918
MSE	15.32625425																														
Average % Error	0.276656032																														
CONVOLUTION	947.228	944.4566	940.34766	942.15753	933.98197	942.4856	939.9758	939.5357	941.61346	940.2341	932.1887	926.76433	938.4385	938.33464	939.01833	935.9478	936.00726	937.13873	942.71858	946.1846	944.5922	941.10634	940.9343	935.9908	940.48627	940.185	943.8468	940.56963	942.6238	938.1197	938.2999
e^2	2.184464	0.07070281	55.58952188	344.7531703	15.14431706	12.64229136	1.07909764	22.41359649	194.899046	197.5177268	126.1129697	186.2043097	171.15202225	2.087562826	59.70493999	61.11799664	0.068182707	42.13656021	155.7913371	33.76266916	9.68576884	0.845745156	80.20456249	29.36772664	7.560901913	41.796225	43.47026624	29.48088194	53.78462244	70.05187809	1.96900025
% Error	0.1562780952	0.0281617074	0.9858373755	2.010384969	0.418407896	0.378686377	0.1361332083	0.5013714298	1.5041596955	1.5174726418	1.2293131148	1.4510495553	0.4393791508	0.1542166103	1.912179473	0.9423173478	0.0274923579	0.6870041573	1.3086849638	0.86981733	0.3305646429	0.0975237736	0.9482144311	0.5756471675	0.297228848	0.6820445587	0.8939697801	0.5775403402	0.7841204332	0.9002097338	0.1493819781
MSE	108.4265572																														
Average % Error	0.8191997291																														
LSTM	946.4499	944.8692	941.49603	928.1832	932.8755	940.1492	939.09646	944.7059	928.64935	927.22206	922.55023	939.72046	941.01227	936.1689	930.4164	928.36084	935.5328	943.06585	952.3043	950.6113	941.04193	941.3043	948.73865	940.7186	942.815	945.9165	949.2553	939.47705	934.756	929.96367	936.64746
e^2	0.48996001	0.46131264	2.005140961	21.29749824	7.75901025	1.48644864	0.83969939	0.19009881	9.978813225	1.087139876	2.823172853	0.461774616	1.140047353	0.51996521	0.00969896	0.053287166	0.04717584	0.3239317225	0.81833443	1.99081369	0.19196832	0.81222849	1.32089622	0.47803396	0.225625	0.53609225	1.40351409	0.4263437925	0.285159	0.045740769	0.04317329
% Error	0.0740047513	0.0719346741	0.1506266899	0.4973202395	0.2984671454	0.1296496358	0.0976952651	0.0461626416	0.1086500658	0.112576389	0.1624611508	0.0722507401	0.1132773676	0.0789674134	0.0098441696	0.0248715154	0.0232113278	0.0603149535	0.3073259078	0.1446229468	0.0485299316	0.0759753238	0.1212867715	0.0734430266	0.0603559705	0.0772724879	0.1246475317	0.0984531607	0.0570845806	0.0230029577	0.0236642206
MSE	14.23788449																														
Average % Error	0.2682191742																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Target	143.47	142.11	141.77	141.84	141.84	142.32	142.07	142.5	140.7	139.7	140.33	141.61	141.14	142.94	143.74	142.51	143.14	142.56	143.03	144.08	143.04	143.82	142.9	142.45	144.86	145.76	145.99	145.64	144.87	144.55	144.39
MLP	143.74863	142.75264	142.2791	142.25195	142.20341	142.82102	142.40453	142.89382	141.22346	140.4169	140.81078	141.3218	142.14735	142.8379	143.5024	142.58424	143.02876	142.51358	142.97563	143.71254	143.63264	143.5813	142.81453	142.49835	144.38833	145.17049	145.43417	145.20152	144.70851	144.45007	144.3949
e^2	0.07634678	0.41299851	0.29916291	0.16975289	0.13206682	0.09061304	0.11191032	0.03759819	0.27401033716	0.51394561	0.23113017	0.09721924	0.00095402	0.01042441	0.06845376	0.00951157	0.01233743	0.00215458	0.00256343	0.13928858	0.00804686	0.13902	0.00720512	0.00237722	0.20503867	0.34173938	0.34752204	0.30894958	0.11459871	0.01243002	0.00938804	0.00602401
% Error	0.1942878483	0.4822130744	0.3591027721	0.2954328821	0.2562112239	0.2115092749	0.2354684311	0.1360140361	0.372035601	0.5131710809	0.3425924606	0.2211190696	0.0051709821	0.0714285743	0.1652984555	0.0520945865	0.0277141263	0.0325617284	0.0378731746	0.2506386673	0.0551454138	0.1689135311	0.0598115868	0.0339417394	0.3394104683	0.4044388035	0.380731557	0.2325683661	0.0766852304	0.0691317831	0.0533163681
MSE	1.284890024																														
Average % Error	0.3550128671																														
CONVOLUTION	144.16348	142.10712	141.53348	142.67415	142.0797	141.68143	142.60367	142.05312	141.96721	140.54628	140.12587	141.12347	141.664	141.8661	142.5327	142.02378	143.51372	143.0673	142.50224	142.67943	142.0862	143.26749	143.25985	143.18697	143.6373	144.25201	144.95206	144.95103	144.08674	144.85594	144.29706
e^2	0.4809145104	0.0000082944	0.0559417104	0.6958062225	0.05745809	0.4077716449	0.2848036689	0.1997017344	1.605821184	0.7161898384	0.04166995691	0.0128754469	0.226076	0.14896981	1.45757329	0.17121388	0.13996663841	0.25735329	0.19163638176	1.96159325	0.00289444	0.3012673001	0.12877332251	0.5431247809	1.35187129	2.27403384	1.27545196	2.24582632491	0.02153262761	0.08939928361	0.00847131616
% Error	0.4833623754	0.0020205091	0.1668338037	0.5880922166	0.1699932318	0.4486860596	0.3756387666	0.3136	0.9009467992	0.6057833225	0.1454642628	0.0094997025	0.3348811031	0.7495950609	0.8396192987	0.2903515543	0.2910870476	0.3558501684	0.3360616054	0.9720789122	0.0378116582	0.3941677096	0.2511199641	0.5173534574	0.9026370289	1.034570527	0.710350024	0.4040907393	0.1013257837	0.2116499461	0.0637440259
MSE	3.361935231																														
Average % Error	0.764162999																														
LSTM	143.03121	142.48818	142.27335	142.22095	142.15699	142.54948	142.3198	143.01299	141.04556	140.29475	140.64435	141.1894	142.19519	142.94373	143.69183	142.45238	143.13988	142.53923	143.01053	143.08016	142.99808	143.69974	142.8614	142.45283	144.09601	145.20653	145.90198	145.34282	144.7605	144.52542	144.37123
e^2	0.2127148641	0.3354494724	0.2533612225	0.1451229025	0.09884996801	0.0526610704	0.06240004	0.01278674901	0.1194117136	0.3537275625	0.2998159225	0.00076516	0.0030409361	0.00001391	0.0923203469	0.00332036441	0.00000001144	0.00043139329	0.00164269809	0.0009669256	0.00175728641	0.0154405476	0.00149396	0.0000001225	0.0696696025	0.2157323809	0.1584199204	0.0388799524	0.00354025	0.0006041764	0.0003523129
% Error	0.3214679627	0.4075975259	0.355048907	0.2885772702	0.2227791878	0.1612422708	0.1756288168	0.0792912287	0.2456003699	0.4257337151	0.2244078961	0.1243883413	0.0882791614	0.0028094864898	0.0335188648	0.0404322507	0.0000838349008	0.0145893415	0.0283387125	0.0692948362	0.0293064877	0.0863996662	0.0270118643	0.0020007020	0.1622104101	0.3186539517	0.272535112	0.1354618545	0.0410854543	0.0170449671	0.0129995152
MSE	1.177279967																														
Average % Error	0.2474589093																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Target	72.4	72.79	73.47	71.41	72.5	73.59	73.22	73.65	72.4	72.49	72.15	73.16	72.72	72.69	72.82	72.83	73.05	74.01	74.77	73.94	73.61	73.4	74.34	73.98	74.76	74.68	75.21	74.77	75.31	75.16	75.44
MLP	72.550156	72.81399	72.52629	71.706116	72.52834	73.4122	73.271736	73.5893	72.433975	72.42908	72.23559	73.04578	72.7466	72.29713	72.81221	72.87577	73.05163	73.70865	74.280235	73.84028	73.604065	73.41531	74.13266	73.865566	74.4483	74.50639	74.89648	74.884285	75.07015	75.05655	75.25209
e^2	0.02254682434	0.00957552	0.009660924	0.02786948546	0.00085031556	0.03161284	0.02676813686	0.00389449	0.001154309625	0.00000064	0.007319093416	0.01304624	0.00007796	0.00005583899	0.00000606841	0.00029498929	0.00003059223	0.00963305923	0.2396857552	0.00994407784	0.000035224225	0.00023439961	0.04329899756	0.008989803556	0.0094079	0.03225597521	0.09629228226	0.006514911225	0.05759269023	0.0235162225	0.03524405479
% Error	0.2073677901	0.0329078238	0.1358286359	0.4148702143	0.0360866517	0.2416089143	0.2706882608	0.2824168639	0.6468297658	0.9911036049	0.1185686625	0.1581259848	0.0857855787	0.0098087786	0.0166976105	0.0828498146	0.0022313483	0.4072895555	0.8550287548	0.1349661077	0.0080627832	0.0298583083	0.2798077213	0.1275128413	0.4196084165	0.2405081696	0.4186541417	0.1079515499	0.3186626802	0.2040313997	0.2488520879
MSE	0.1666436585																														
Average % Error	0.3048075758																														
CONVOLUTION	72.22048	72.76973	72.581024	72.61098	72.39271	72.83829	72.69057	73.24359	73.061905	73.06869	72.58928	72.87184	72.66	72.64546	72.63311	72.783985	72.961624	72.97337	73.84532	73.603294	74.0086	73.64046	73.38483	73.676384	73.79892	74.11258	74.51581	74.8212	74.89681	74.84941	75.21178
e^2	0.00322714304	0.0004108729	0.01323632868	0.14235295	0.01151111441	0.56506779041	0.05263812479	0.00420187099	0.0038311229	0.2325713561	0.1929669184	0.0803081856	0.00000401	0.00019636116	0.03459767221	0.00035079225	0.00761031737	0.0026708224	0.0058296681	0.2264617744	0.0072471169	0.0063441225	0.0074622625	0.2170693764	0.02529864	0.0236742754	0.0222293729	0.0068584449	0.0173585925	0.0454073481	0.02715113016	0.03294905826
% Error	0.2479558011	0.0278472316	0.1531909448	1.68180927	0.1470682089	1.0214853807	0.3133433488	0.5518126273	0.6142334254	0.706544213	0.6088426888	0.3938764352	0.0826982583	0.0612730288	0.2566465257	0.0906425934	0.1209801506	1.4006620273	1.236860211	0.465377333	0.5415925132	0.3275885559	1.284666828	0.4104028116	1.285553772	0.7588018211	0.6230022803	0.6847866178	0.4055364411	0.4132384247	0.3025583245
MSE	0.5684614828																														
Average % Error	0.732727986																														
LSTM	72.3495	72.76309	72.46628	71.56732	72.4958	73.50469	73.149536	73.584564	72.39799	72.49099	72.17834	73.121544	72.68685	72.653496	72.78533	72.78524	72.99832	73.78459	74.29412	73.85467	73.53035	73.32955	74.20958	73.9072	74.568756	74.53777	74.95143	74.63825	75.09691	74.985224	75.258484
e^2	0.002555025	0.0097241481	0.00001238384	0.02474958224	0.00001936	0.0072779961	0.00496157296	0.00428187099	0.000004041	0.000003901	0.00000004401	0.00139669184	0.0000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00000000001	0.00529984	0.0036742754	0.0022293729	0.0068584449	0.0173585925	0.0454073481	0.02715113016	0.03294905826
% Error	0.6697513912	0.0369936352	0.0081319858	0.2203052794	0.0806866517	0.1159260769	0.0962360109	0.0888472691	0.0277623409	0.0913657061	0.0392702928	0.0826842376	0.0458560858	0.0502187371	0.0476105455	0.0461584801	0.0270145064	0.0456669604	0.9364584728	0.1151338923	0.1082584069	0.0969802643	0.1754371805	0.0984049742	0.2568165939	0.1904529578	0.3437973674	0.1762070349	0.2629504714	0.2192335349	0.2406097961
MSE	0.1631673635																														
Average % Error	0.3074090796																														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Target	106.6	106.4	105.69	103.81	104.46	105.7	105.89	105.16	104.55	104.82	104.89	105.41	105.96	104.61	105.12	105.37	104.78	104.53	104.95	105.24	104.76	105.07	106.01	106.3	110.19	109.94	109.8	109.2	109.74	110.05	110.33
MLP	106.09004	106.41996	105.7706	103.99291	104.545395	105.74584	105.88827	106.08078	104.7805	104.885475	104.95376	105.432415	105.98178	104.83085	105.2493	105.489914	105.03083	104.575676	104.90874	105.499664	104.831436	105.734535	106.96015	107.56296	109.6216	109.20337	109.29021	109.96563	109.44024	109.83516	110.02918
e^2	0.0009839696	0.0009394016	0.00049636	0.03345609881	0.007292306025	0.002101305	0.000029929	0.006275084	0.05313025	0.004286975825	0.0005645376	0.001796932225	0.0000031684	0.0009179225	0.01671849	0.01437939074	0.05236316899	0.002863299	0.001732878	0.025492929	0.005103192096	0.1125387662	1.902185023	0.5993733996	2.45987656	0.5426237589	0.2596854441	0.6545543449	0.6986560576	0.0505530256	0.3901104338
% Error	0.0985628705444	0.0187909885	0.0762507626	0.1761969882	0.0817489483	0.0433680227	0.0016337788	0.0746232102	0.2204868753	0.0624642238	0.0226436721	0.0402361783	0.0015862479	0.2157089555	0.1230022831	0.113807962	0.2183909143	0.0436954645	0.0383139503	0.1515701538	0.0681901481	0.3182675092	0.871993334	0.4599064535	1.423359652	0.6730291668	0.4642896175	0.2138619414	0.2731547294	0.2042885699	0.2720783105
MSE	0.9949767783																														
Average % Error	0.1605827842																														
CONVOLUTION	107.02798	106.559746	106.10927	105.58203	105.01406	105.12097	105.184875	105.48702	105.21691	105.18767	104.73979	104.923225	105.09523	105.06895	104.97243	105.31033	104.805244	105.09736	104.450264	105.407394	105.176955	105.10725	105.29796	107.14335	107.7597	109.23645	109.83488	109.544846	109.77551	109.42529	110.00398
e^2	0.1831668804	0.0255187452	0.1732607129	3.140093201	0.3066824836	0.3352757409	0.4972012656	0.4529028804	0.444768461	0.1351612269	0.0361764441	0.2389499066	0.2180111529	0.2106351025	0.0217756949	0.0035605089	0.00637258536	0.3218701369	0.2497360897	0.004541061236	0.175523292	0.9256875625	7.355160962	1.337839233	5.95084449	0.4946826525	0.6012027024	0.1186187837	0.6012609601	0.4028567841	0.1062690404
% Error	0.4014821764	0.1501372118	0.3938594001	1.7569993646	0.5304039624	0.5478051088	0.6559032959	0.6332999171	0.6378861789	0.3507632131	0.1812732298	0.4617920561	0.4402898525	0.4387247873	0.1403824201	0.0566290154	0.2240923804	0.5427494499	0.4761657932	0.0639778635	0.3969188622	0.8078553222	2.5109156556	1.0680055	2.213721753	0.6399399673	0.0315846994	0.3157930403	0.0323583014	0.5766945303	0.2954953322
MSE	1.016374687																														
Average % Error	0.6955296373																														
LSTM	106.49908	106.3288	105.65303	103.93715	104.44456	105.70063	105.84161	106.09051	104.54126	104.7986	104.910934	105.37179	105.51191	104.61812	105.117615	105.3429	104.77027	104.56069	104.95811	105.34036	104.77349	105.034635	107.00863	107.62558	108.600764	109.17847	109.25034	109.63928	109.33477	109.65733	109.93523
e^2	0.01018484664	0.005606944	0.001366780	0.016571225	0.009238306	0.000003369	0.002341621	0.006318691	0.000076876	0.00045796	0.000363512356	0.001460004	0.002322761	0.000059344	0.000605882225	0.00091	0.00034129	0.00014761	0.000065772	0.00009129	0.00018981	0.002057983225	1.002741877	0.4548423364	2.525007495	0.5799279405	0.3019128852	0.0012027024	0.1186187837	0.6012609601	0.4028567841	0.1062690404
% Error	0.0946719667	0.0669172923	0.0349796749	0.1224833831	0.0147807733	0.0005962649	0.0459839623	0.0748754393	0.0583093663	0.0204159515	0.0181702871	0.0362489274	0.0455178203	0.0077621642	0.0022683518	0.0257188532	0.0029812306	0.0033599235	0.0077274892	0.0093475922	0.0129770523	0.0427682618	0.0071886011	0.6227331467	1.4422506658	0.6926778243	0.300424408	0.2808791209	0.3692837142	0.3658407141	0.3578083993
MSE	0.9763646329																														
Average % Error	0.1374447639																														

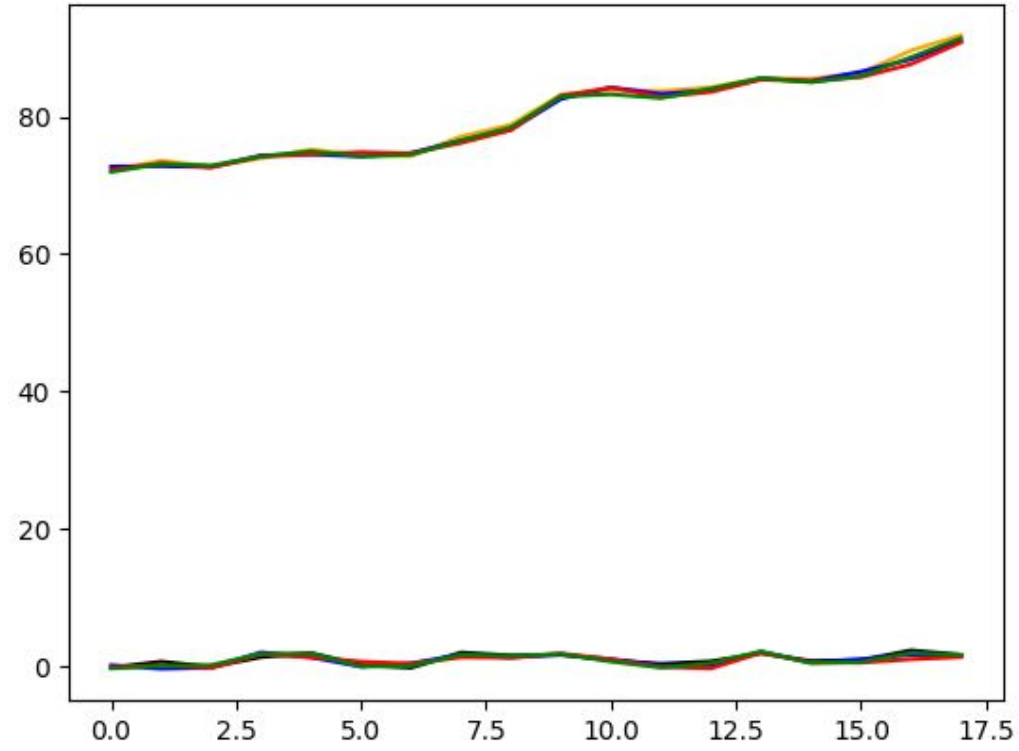
95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
174.35	175.01	175.01	170.37	170.8	171.08	169.23	172.26	172.23	173.03	175	174.35	174.33	174.29	175.28	177.09	176.19	179.1	179.26	178.46	177	177.04	174.22	171.11	171.51	167.96	166.97	167.43	167.78
174.20189	174.87969	174.7552	171.13873	170.79704	171.21472	169.79381	171.59675	172.26317	173.02364	174.7453	174.22528	174.1041	174.19547	174.86899	176.46152	175.94276	178.0411	178.52666	178.16574	176.82008	176.98038	174.46914	171.98279	171.8064	169.6491	168.57628	168.79596	168.86338
0.0219365721	0.1091046961	0.06492304	0.3234538129	0.0388247616	0.0181494784	0.3178617161	0.4372516625	0.0011002489	0.0000484416	0.06467209	0.0155550784	0.02603081	0.0089359209	0.1689292201	0.4740947104	0.0611276176	1.42126921	0.6377875556	0.0865889476	0.0323712064	0.0033545444	0.0724363396	0.7617623841	0.15713296	2.65305881	2.580135435	1.865846722	1.217447424
0.06494981359	0.1887377864	0.1455916805	0.3334290907	0.1154982415	0.07874678513	0.3331819689	0.3838874097	0.01925913023	0.004022423857	0.1455428571	0.07153427015	0.1295816276	0.0542379089	0.3344876789	0.3887746697	0.1403257847	0.5912339475	0.4089929376	0.1648884904	0.1016497175	0.03367600542	0.1544828378	0.5109753901	0.2311235496	1.0096561099	0.9520171288	0.8158394553	0.6576349982
174.23969	174.10986	173.93936	174.06238	172.86705	171.9944	171.65956	169.82155	171.63023	172.21525	172.79088	173.36076	173.246	173.40111	173.9064	174.4105	175.82582	175.81625	177.79146	177.26593	177.67503	177.42224	176.44771	175.79475	174.03218	172.00665	168.84624	166.88269	168.36217
0.0121682961	0.8102520196	1.14627001	12.19671806	5.139515703	0.83612736	5.902713203	5.946638402	0.3597240529	0.9638012676	4.860211174	0.9765957776	1.175056	0.7901254321	1.68677696	7.17972025	0.1328270724	10.78301406	2.156609732	1.425031165	0.4556655009	0.1461074176	4.962691844	21.94668256	6.361391952	18.37537622	3.520276538	0.2995482361	0.3273785899
0.06323692852	0.5143363236	0.611759328	2.04747611	1.328868699	0.5344867898	1.435649707	1.415583683	0.3482378215	0.4706663238	1.262354286	0.5673874391	0.6218092124	0.5100063113	0.783660429	1.613072449	0.2066973154	1.83347292	0.8192234743	0.5690987164	0.3813728814	0.2159060099	1.2786763385	2.737858687	1.470573144	2.4092939579	1.123898665	0.3268888491	0.3410238699
174.25833	174.90933	174.93237	170.7443	170.67883	171.11682	169.40446	172.22891	172.24155	173.05823	174.97217	174.42997	174.3558	174.2857	175.24309	176.74872	176.07712	178.36353	178.70703	178.22601	177.01567	177.06029	174.29787	171.42996	171.74646	168.86026	168.84195	168.18488	168.27487
0.0084033889	0.0107474689	0.009029149	0.03038949	0.009609489	0.0013557124	0.0334352916	0.0099665881	0.0001334005	0.0006880129	0.0007745899	0.0056043806	0.00006564	0.00031649	0.0013623481	0.1154726384	0.0127418944	0.8572174609	0.3057758709	0.0547513201	0.0003455489	0.0004116841	0.0066037369	0.1073680025	0.0559133316	0.9656149676	1.149076883	0.5088436144	0.2448963169
0.0525781474	0.8592368105	0.644357465	0.1021867855	0.845035169	0.021522669	0.1039004688	0.0180482908	0.006708146755	0.015155922894	0.011502857	0.004146831	0.014795151	0.0224671524	0.021057736	0.1927155684	0.0640572018	0.4167896031	0.3084737253	0.1311162165	0.0086531107	0.0114806685	0.0446963569	0.1883866904	0.137865512	0.5695808526	0.6420015572	0.4508630472	0.2849617225
95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
1073.56	1070.85	1068.86	1066.85	1060.2	1056.95	1053.4	1073.21	1091.82	1095.76	1110.29	1114.21	1112.79	1110.14	1112.05	1130.65	1130.7	1139.1	1135.97	1143.5	1164.16	1176.17	1171.29	1182.14	1187.56	1186.46	1177.37	1182.22	1181.59
1071.5504	1059.7333	1058.1847	1064.2467	1059.299	1055.6763	1053.7151	1071.4113	1087.0523	1091.8317	1109.5418	1105.3083	1107.3015	1107.4154	1108.6233	1123.3888	1125.6123	1131.9717	1130.592	1138.1566	1154.539	1165.1445	1164.7771	1173.672	1180.2919	1181.6423	1175.7495	1179.5021	1179.2356
4.03849216	1.24701889	0.45603009	1.00661089	0.361201	0.07491169	0.99928801	3.23532169	15.43154089	93.08776324	79.24026289	30.12363225	7.42344515	5.88887289	52.72502544	25.88469129	50.81266089	28.9222884	28.53053398	92.563641	121.5616502	42.41789641	71.707024	55.77251781	23.40334129	2.82602025	7.38886641	4.21645156
0.1671902828	0.1042816454	0.06317946223	0.09413144439	0.06668741747	0.02591978787	0.02991266376	0.1676000037	0.4083937995	0.3585000365	0.6989801764	0.7989247987	0.493219745	0.2454285045	0.2182186053	0.6422146553	0.4499662016	0.6257835133	0.4734279955	0.4671097508	0.8264327927	0.9374070075	0.5569450444	0.7163280153	0.8288608576	0.4977354665	0.1376372763	0.2289979685	0.1737827842
1064.1564	1063.0883	1063.2401	1062.4746	1065.6007	1063.6796	1063.7733	1067.9463	1067.8617	1081.0687	1090.4624	1104.8823	1104.2005	1107.6919	1108.861	1112.6992	1121.9515	1124.3176	1131.301	1129.2653	1136.0638	1154.3204	1162.4064	1175.0717	1188.5736	1185.6165	1182.0621	1181.7227	1183.2943
88.42769296	60.24396689	31.48219661	11.3833516	30.14778649	59.74671616	107.6053529	232.8800377	566.1864249	215.8342957	393.1337218	87.00598729	77.23531025	5.99319361	10.169721	322.2312206	76.53625225	218.5193498	21.799661	191.3889241	789.3964544	477.4050202	78.91834896	49.96086469	1.02738496	4.58463225	22.01560241	0.24730729	2.90453849
0.6758288229	0.7248187344	0.5249424821	0.3166862129	0.5178928504	0.7320043563	0.9847446364	1.422247277	2.168379874	1.340740673	1.765803709	0.8371581659	0.7698815192	0.220521736	0.2867678613	1.58765312	0.7737242416	1.297726275	0.4110143754	1.209855706	2.413431143	1.85769084	0.7584456182	0.5979241037	0.08535147698	0.1800704605	0.3985236285	0.04206492869	0.144237849
1071.838	1069.5211	1067.8423	1065.0894	1059.3588	1054.7413	1052.286	1071.2653	1087.7183	1091.4094	1101.4857	1105.0806	1107.2965	1106.4325	1108.5323	1123.8948	1125.5754	1132.9973	1131.3202	1137.2635	1154.5449	1165.5133	1164.2565	1174.0376	1179.6605	1180.3417	1173.4907	1178.1348	1177.557
2.065284	1.76597521	1.03571329	0.57851236	0.70767144	1.46095569	1.240996	1.75483009	14.46292289	18.92772036	77.33971249	66.08714436	30.50905225	13.74555625	10.35359329	44.29168704	26.28152516	37.24294729	21.62064004	31.54507225	92.46014801	113.5662549	49.46871656	65.64888576	62.46210025	37.67872689	15.04896849	16.68885904	16.265089
0.1604009061	0.1240976784	0.09521356205	0.07136088566	0.07934352009	0.114465647	0.1057528006	0.123433438	0.348294122	0.3970394977	0.7920723415	0.7296111146	0.4953549925	0.3339868871	0.2883485005	0.5666171671	0.4532236668	0.53574752	0.4093241499	0.4911674683	0.8258258896	0.9060609666	0.6004832279	0.6854010523	0.8651874432	0.5173538534	0.329486808	0.3455532811	0.3413197471
95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
152.95	151.5	152.5	152.83	153.13	154.04	153.42	154.25	158.49	161.7	162.49	163.47	163.83	164.16	164.2	163.14	163.85	166.65	169.12	167.37	167.6	166.25	165.37	165.47	167.34	166.8	163.62	163.7	163.4
153.10376	151.8899	152.51805	152.96393	153.62505	153.81505	153.46161	154.172	156.95038	158.27824	160.04155	161.51704	162.3213	162.97765	163.00629	162.37334	163.0212	166.27448	167.45995	162.10415	162.39255	165.72069	165.10754	164.88878	166.59388	166.7054	163.52042	163.85216	162.37445
0.0236421376	0.152022201	0.00032586025	0.01153586449	0.01019909025	0.00566259025	0.00173113921	0.006084	2.370429744	11.7084415	5.994907403	3.814052762	2.27617589	1.445845522	1.424943564	0.5877675856	0.88690944	5.84309527	2.755786003	0.9722803225	0.8430355025	0.2801690761	0.9688852516	0.3378166864	0.1625218596	0.00894916	0.0099161764	0.0014318696	0.0006528025
0.1005295848	0.2573597361	0.01183606557	0.08109810011	0.06592437798	0.1460334978	0.02712162691	0.0505672694	0.9714303742	2.116116265	1.50683119	1.194690157	0.8208939092	0.73233646	0.7268853837	0.4699395259	0.6058288017	1.4085502	0.9815811258	0.1655786167	0.1275830258	0.3183819549	0.1587107698	0.3512540037	0.2409107207	0.0567146283	0.0686605305	0.0231154551	0.0157325852
153.62555	152.93272	151.71933	152.63208	152.63101	153.03787	152.98985	154.41411	154.81831	156.99893	160.40668	160.37054	161.51112	162.72487	163.20337	163.01129	162.05453	165.06232	167.58025	165.05079	162.2436	162.54012	163.3136	163.28777	163.75284	164.61339	165.67857	164.76045	164.42116
0.4563678025	2.002686098	0.6094456489	0.02918024336	0.2489910201	1.0042645337	0.1850290225	0.5269320921	13.48130748	22.10005914	3.961373702	9.606652292	5.377204454	2.117403317	0.9932713569	0.6156562641	3.223712521	12.72834058	2.370830062	7.402688224	0.12702096	13.69651277	4.22878096	4.762127773	12.86771687	4.781263292	4.237710445	1.124554293	4.085087746
0.4416802877	0.945689769	0.5119147541	0.1295164502	0.3258003801	0.6505647894	0.2803741364	0.1063922204	2.316669821	2.907278912	1.224887685	1.890042087	1.415418422	0.8863016202	0.6069610231	0.7689542724	1.095801038	2.115434331	0.9104482025	1.875672846	0.2191881919	2.228093233	1.243514543	1.318807035	2.143635712	1.310917266	1.258140814	0.6478088552	1.24456865
153.13533	151.81546	152.65639	152.96364	153.24404	154.13603	153.51718	154.34946	157.31335	158.96458	160.4819	161.80142	162.01399	163.09328	163.3738	162.66073	163.33476	167.20833	167.94714	162.43852	162.82466	165.26959	165.40225	165.54834	167.25943	166.68544	163.59636	163.86567	162.6136
0.8421604089	0.3955150116	0.6244578321	0.01529684936	0.01311794116	0.0092217609	0.0094439524	0.0098922916	1.384505222	7.482522576	3.26922561	2.784159216	1.73188232	1.176617478	0.66290644	0.2393851329	0.2654722576	2.078412389	1.375800658	0.0046949904	0.0504721156	0.0003837681	0.0010400625	0.0061371556	0.0064915249	0.0131239936	0.0069956496	0.0002053489	0.04662496
0.1342464858	0.2082244224	0.1025508197	0.08090034679	0.07479919023	0.0523409504	0.0533424561	0.06447974088	0.7424127705	1.691663575	1.1127454	1.020725515	0.803277788	0.6606894871	0.5031668697	0.2999080544	0.314458346	0.8548295286	0.6935075685	0.04215691378	0.1361672817	0.01178345865	0.01959172341	0.04734392941	0.04814748416	0.06688105516	0.05111844518	0.00875381796	0.1315270836
95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
85.52	85.5	85.51	85.4	85.71	85.72	85.54	85.96	86.35	87.11	88.19	88.28	88.22	87.82	88.08	89.6	88.35	90.14	90.1	90	91.61	91.9	91.82	92.33	94.06	93.82	92.74	95.01	94.26
85.4469	85.52543	85.49097	85.47305	85.563995	85.60317	85.472855	85.936646	86.325485	86.89079	87.743904	87.86541	87.87129	87.791935	87.88091	88.927886	88.16909	89.52839	89.64483	89.70002	90.84214	91.2406	91.33048	91.87934	93.09074	93.288376	92.443184	94.29126	93.848335
0.005343461	0.0009486849	0.0003621409	0.00533630325	0.021317460	0.013649249	0.0045084512	0.0001783231	0.0006009852	0.0480530241	0.1990016412	0.1479094681	0.1215986641	0.0139393442	0.0396368281	0.4616028162	0.0327284281	0.3740667921	0.2070887049	0.0890680004	0.5890389706	0.43480836	0.2396295304	0.2031899868	0.9394649476	0.3994883774	0.0890973786	0.6540603876	0.1604680722
0.08547708138	0.0297426906	0.02225470705	0.08553864169	0.1703476841	0.1362925805	0.0784664073	0.01553894008	0.02839027215	0.2516473424	0.5058351287	0.4356479384	0.3952731807	0.1344397632	0.2260331517	0.7500156825	0.2047851387	0.6785112048	0.5050721421	0.3333111411	0.8361836044	0.7175190424	0.533130037	0.4882118488	1.930466913	0.6729127768	0.3200517576	0.8512158615	0.4387336831
85.29717	85.18499	84.894264	85.253914	85.088585	85.31556	85.160965	85.44039	86.12797	86.01584	86.68722	87.23169	87.80807	87.815704	88.09888	88.19758	89.22771	88.595116	89.33282	89.13614	89.8381	90.34941	91.254616	91.75923	92.44374	93.703064	92.96885	93.20941	94.42506
0.04950532089	0.0992313001	0.3791308217	0.02134111194	0.3861566202	0.1535717136	0.1436675312	0.2597023521	0.6492973209	1.197842692	2.256347728	1.998953856	0.1696063249	0.0000184555616	0.0003564544	1.996781856	0.7703746441	2.386666573	0.5805651524	0.7462540996	3.13902901	2.404329348	0.3196590675	0.3257783929	2.612295388	0.8470612281	0.9523723325	3.256545068	0.02724488301
0.2605589336	0.3684327485	0.720074845	0.1710808859	0.7250264177	0.4718152123	0.4431084873	0.5929144852	0.2571279576	1.256411434	1.7040254	1.187483009	0.4669349354	0.0048918241 86	0.02143505904	1.585200893	0.9034465195	1.713871755	0.8514761376	0.9598444444	1.934177462	1.887257889	0.6157525594	0.818184772	1.718328726	0.2309765571	0.2467651499	1.899368488	0.175111394
85.37913	85.37588	85.39231	85.29572	85.60103	85.56684	85.43027	85.82668	86.194048	86.88262	87.725395	87.88702	87.86121	87.81884	87.87028	89.0215	88.2212	89.698505	89.85956	89.63128	90.889084	91.21901	91.27885	91.7751	93.12912	93.2067	92.36589	94.29794	93.77971
0.0198443569	0.0153809604	0.0133809361	0.0106667584	0.0118746609	0.0154157056	0.0120406729	0.0154554624	0.0243216502	0.0517016644	0.215857936	0.1544332804	0.1147786641	0.0404683456	0.0439824784	0.33466225	0.0195884144	2.281424945	0.1939873936	0.1395641384	0.5184230403	0.4637473801	0.2930596225	0.307914101	0.8665375744	0.508796891	0.1253939921	0.5070294436	0.2300784841
0.1647217025	0.1450525316	0.1378330254	0.1209367581	0.1271380236	0.1448436771	0.1282791676	0.1445422339	0.1806606327	0.2610262886	0.526822769	0.4451517898	0.384026565	0.2290594306	0.2381617257	0.6456473214	0.1457838144	0.588523408	0.4888346252	0.4096888889	0.7859578649	0.7410119695	0.5895774341	0.6009964259	0.9696661708	0.7594761499	0.3818309252	0.7494579918	0.5095374496
95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
113.16	112.88	112.58	112.36	113.1	112.4	112.36	112.39	113.31	115.05	116.33	114.8	114.16	112.87	109.64	110.13	110.68	111.19	111.67	113.2	112.78	114.27	114.37	112.84	114.54	113.54	110.81	113.25	111.65
113.38504	112.99228	112.74139	112.52957	113.313576	112.53911	112.34077	112.489586	113.36661	114.79143	115.4713	114.78777	114.24967	113.180984	110.70529	110.51673	111.0317	111.34716	111.78466	113.074615	112.83812	113.945595	114.28951	113.07719	114.37137	113.52856	111.38626	113.89237	111.736335
0.0508439016	0.0126067984	0.0280467321	0.0287539849	0.04561479778	0.0193515921	0.0093697929	0.0099171371396	0.0032049821	0.0968584449	0.43388569	0.0901495729	0.0980766169	0.0967104826	1.134842784	1.144953329	0.123699289	0.0248992696	0.0131515024	0.0157213623	0.0033779344	0.105238604	0.0064795401	0.0562590991	0.0284360769	0.0901308736	0.3320755676	0.0248472169	0.0079033722225
0.1986588583	0.0994684208	0.1433558359	0.1909166963	0.1688381963	0.1237633452	0.0171145315 4	0.0860752736	0.04996028594	0.2247457627	0.5622339895	0.2106533101	0.07872284513	0.2755240342	0.971623192	0.3457096159	0.3177629201	0.1413436461	0.1026954419	0.1197641343	0.05153395992	0.2838934103	0.07037684708	0.2102002836	0.1472236773	0.01007574423	0.5200433174	0.13918768	0.07911777877
114.13149	113.939935	113.68377	114.10839	113.505455	113.28892	112.590195	111.89942	113.02813	112.84684	114.57485	114.51829	114.401215	114.17643	113.71631	110.74691	110.55401	110.786376	111.761855	111.9364	113.60616	113.19327	114.19014	114.21693	113.21719	113.29397	112.10983	111.20461	112.08942
0.9431928201	1.123250227	1.669816213	3.059667552	0.184353757	0.7900306924	0.8529897302	0.2406687364	0.0794506969	4.855235972	3.260551523	0.0703505241	0.0581846762	1.706759345	15.61630322	0.3730888561	0.0158734801	0.1613016374	0.2684375247	1.56685496	0.8525403456	1.165815873	0.0323496196	1.685936225	1.749826296	0.9653507609	1.679434565	4.183620252	0.2019783354
0.8585100742	0.938904146	1.158083141	1.956066876	0.3584924845	0.7907651246	0.2048727306	0.4354879091	0.2487600388	1.91521947	1.50876816	0.2453919861	0.2112985501	1.15746434	3.71790408	0.5546263507	0.1138326708	0.3612061443	0.6822566806	1.116254417	0.7325412307	0.9448936729	0.1572615196	1.220249911	1.154889122	0.2166901532	1.169506362	1.800803885	0.4028257591
113.21146	112.90963	112.58686	112.37721	113.148544	112.424965	112.40108	112.437836	113.34545	114.0009	115.80314	114.85475	114.192314	112.92966	110.51269	110.39886	110.87402	111.31764	111.73253	113.21449	112.78525	114.22544	114.37273	112.92612	114.62147	113.61259	110.99141	113.32444	111.620674
0.0025481316	0.0008779369	0.0009343396	0.0002961841	0.002385519935	0.0006232512 25	0.0016875664	0.0022882826866	0.0012567025	0.02223081	0.2775814596	0.0026997525	0.0010441945 96	0.0035593155	0.7615878361	0.0722856996	0.0378437604	0.0162919696	0.0039100009	0.0002099501	0.0001055525	0.0011855936	0.0009074529	0.0074166544	0.0056373609	0.0052693081	0.0329095681	0.0055413136	0.0006600142 78
0.04547543302	0.0262491141	0.005205187422	0.01531683873	0.04292130858	0.02221085409	0.03696105376	0.04256250558	0.03123858297	0.1295658279	0.4529012293	0.04769163763	0.02830588648	0.0528572694 2	0.7956595038	0.2441296649	0.1752981568	0.1147944959	0.0559534342	0.0128003536	0.002881716616	0.03699536186	0.002386989595	0.07832045374	0.07112799022	0.06393341554	0.1637126613	0.06573068433	0.02826600985

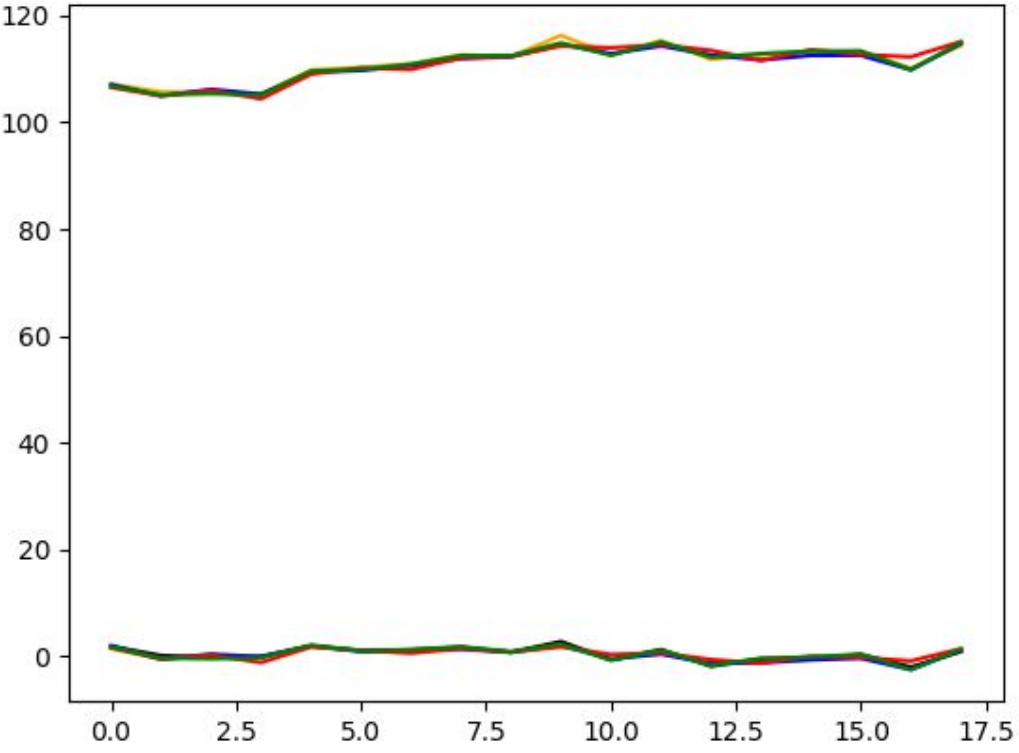
Appendix C - 7 day forecast preliminary results






175
150
125
100
75
50
25
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


80
60
40
20
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


120
100
80
60
40
20
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Appendix D - Links to Source Code, Market Data and Results